The MDriven Book

Table of Contents

What is MDriven

The MDriven Book

Introduction

Write the content here to display this box The printed book can be purchased on Amazon: https://www.amazon.com/MDriven-effective-business-control-information/dp/ 1729341098 As I have worked as a software developer and architect in Sweden for the last 20 years, I have seen a lot. I am not going to bore you with my historic reflections at all. That was my first attempt at an introduction to this book. Then, I realized that most of the things we have done with MDriven are very much anchored in the historic reflections of things we have experienced previously. So I guess what I need to say is more like this: Having worked many years as a software architect, I have seen a lot of things that change – but also very much that stays the same over time – a stable core.What is the stable core? The need to store and retrieve the information we are dealing with. The need to somehow display it for users and handle the users' need to update it according to the rules we want to enforce. This is the technology of any application or system and is what every system developer needs to deliver – using modern technology at the time of implementation. I will refer to this as system modernity. Furthermore, we need to understand the business information in the system and how the rules that govern the information's evolution and consistency protection work. This part I will refer to as the system gist. On the other hand, we all know that to make our software sell – to an external market or in-house users - it must be perceived as modern and cool - but in the software business, modern and cool change every year or so, at least on the surface. Thus, mixing the system gist down into a currently modern format that will be obsolete in a couple of years

The MDriven Book

seems to be something one should avoid. It would be better if the system gist could somehow be separated from the current modern implementation strategy – which we know will be less modern as time goes by. This book contains my suggestions for how we deal with system gist and system modernity. The MDriven Book - Next Chapter: Praise to UML

The MDriven Book

Praise to UML

Write the content here to display this box UML from Wikipedia: "The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system. It was created and developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software during 1994–95, with further development led by them through 1996." To fully convey my appreciation for UML, I must explain how I look at the world. Below, I define three areas that will help explain my reasoning: # Fashion: what goes together with what in a manner that is hot and sexy that people somehow crave without further need to understand it. # Modernity: ways to solve known problems. Tools and strategies have a modernity aspect. A more modern tool is not necessarily better – but it often is considered as such since the new tool has had the advantage of being created in a world that has more knowledge than the world that created the old tool. # System gist: anything that combines a series of ideas and actions in order to produce value. The gist of the system, in this context, is the system description stripped from everything that is either fashion or modernity as described above. Let me exemplify these definitions First example: A car manufacturer is very much reliant on fashion. How the lines of the car body appeal to the target audience is very important but almost totally based on feelings and soft aspects that are hard to measure. The modernity aspect of car manufacturing is important for the manufacturing process: what tools to use, how to apply industrial robots, and what third-party systems to include like anti-lock braking and airbags, etc. The system gist is captured in the design phase of

the car construction process. It involves all the inherited knowledge about what is important for cars in general and also some new things that are important for this car model in particular. Second example: A surgeon. Modernity provides important tools to diagnose a patient, like MRI. It also provides even better tools for fixing what is wrong, like minimally invasive surgery. The system gist for the surgeon is the knowledge of what and where to cut and how and why organs act as they do. It is important for a surgeon to be able to draw the intuitive line between gist and modernity. Having the latest tools will not be enough if you are not educated on what to look for and how to act upon what you find. As a patient, you will want a surgeon that masters both the gist and modernity and does not neglect one for the benefit of the other. When it comes to fashion, it is important for plastic surgeons because they too must have the main focus on gist and modernity or the patient will end up with a defective system. I know little about the field of surgery but I am sure that if I knew more, I would also see aspects of fashion in appendix removal procedures. It might be how high to cut and how to stitch the wound that has no immediate support in current science but feels right or looks good. Where is the software industry in this spectrum? The software industry differs from the two examples given by not having ONE fixed or slow-evolving system gist. The software industry is actually about producing new systems and as such, new areas that have a gist, modernity, and fashion of their own. The software industry is one Meta level up compared to surgery or car manufacturing. It is in this way not just a human activity like surgery but an activity of activities. This is what makes software development a field that will leave no other area of human activity untouched. Hundreds of new unique software systems are finalized every day

and they will resemble each other when it comes to modernity and fashion. What makes each of them unique is mostly their system gist. Look at what all the apps on mobile devices share: their execution environment, their use of the network, the interaction patterns, and widgets. Modernity and fashion is a time marker that makes it easy to guess the time of construction for a particular software system. The modernity aspect of software development is very important to be able to produce a well-behaving system. The fashion aspect of system development is very important to attract users and make the system intuitive to use. These two areas, reused over and over, also evolve at a rather high pace. Refined strategies – or as I call them – modernity aspects - on how to build software systems is a topic discussed endlessly in the developer community. It is easy for software developers to completely get lost in the modernity aspects. As they do, they will leave less room in their minds for system gist. When we get software developers that move cross-field and mostly work with modernity and fashion, is anyone taking care of the gist? What sets one software system apart from another is mainly its gist, not its modernity. UML is all about system gist. This is why we need UML or something that solves the same problems. We use UML to describe system gist in an easy, clear, and partly visual format without any possibility for alternate interpretations. UML is the most prominent way to handle system gist. For a software engineer, it is important to be able to quickly place arguments on design decisions in the correct category. * If it is about system gist then there will be facts to research in the domain of the system in order to make the best decisions. Is it about modernity – then it is important to analyze best practices from the development communities and consider the pace and gain of change to see which path to take. * Is it about fashion –

stand back and let the end users or market decide. Take a vote if they do not reach a consensus. Just as software frameworks – like Entity Framework, Hibernate, or the like – aim to help developers with modernity issues, there are manufacturers of generic software systems that aim to solve everything. These are meta-systems where you can somehow describe your system gist and then you would be done. Since a meta-system like this is a holy grail – the search for it engages many software companies. Of course, many claims are made that the grail is found, but is it really? And is there really such a thing as the holy grail in the first place? SharePoint SharePoint is a tool that I have seen used like this on a number of occasions. As SharePoint may have an appealing modernity and may be fashionable in certain populations it is an easy sell if it also lends itself to handle ANY system gist. As SharePoint is a software system it has a system gist of its own. One that may be described with UML. {{image:Gist_of_SharePoint.png|frameless|358x358px}} As you see the gist of SharePoint is tiny. In fact the gist of a Meta systems often are tiny. Alan Turing thought us in 1936 that one can build universal machines – a machine that can simulate all other machines. Universality of machines is reached very fast as Stephen Wolfram showed us in his book "A new kind of science (NKS)"-2002 where he suggest that a cellular automate of only two state and three colors is universal. Alex Smith later proved this in 2007. In light of this it is not surprising that the system gist of SharePoint is Universal – so it can be used to implement any and all systems conceivable. This may seem fantastic – but just because something is possible does not necessarily make it a good idea. It may for example be possible to build anything by gluing grains of sands together under a microscope – but not practical or economical defensible. Folding down the system gist into ListDefinitions in

The MDriven Book

SharePoint is not the best way to treat the gist. In that format it is not easily evolved and maintained. It is however possible, I do not question that. My opinion is that the best way currently available to describe system gist is UML using the language of the domain. Refrain from building Meta systems. Accept that each area of human activity has its own gist that deserves its own UML description. When each area has its own gist clearly described in UML it is easy to maintain and evolve. Free from modernity and fashion issues. I wish all developers would be aware of the three different areas of gist, modernity and fashion. It is my belief that we limit our ability to develop everything due to lack of focus and lack of discussion on system gist. Having a language for the gist opens up for discussions and thinking that helps development in all areas of human activity. The MDriven Book - Next Chapter: What if UML was forbidden See also: UML School

The MDriven Book

What if UML was forbidden?

Write the content here to display this box If UML or other structured ways to define system gist were forbidden – what would happen then? Maybe UML need not be forbidden – the effect would be the same if UML simply were not used. Well, the system gist still exists – even if it is not explicitly documented. It must be extractable from the source code of any running system since the system is a transformation of the system gist, no matter how that gist was captured in the first place. Maybe the system gist is held in documents outlining requirements or prose text that describes scenarios the system should solve or support. Maybe the developers were obligated to write other prose documents that act as the documentation of the produced software. Documenting software is boring compared to coding for any developer. Code can be compiled and type-checked so that the bugs can be removed. Code can also be executed and further tested to ensure that the ideas we wanted to cover are covered. Documentation does not work that way. Most likely, any existing documentation is filled with bugs beyond belief since the verification process lacks compilation and testing. Since developers suspect this, developers seldom trust and consult the documentation of existing software systems. Instead, they tend to go to code. If the only option is prose documents, they are probably correct. Even if the developer finds information in the code – it is important to remember that the code is just the original developer's interpretation of the requirement – and this need not be equal to the requirement. Software tools and libraries may follow other rules, but, when an experienced developer is confronted with a software system, they very seldom expect that the documentation is complete or correct. Prose

documentation of system-gist is, for the reasons stated above, rarely used for anything except making the stakeholders feel a bit safe – like life jackets on an airliner that flies over land. It was never intended to help anyone, but the stakeholders want it so we provide it. Another way I have come across to protect and keep the know-how that is the system gist is what I would describe as "invest in the team". Let skilled and motivated software developers solve things with their talent and memory. Let the collective team memory be your know-how and documentation of the system gist. This is the common way for most businesses that produce software that I have come across. I argue that this is not a strategy. It is an abdication of ownership and control. Developers might not see this as a problem, at first, since they take pride in the trust management places in them. Still, I have seen many cases over the years where this strategy created a chasm between the ones that know (developers) and the ones that make decisions (management). It usually ends with a collapse that benefits no one. Since developers are now left on their own to decide where the resources should go – into system gist or modernity or maybe into fashion, they will soon lose the management's trust since management lacks control and does not understand why the developers chose to prioritize the way they do. Nevertheless, this is commonly how small and midsized businesses handle their software investments today. Let the code speak for itself – there is no other representation of the ideas within the software than the code itself. The same code strongly flavored by the team members takes on what is modern and what is fashionable this year – or was last year. The MDriven Book - Next Chapter: Luckily UML is Not Forbidden

The MDriven Book

Luckily UML is NOT forbidden

Write the content here to display this box For prose document writers and nothing but code cowboys, there is a missing link – a way to describe system gist separated from the flavor of the year implementation method in a format that is not as flimsy and open to interpretation as prose documents. What I propose here – and what many have proposed before me – is that we can document with models. The models are more descriptive than prose text, leaving less room for alternate interpretations. At the same time, models are less complex and easier to read than code written in the flavor and style of the year. The model is then home to the system gist. Here it can be understood, discussed, criticized, and evolved long before it has taken the expensive form of implementation code. What is somewhat new in what I say is that models can be compiled and executed just like code. We can then focus on using the current modern technique to execute the model. This way, the modeled solution can have a much longer lifespan than the user interface or delivery method heavily subjected to modernity and fashion. Models can cover the uniqueness and true solution of your software. A model executor brings your model to life in a specific environment. When one model executer goes out of style and something new is requested by the market, we do not rewrite the system gist. We create a new model executer and feed it the same model. MDriven is the latest model executer we have created – but we have been involved in making many. A model that was executed with Delphi 1999 (BoldSoft) can be executed with c# MVC5 or with WPF (CapableObjects) today. It was executed with Windows Forms in 2003 (Borland, Embarcadero), with Silverlight in 2007, and with ASP.NET in 2005. No doubt, there will be

new model executors when modernity so requires – and sure enough, MDriven Turnkey that brings any system gist to AngularJS is currently available. One key to a good investment in software is to avoid entangling things that change for different reasons and with different intervals and speeds. Your system gist changes and evolves along with the business it supports. The modernity of the solution changes by market forces no one can control, but everyone must adapt to it. I propose we keep these two areas apart so that they do not get confused as being the same problem. I define Model-driven development as developing a system gist in its own machine-readable format. Build a software machine that turns the system gist into a complete software system and fulfills the required modernity aspects. Giving such a machine a descriptive name: ModernGistExecuter – we at MDriven call our implementation MDriven Framework. The MDriven Book - Next Chapter: What is not to like

The MDriven Book

What is not to like?

Write the content here to display this box Having worked with model-driven development for the last 20 years, I have been surprised many times by how much resistance we have met. It is not like other developers are indifferent or do not care. Many do care – but the reactions are surprisingly often skeptical and negative. We have tried to take this as an indication that we are on to something relevant and beautiful. Big and disruptive shifts never come without agony and pain. The suggestion is that it is a defense mechanism that kicks in. People are afraid that model-driven development will change their current setup – and resistance to change is natural and triggers unconsciously. I do not believe in the anti-change theory. I think it is a simple case of not correctly separating the different issues at hand: modernity versus gist. Modernity and Gist Developers know that everything changes. Experienced developers have been left stranded with abandoned techniques and products throughout their careers. It is not a good feeling nor is it good for business or the credibility of the developers. No one likes to be forced into change by external influence, but a product that has lost new development and support must be replaced. This has led most developers to a minimalistic approach to using products. Minimalistic or gigantic – trust only the really big companies in software like Microsoft or Oracle – and things that are transparent like open source and simple tools. The problem with traditional development that blends and mixes modernity with system gist is that change hits so hard. Everything must be rewritten once a technology change is required. If the gist changes a lot – rewrite. If modernity changes a lot – rewrite. If the fashion changes heavily – rewrite. There are more reasons to

abandon made investments than what any investor dares to think about. All this is because of the mix-up of the three different areas. If we separate the gist from modernity, we will find that most changes in technology leave the gist untouched. All different types of gist will be handled by many different approaches and technologies over their lifetime. For this, we can plan from the start. Since few have had the opportunity to try this in real life, few know the benefits it brings. Separating the gist from modernity protects that part of the system from the IT wind-of-change that is always blowing at hurricane strength. At the same time, free up the modernity area to change without having to change all the gist stuff at the same time. Both areas will win by keeping this separation. As an information architect or developer of a domain work organization, the gist is the most interesting area. It is the theory and motive. This is also where the structural capital of the enterprise resides. Having this documented in a useful and actively maintained format is very attractive to any business. For classic software architecture, Modernity is extremely important as it ties into the projected lifespan of the system, maintainability, how hard it is to build, usability concerns, security, efficiency, and overall investment sanity like "our maintenance burden must not be too disparate". However, for a business wanting to build in-house systems to become more automated the modernity aspect is still important but they should put system gist in the front seat. It will be good for the whole business to structure their knowledge of whom they are and how they do things. I agree that if you use no tools to manage your gist, it is easier to just mix gist and modernity and let it stew. Then sit back and wait for the inevitable rewrite need that forces you to do it all again. I propose that this is the wrong approach and an approach that is not very smart or efficient.

The MDriven Book

Looking back on many discussions over the years, I now believe that a lot of developers and software architects do not separate the system gist area from modernity with enough clarity. Many software developers also focus mainly on modernity issues – after all, as a professional developer, this is what you can reuse for different clients that all have different gist. I believe that this limits their ability to produce robust long-term systems. I suspect that this is part of the explanation for why software development has too low a success rate. It can also explain why legacy ERP systems that are so far from modernity that it hurts still have an appeal in the industry. I believe we must be vigilant to correctly identify modernity and fashion arguments when solving system gist problems. If not, people will wrongfully let modernity or fashion arguments color their gist and by doing so, confuse both other developers and stakeholders. This book is primarily on how to use MDriven to handle and manage gist and secondarily, how to use MDriven to help you manage modernity. I will show how the modernity issues are managed with standard techniques in Visual Studio with no limitations or assumptions – so that you are ready for all new modernity requirements that will be sailing up. Lastly, this book will cover how the gist and modernity offered by MDriven Framework are easily dressed up with the current fashion. The MDriven Book - Next Chapter: What is next

The MDriven Book

What is next

Write the content here to display this box I propose and will show examples of how you can maintain the gist and idea of your software in a model – and how you can apply existing or create your model execution engines to act as modern and current delivery mechanisms for your solutions. The steady pace of shifts in the market's perceptions of what is new and cool often paralyzes companies from building support systems and with them, control their information. The main problem is that most techniques today mix the two different problems of the system gist and modernity. Entangling these two problems makes solving them as one hard and risky and something that many businesses will avoid. To summarize: Model centric – describe the system gist in a model that uses a model executer of correct modernity level to bring it to life. The model is then the documentation, the know-how, and the gist of the system– invest heavily in this. Put modernity and fashion in the model executer and into the user interface and delivery –invest in this too but keep in mind it may change soon. Make sure you involve management in the system gist governance so that they are in control and can make informed decisions. The series of tools and strategies presented here reduces the effort needed for businesses to take control of and own their information. It does so by introducing a clear separation between system gist and modernity. I believe that companies that own and control their information are better equipped to compete than companies that are clueless about it. A no-brainer, of course – but still – most companies lack a good and deep understanding of their information. The few that have control often spend too much on resources, trying to evolve system gist and modernity in one

complex and risky process. It is the aim of the tools and strategies presented in this book to show new ways to produce and maintain domain-specific software support systems – and to vastly reduce the costs and increase the quality and speed of how to produce and maintain them. It is not magic – it is just a matter of raising the abstraction level a bit and refraining from entangling problems that are much better solved separately. The MDriven Book - Next Chapter: Information design

Information design

The MDriven Book

The Information
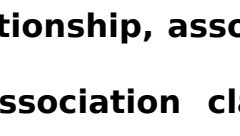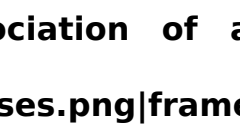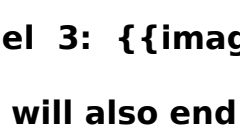
Write the content here to display this box I claim that for understanding a business, you should understand its information – what it deals with – the information it creates while it produces its product. Others may argue that the processes are the most important part – but I differ. If you know the information first, you know – or can figure out - what processes must be present that create this information. If you just know the processes, you still do not really know the information. As a software architect, you can use the phrase "follow the information" in the same sense as detectives of crime use "follow the money". You will find the truth this way. When you follow information, it is easy for all to see if it is valuable information or not, but when following processes, you track work that is performed currently. Suppose you found an unneeded process step – how will you know that it is unneeded? You cannot get a correct answer from the ones performing it now – since they are biased that it is important - and no one else will have enough information to really know. If you learn the important information first, you can easily see what the process step at hand does to that information. It should evolve the information in some way. If it does not create or change information in a valuable way the process step is unnecessary – for this business at least. The Information There are many ways to describe Information. My recommendation is UML – the Unified Modeling Language. UML contains a set of rules – and it lets you describe everything you need without further need for interpretation. UML is the core defining the models available in MDriven. This book will use UML extensively. The MDriven Book - Next Chapter: Short introduction to UML– class diagram

The MDriven Book

Short introduction to UML- class diagram

Association classes

Write the content here to display this box Associations define the relationship between Classes. Whenever you need additional information on that relationship, association classes will come in handy. Model 1: {{image:Model 1 Association classes.png|frameless|377x377px}} Even if the Association class is often used for many to many relationships, you can use them on the association of any cardinality. Model 2: {{image:Model 2 Association classes.png|frameless|349x349px}} OR-mapping (the process of taking an object-oriented model (a standard UML class diagram) and transforming it into a relational database schema (tables, fields, primary and foreign keys)) will turn this model into three tables; one to store Person, one to store Flight and one to store Booking. If you had not used the association class, OR-mapping would still create three tables due to the many-to-many associations. The third table would store two foreign keys, one to identify the Person and one to identify the Flight. The third table would implicitly be named, if you did not explicitly give it a name, to PersonFlight or FlightPerson. This table the DB-guys often refer to as a link table. The funny thing is that modeling this another way will give the same OR-Mapping result: Model 3: {{image:Model 3 Association classes.png|frameless|321x321px}} This will also end up in the database as three tables where Booking points out Person and Flight with one foreign key for each. So for a DB-centric guy, this is the same... To an OO guy, this is NOT the same. What is the Difference? The rules that association classes adhere to in any well-behaving MDD framework are these: # Lifetime control: The booking cannot be explicitly created. It is created as a consequence of associating a Person with a Flight:
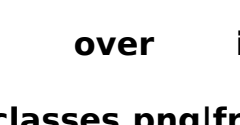
aPerson.Flights.Add(aFlight). It is destroyed automatically whenever the association is removed: aPerson.Flights.Remove(aFlight) # Uniqueness: In UML, one instance must be unique in the relation; you cannot add one person to a flight twice. This way, using the association class has effectively given the UML reader the information that a person can only be one passenger at a time - not two. Whenever you see the need for lifetime control and uniqueness, use the association class. It will help the reader and the developer – the DB guy will not know the difference - however, they seldom do. (Side note: Entity Framework does not currently support association classes. They argue there is no need since the Model2 can be replaced with Model3 (No surprise since they are DB guys.).) Adding a Link Object Look at Adding a link object for an example of EAL to help you easily handle adding a link object. The MDriven Book - Next Chapter: UML Inheritance

The MDriven Book

Inheritance

Write the content here to display this box With the ambition to make it easy for people to benefit from object-oriented approaches using MDriven and MDriven Designer, I will give a quick introduction to UML inheritance. * UML inheritance differs from "I get you stuff when you die." It is also different from "Oh, look, that kid really looks like her Mother." * UML inheritance is this: "A child class has all attributes and associations that a parent class has, and the child also has attributes and/or associations of its own that the parent does not have." In other words, UML inheritance is "specialization" and "generalization"; a child class is a "specialized" version of the parent, and a parent is a more "generalized" definition of the child class. * UML inheritance is the same as OO inheritance (Object-oriented inheritance). * UML inheritance will allow you to inherit the properties of multiple parents – but very few OO languages allow for so-called multiple inheritances (c++ does, c# & VB.NET does not, and since ECO focuses on the latter languages, ECO does not support it either), so I will not mention multiple inheritances again in this post. This means a class can have only one parent class (or no parent class, but never many parents). An Example Fruit. Fruit is a pretty generic class. If we think of specializations of fruit, we will find Apple, Orange, Pear, Banana, Pineapple, etc. {{image:FRUIT MODELS.png|frameless|278x278px}} The lines ending with the big arrow are called a Generalization-association, meaning that if you follow it, you get something more generalized of the class that you leave. If you follow it in the other direction, you get the opposite of generalization – specialization. You will notice that in MDriven when you add generalization associations, the

class's superclass is updated in the object inspector. {{image:Properties fruits.png|frameless|314x314px}} Superclass is a more correct UML term than "Parent class." Instead of "Child class," the correct UML terminology is Subclass. So I will use Super- and Subclass from now on. Why is Inheritance So Useful? The obvious benefit of inheritance is the ability to introduce common properties that all fruit has in one place. If there are properties that all fruits have, they will go into the Fruit class rather than defining them over and over in the subclasses. {{image:Model fruits + subclasses.png|frameless|377x377px}} The true power of inheritance is that it resembles how people reason and think. As humans, we always generalize. Our language and communication depend on it. This fact is the reason for some bad things in society – prejudice where we jump to conclusions based on earlier experience or hearsay – and some good things, like instantly knowing how to use a door knob even if we have never seen that particular type of door knob before. With the model we now have, we can see the benefit that strong types give. Our code will now look like this: 1: Country malaysia=new Country(this.ServiceProvider()); 2: 3: Apple apple = new Apple(this.ServiceProvider()); 4: Orange orange = new Orange(this.ServiceProvider()); 5: 6: apple.GrowsInTheseCountries.Add(malaysia); 7: orange.GrowsInTheseCountries.Add(malaysia); 8: 9: foreach (Fruit fruit in malaysia.ExportsTheseFruits) 10: { 11: if (fruit is Apple) 12: { 13: // Do apple specific operations 14: } 15: else if (fruit is Orange) 16: { 17: // Do orange specific operations 18: } 19: } The MDriven Book - Next Chapter: Polymorphism {{Edited|July|12|2024}}

Polymorphism

Write the content here to display this box Polymorphism is a fancy word for an important concept: poly = many, morph = shape => many shapes. In our example, we use polymorphism in the association from country to fruit - namely, a resulting list that can contain different subclasses of fruit such as apples, oranges, etc. Polymorphism allows us to operate on stuff we do not know much about. Check this out: {{image:Fruit model 2.png|frameless|328x328px|link=https://wiki.mdriven.net/index.php/File:Fruit _model_2.png}} I add a method on Fruit that I make virtual: {{image:Properties 2.png|frameless|349x349px|link=https://wiki.mdriven.net/index.php/File:Prop erties_2.png}} I can implement this to return a default value on Fruit and override it on the subclasses that should return a different value: {{image:Overrided subclasses fruits.png|frameless|355x355px|link=https://wiki.mdriven.net/index.php/File: Overrided_subclasses_fruits.png}} 1: public partial class Fruit { 2: public virtual bool HasSeedsYouNoticeWhenYouEat() 3: { 4: return true; 5: } 6: } 1: public partial class Banana { 2: public override bool HasSeedsYouNoticeWhenYouEat() 3: { 4: return false; 5: } 6: } Having this, I write code that goes over a list of Fruit and ask if the fruit HasSeedsThatYouNoticeWhenYouEat like this: 1: List crapfruit = new List(); 2: List okfruit = new List(); 3: foreach (Fruit fruit in malaysia.ExportsTheseFruits) 4: { 5: if (fruit.HasSeedsYouNoticeWhenYouEat()) 6: crapfruit.Add(fruit); 7: else 8: okfruit.Add(fruit); 9: } If you are still with me, I also want to mention the

concept of "Abstract". When we have a model like the one above, think of the Fruit class as being abstract – meaning that having an instance of a fruit (a real fruit) that is of type Fruit should not be legal. A fruit-instance must be one of the subclasses; it can be an Apple, Pear, Orange, Banana, or Pineapple (in our model) but never just "Fruit." In Object Orientation terms, Abstract means that the compiler will treat any attempt to create an instance as an error. It is an error because the developer that defined the class never intended it for direct use; it was designed as an abstraction or generalization of a set of subclasses. My recommendation is to always treat classes that have subclasses (aka superclasses) as being abstract. In the Fruit sample above, this might be obvious, but remember this when you classify your domain where it might not be so obvious. The MDriven Book - Next Chapter: Composite and Aggregate and what they imply

The MDriven Book

Composite and Aggregate and what they imply

Write the content here to display this box Associations between classes are easy enough to understand. Car has 4 wheels. {{image:Car4wheels.png|frameless|307x307px}} But in the information system we build, it may be obvious that a Car owns all its Wheels; the car and all its wheels can be looked at as a complete entity of their own – a composite. If the Car is scrapped, the wheels are also implicitly thrown away. A Composite in UML is created by decorating the association with a filled diamond: {{image:Diamond.png|frameless}} For any well-behaving MDD tool, this decoration should imply a cascading delete of all associated wheels when the Car is deleted. A composite also signals to the UML reader that the Car and Wheels are created at the same time and may not be meaningful on their own. So some will argue that the model should be changed from "0..1 Car" to "1 Car" (i.e wheel must always belong to exactly one car): {{image:Diamond 2.png|frameless|124x124px}} On the other hand, this might not be the best idea for the domain we are modeling now. If the system we build is one that describes a garbage sorting facility, we may want to say: "Yes, a car often has wheels, and the car and its wheels can be looked at as an entity of their own (a composite), but we sometimes want to take this composite apart and treat the parts separately". If this is the case, the aggregate decoration can be used: {{image:Aggregate.png|frameless|143x143px}} The aggregation symbol signals to the UML reader that the connection between Car and Wheel is "strong and common" and that "Car owns wheels" is more appropriate to the domain than "Wheel owns Car" (this also applies to the composite symbol). A
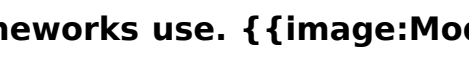
well-behaving MDD tool should probably prohibit the deletion of a Car until the Wheels are gone - so that the scrap yard guys do not delete a car while the valuable wheels are still on it. To sum it up: Composite is stronger than Aggregate. Both symbols imply that the domain sometimes looks at the classes as a bigger unit. The symbols help the UML reader understand the larger compositions in the domain. The symbols imply specific destruction behavior to well-behaving Model-Driven Development Frameworks (MDD-Frameworks). What Does MDriven Do Each Association end in MDriven has the "Delete Action" property: {{image:ECO Delete action.png|frameless|299x299px}} The DeleteAction can be set to one of these values: {{image:Default.png|frameless}} {| class="wikitable sortable" |DeleteAction on the Wheels Association |Calling Car.AsIObject().Delete() |- |Allow |will work even if you have wheels left on the car, but the wheels will be left dangling |- |Prohibit |will not work as long as you have wheels on the car |- |Cascade |will delete any remaining wheels |- | |If the association end is Composite, treat as Cascade; if the association end is Aggregate, treat as Prohibit; if the association end has aggregationnone, treat as Allow |} The recommendation is to leave the DeleteAction on and use the Aggregation setting to control the delete action AND help UML readers understand the domain. Deleting Wheels Without the Car Assuming that your context is the Car: this.Wheels.delete would delete the wheels. Note! Not only unlinking them. If you would like to unlink them, do: this.Wheels->clear Note: if you use the aggregation of type Composite, you would create "floating" objects that have a validation error of not having the Car association set. The MDriven Book - Next Chapter: Derived attributes & associations

The MDriven Book

Derived attributes & associations

Write the content here to display this box Derived attributes have been in MDriven for ages. To an SQL guy, derived attributes resemble calculated fields. Derived attributes are like calculated fields that subscribe to all the values they are calculated from. Whenever data is seen by the expression and changed, the derived attribute is marked as out-of-date. The next time you or your UI tries to read it, the attribute is re-evaluated. The key thing with derived attributes is that they are NOT calculated each time you access an attribute. If such were the case, the performance would suffer. It is calculated (or derived) as few times as theoretically possible - only when read the first time after a change of anything that the derivation expression looks at. The concept of derivation relies on the concept of subscription; everything in your domain layer can signal whenever it is changed (the publisher pattern), and subscribers subscribe to publishers to detect these changes. Although this is a different story, I must mention that publishing events to catch changes is a key difference between the ECO approach to implementing a domain layer and the POCO way (plain old c#-objects) like NHibernate and some other frameworks use. {{image:ModelOrder+Customer.png|frameless|395x395px}} Having a model like the one above, I can create a derived attribute on the Order that calculates the total shipping cost by checking the products ordered and the customer's country in OCL: self.OrderItems->FilterOnType(ProductThatNeedsShipping).Weight->sum * self.Customer.Country.CostToShipHerePerKilogram I do this by creating a new Attribute, setting the AttributeMode to Derived, and filling in the DerivationOCL. Derived attributes can be used in other derivation expressions

so I can make a derived TotalCost without repeating the definition of the Shipping cost: self.OrderItems.Product.Price->Sum() + self.ShippingCost The expression of derived attributes can be expressed with C# / VB.NET / Delphi.NET code instead of in OCL (MDriven for Visual Studio). Do this by setting the AttributeMode to Derived as before but leaving the DerivationOCL property empty. Generate code and inspect the Order.eco.cs file. At the bottom of the file, you will see these stubs: Code Snippet /// This method is called when ShippingCost needs to be calculated partial void ShippingCostDerive(ref string res); /// This method is called when TotalCost needs to be calculated partial void TotalCostDerive(ref bool res); Never change the files named something.eco.cs. Instead, implement these partial methods in the file something.cs (or vb or Delphi). So in this case, open up the Order.cs file and enter: Code Snippet partial void ShippingCostDerive(ref double res) { res=0; double totWeight=0; foreach(OrderItem i in OrderItems) { if (i is ProductThatNeedsShipping) { totWeight += (i as ProductThatNeedsShipping).Weight; } } if (Customer!=null && Customer.Country!=null) res = totWeight * Customer.Country.CostToShipHerePerKilogram; } Exception Please note that because of how time flows, you can't subscribe to time, like DateTime. Now, if it could, it would always need updating and lead to an infinite update loop. See also the Default String Representation and asString which are not subscribed. What to Remember All you need to remember is that derived attributes can be defined in code or in the model with OCL. Derived attributes are efficient and always return the correct up-to-date result. Associations can also be derived in the same way as attributes. In the example model above, the Shipping cost is out of date if: you add a new order line or change the

customer, or the customer changes country, or an already picked product that needs shipping gets an updated weight, etc. It covers every and any change that affects the calculation as long as that change is part of your domain layer. I hope that you see the positive effect this will have on your UI implementations – showing the shipping cost and seeing it update as you change anything it depends on by one central definition far away from the UI. Using derived attributes and associations cleans up your code and consolidates central definitions to single points and thus, greatly reducing maintenance costs and efforts. Having the definition in OCL also makes the derivation a part of the documentation – the model – rather than just the implementation. See also: * Derived settable attributes * Derived settable associations C# and the AutoSubscribeService In versions of the product ECO, you needed to explicitly explain to ECO what ECO should subscribe to stay updated if anything changes. Since the introduction of the IAutoSubscriptionService, this is no longer necessary. The IAutoSubscribeService deserves a chapter of its own, but here is a brief overview:The caller of your code derivation has a subscriber and starts an Auto subscription session. An Auto subscription session is notified about all access to the domain layer as long as the session is in effect. All access is inspected and builds up the description of what ECO needs to subscribe to in order to know when to mark the attribute out of date. The session is ended after your code derivation returns.Conceptually, this is what happens: Code Snippet using (this.AsIObject().ServiceProvider.GetEcoService().StartSubscribe(subscriber)) { double res; this.ShippingCostDerive(ref res) } The MDriven Book - Next Chapter: UML – State machines
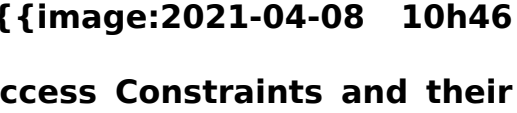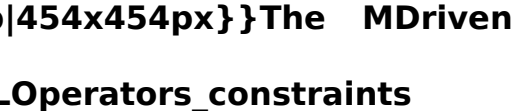
UML - State machines

The MDriven Book

Constraints

Write the content here to display this box There are other ways to introduce business rules in the model than using state machines and guards. You can use Constraints. The model already has several implicit constraints from the cardinalities of the association ends. If you have cardinality 1..4 and you have zero objects in that relation – then you have a broken constraint. Define Constraints You may also define your own constraints: {{image:Model car constraints.png|frameless|397x397px}} You can choose if a broken constraint (a constraint that evaluates to false) should be treated as Information, Warning, or an Error to the user. Delete Constraints You can also define the constraint as being a delete constraint only: {{image:Constraint wrappper.png|frameless|437x437px}} This way, you have explained at the model level that the domain does not consider it acceptable to delete a Car-object as long as we have the deposit unless it is in state Scrapped. The delete constraints will be checked when Deleted by MDriven. As a result, the Delete operator is executed on the class. Other things that are checked when the Delete operator is run are the Business Delete Rules that exist on all association endpoints: {{image:Business delete rule.png|frameless|438x438px}} As modelers, we should decide the best rule for each association end. In this case, is it acceptable to delete a Brand if Cars are left in the AllCarsOfThisBrand association? No, I think not. I am setting it to "MustBeEmpty." The association is in the other direction on the other hand. {{image:Business delete rule need not be empty.png|frameless|438x438px}} I set that to "NeedNotBeEmptyNoWarning" – because deleting a car object is okay even if

it has a brand. Constraints Evaluation in OCL If you want to evaluate constraints in OCL, use OCLOperators constraints, for example. Usage of Constraints Constraints automatically show up in ViewModels - but you may opt them out on a per-ViewModel level: {{image:2021-04-08 10h46 32.png|none|thumb|484x484px}} You can also access Constraints and their "brokenness" via OCL Operators brokenConstraints and Constraints: {{image:2021-04-08 10h48 43.png|none|thumb|454x454px}}The MDriven Book - Next Chapter: The ViewModel See also: OCLOperators_constraints

The ViewModel

The declarative ViewModel

#REDIRECT Documentation:ViewModel

Taking it further still

Write the content here to display this box If the cost of creating and maintaining a ViewModel were high, fewer ViewModels would be created. Our mission is to reduce the cost of creating and maintaining them. WPF is a declarative way to describe the UI. This means that the same basic look-less components like TextBlock, TextBox, CheckBox, Combobox, and Image, etc will be used again and again and they will be given a look by an external style or template. What if we use this fact to provide some basic rendering/placing hints for the ViewModel columns? We could then use those clues to spill out the correct look-less control in the intended relative position; we would not need to mess about with XAML every 5 minutes. This is what the ViewModel-Editor looks like without rendering hints: {{image:View Model Further still.png|frameless|448x448px|link=https://wiki.mdriven.net/index.php/File:View_Model_Further_still.png}} And this is the way it looks like when I have checked the "Use Placing Hints" checkbox: {{image:VM UI.png|frameless|453x453px|link=https://wiki.mdriven.net/index.php/File:VM_UI.png}} Given the extra fields for "Presentation", "Column", "Row", "Span" etc, I can work the ViewModel – review to look like this: {{image:MV UI 2.png|frameless|450x450px|link=https://wiki.mdriven.net/index.php/File:MV_UI_2.png}} Now, I really need to stress this so that I do not get misunderstood: We are not designing the presentation here at all. We are describing what data is available, which values are valid, possible selection lists of data, and, if the UI designer wishes to take notice of it, *hints* as to how to arrange the controls in relation to each other - which happens to give

the option of generating the user-interface automatically by whatever front end is currently in fashion. This is all the natural information we have in mind while designing the ViewModel. Having a ViewModel with placing hints, you can add a ViewModelWPFUserControl to your form with just one row: And the result is: {{image:VM UI The game.png|frameless|392x392px|link=https://wiki.mdriven.net/index.php/File: VM_UI_The_game.png}} Remember that these auto layout controls also adhere to external set styles. Having the ability to get simple UI automatically derived from the ViewModel, placing hints lowers the effort to produce and maintain. Experience has shown that a lot of the administrative parts of your application are left automated so that more time can be spent on the signature screens that are most important for your users. Note: the #Span.Savebar is a tagged value feature that can be helpful on your ViewModel. The MDriven Book - Next Chapter: What an Action can do See also: *Seeker view *UIOverride *Development info in runtime *Edit in Grid *ViewModel access and security *Custom controls in ViewModel-aided Views

What an Action can do

ExecuteExpression

Write the content here to display this box Actions can do a couple of things. ExecuteExpression {{image:Can do action -2.png|frameless|466x466px}} The ExecuteExpression is an Extended-Action-Language expression that is rooted in some context. For GlobalActions, this context is null. What you can do is limited to expressions that stem from a class of your model, like X.allinstances or X.Create. The EAL Editor is displayed by the button next to the expression: {{image:Can do action -1.png|frameless|330x330px}} If the Action is a ContextAction, you get access to the variables that the context defines. These are named: vCurrent_TheViewModelClassName. You can either act on classes as with GlobalActions or act on the current value of variables from the context. The variables follow the selections the users make in ViewModel grids, so you can act on things the user has selected. When you have an object context like this, you can call methods that the objects class defines: {{image:Can do action -5.png|frameless|398x398px}} You can use all the attributes and navigate all the associations in as many steps as you need to get the desired effect. If the action is a ClassAction, the action is rooted in an object of that class, and you use the OCL identifier "self" to operate on this. EnableExpression The EnableExpression is very similar to the ExecuteExpression, except it is not an EAL expression but rather, an OCL Expression. As such, it cannot have side effects (it cannot change anything in your domain of objects). The EnableExpression must also evaluate to a Boolean value of true or false. {{image:Can do action -3.png|frameless|360x360px}} Except for these differences, the expression operates in the same context and can reach the same variables, etc., as

described for the ExecuteExpression. {{image:Can do action -6.png|frameless|376x376px}} The EnableExpression is used to control whether an action should be executable or not. Consider that you have a state machine on class X and you only want the Delete action enabled when X is in the state Deletable: self.oclIsInState(#Deletable) BringUpViewModel & ViewModelRootObjectExpression An action can bring up a ViewModel-defined UI, and when it does, it will assign the Root object of the ViewModel, the result of the ViewModelRootObjectExpression. If the ViewModel RootObject Expression is empty, the Root object of the brought-up ViewModel-defined UI will be unassigned – and this might be perfectly fine for certain types of UIs, like UIs designed to seek persistent storage, for example. ViewModelIsModal & ExpressionAfterModalOk {{image:Can do action -7.png|frameless|368x368px}} If the action is set to bring up a ViewModel-powered UI, you may also choose to bring that UI up as a Modal window (a dialog you need to Ok or Cancel to get away from). The WECPOF runtime will add an OK and Cancel button to the bottom of your window if this is set. If the user clicks Cancel in such a window, nothing else happens besides the fact that the window is closed. However, if the user clicks Ok, the WECPOF runtime will execute the ExpressionAfterModalOk if you have defined it. The ExpressionAfterModalOk is an EAL expression that works on the context exactly as the ExecuteExpression does, but the ExpressionAfterModalOk also brings the variables from the ViewModel UI that were modal and are now closing. This fact enables you to read the values of the window about to close and to apply these values to your current context. The variables taken from the window that is closing are prefixed with "vModalResult_" to avoid name collisions with the calling context.

The MDriven Book

{{image:Can do action -8.png|frameless|388x388px}} This will enable you to create a Seek/Pick/Assign pattern in a WECPOF application. Consider that you have a Car-rental-model, you have a ViewModel that shows the rental contract and you now need to assign a Car to the Contract. The choice to use a ComboBox to scroll and find a car is no good because the number of possible Cars is 500+. Instead, add a ContextAction to the Rental contract ViewModel that brings up the Free-Car-Seeker-ViewModel, and you tick the ViewModelIsModal checkbox. You also define the ExpressionAfterModalOk to be vCurrent_RentalContract.AssignedCar:=vModalResult_Current_PickedCar. Framework Action If you pick a framework action, none of the other expressions will apply. {{image:Can do action -9.png|frameless|418x418px}} The Framework action is added to allow access to functions that operate on a different level than your model. * Save – saves the changed (created, deleted, updated) objects using the persistence mapper you have used in your EcoSpace (Or in Gaffr). * Refresh – calls refresh on your Persistence mapper if it supports Refresh (set up sync server to allow) * Undo/Redo – calls undo or redo on your ecospace. The current WECPOF runtime uses a timer to create new undo-blocks every other second if there are new changes. * Exit – quits the WECPOF application The MDriven Book - Next Chapter: Global actions {{Edited|July|12|2024}}

EnableExpression

Write the content here to display this box Actions can do a couple of things. ExecuteExpression {{image:Can do action -2.png|frameless|466x466px}} The ExecuteExpression is an Extended-Action-Language expression that is rooted in some context. For GlobalActions, this context is null. What you can do is limited to expressions that stem from a class of your model, like X.allinstances or X.Create. The EAL Editor is displayed by the button next to the expression: {{image:Can do action -1.png|frameless|330x330px}} If the Action is a ContextAction, you get access to the variables that the context defines. These are named: vCurrent_TheViewModelClassName. You can either act on classes as with GlobalActions or act on the current value of variables from the context. The variables follow the selections the users make in ViewModel grids, so you can act on things the user has selected. When you have an object context like this, you can call methods that the objects class defines: {{image:Can do action -5.png|frameless|398x398px}} You can use all the attributes and navigate all the associations in as many steps as you need to get the desired effect. If the action is a ClassAction, the action is rooted in an object of that class, and you use the OCL identifier "self" to operate on this. EnableExpression The EnableExpression is very similar to the ExecuteExpression, except it is not an EAL expression but rather, an OCL Expression. As such, it cannot have side effects (it cannot change anything in your domain of objects). The EnableExpression must also evaluate to a Boolean value of true or false. {{image:Can do action -3.png|frameless|360x360px}} Except for these differences, the expression operates in the same context and can reach the same variables, etc., as

described for the ExecuteExpression. {{image:Can do action -6.png|frameless|376x376px}} The EnableExpression is used to control whether an action should be executable or not. Consider that you have a state machine on class X and you only want the Delete action enabled when X is in the state Deletable: self.oclIsInState(#Deletable) BringUpViewModel & ViewModelRootObjectExpression An action can bring up a ViewModel-defined UI, and when it does, it will assign the Root object of the ViewModel, the result of the ViewModelRootObjectExpression. If the ViewModel RootObject Expression is empty, the Root object of the brought-up ViewModel-defined UI will be unassigned – and this might be perfectly fine for certain types of UIs, like UIs designed to seek persistent storage, for example. ViewModelIsModal & ExpressionAfterModalOk {{image:Can do action -7.png|frameless|368x368px}} If the action is set to bring up a ViewModel-powered UI, you may also choose to bring that UI up as a Modal window (a dialog you need to Ok or Cancel to get away from). The WECPOF runtime will add an OK and Cancel button to the bottom of your window if this is set. If the user clicks Cancel in such a window, nothing else happens besides the fact that the window is closed. However, if the user clicks Ok, the WECPOF runtime will execute the ExpressionAfterModalOk if you have defined it. The ExpressionAfterModalOk is an EAL expression that works on the context exactly as the ExecuteExpression does, but the ExpressionAfterModalOk also brings the variables from the ViewModel UI that were modal and are now closing. This fact enables you to read the values of the window about to close and to apply these values to your current context. The variables taken from the window that is closing are prefixed with "vModalResult_" to avoid name collisions with the calling context.

The MDriven Book

{{image:Can do action -8.png|frameless|388x388px}} This will enable you to create a Seek/Pick/Assign pattern in a WECPOF application. Consider that you have a Car-rental-model, you have a ViewModel that shows the rental contract and you now need to assign a Car to the Contract. The choice to use a ComboBox to scroll and find a car is no good because the number of possible Cars is 500+. Instead, add a ContextAction to the Rental contract ViewModel that brings up the Free-Car-Seeker-ViewModel, and you tick the ViewModelIsModal checkbox. You also define the ExpressionAfterModalOk to be vCurrent_RentalContract.AssignedCar:=vModalResult_Current_PickedCar. Framework Action If you pick a framework action, none of the other expressions will apply. {{image:Can do action -9.png|frameless|418x418px}} The Framework action is added to allow access to functions that operate on a different level than your model. * Save – saves the changed (created, deleted, updated) objects using the persistence mapper you have used in your EcoSpace (Or in Gaffr). * Refresh – calls refresh on your Persistence mapper if it supports Refresh (set up sync server to allow) * Undo/Redo – calls undo or redo on your ecospace. The current WECPOF runtime uses a timer to create new undo-blocks every other second if there are new changes. * Exit – quits the WECPOF application The MDriven Book - Next Chapter: Global actions {{Edited|July|12|2024}}

BringUpViewModel & ViewModelRootObjectExpression

Write the content here to display this box Actions can do a couple of things. ExecuteExpression {{image:Can do action -2.png|frameless|466x466px}} The ExecuteExpression is an Extended-Action-Language expression that is rooted in some context. For GlobalActions, this context is null. What you can do is limited to expressions that stem from a class of your model, like X.allinstances or X.Create. The EAL Editor is displayed by the button next to the expression: {{image:Can do action -1.png|frameless|330x330px}} If the Action is a ContextAction, you get access to the variables that the context defines. These are named: vCurrent_TheViewModelClassName. You can either act on classes as with GlobalActions or act on the current value of variables from the context. The variables follow the selections the users make in ViewModel grids, so you can act on things the user has selected. When you have an object context like this, you can call methods that the objects class defines: {{image:Can do action -5.png|frameless|398x398px}} You can use all the attributes and navigate all the associations in as many steps as you need to get the desired effect. If the action is a ClassAction, the action is rooted in an object of that class, and you use the OCL identifier "self" to operate on this. EnableExpression The EnableExpression is very similar to the ExecuteExpression, except it is not an EAL expression but rather, an OCL Expression. As such, it cannot have side effects (it cannot change anything in your domain of objects). The EnableExpression must also evaluate to a Boolean value of true or false. {{image:Can do action -3.png|frameless|360x360px}} Except for these differences, the expression operates in the same context and can reach the same variables, etc., as

described for the ExecuteExpression. {{image:Can do action -6.png|frameless|376x376px}} The EnableExpression is used to control whether an action should be executable or not. Consider that you have a state machine on class X and you only want the Delete action enabled when X is in the state Deletable: self.oclIsInState(#Deletable) BringUpViewModel & ViewModelRootObjectExpression An action can bring up a ViewModel-defined UI, and when it does, it will assign the Root object of the ViewModel, the result of the ViewModelRootObjectExpression. If the ViewModel RootObject Expression is empty, the Root object of the brought-up ViewModel-defined UI will be unassigned – and this might be perfectly fine for certain types of UIs, like UIs designed to seek persistent storage, for example. ViewModelIsModal & ExpressionAfterModalOk {{image:Can do action -7.png|frameless|368x368px}} If the action is set to bring up a ViewModel-powered UI, you may also choose to bring that UI up as a Modal window (a dialog you need to Ok or Cancel to get away from). The WECPOF runtime will add an OK and Cancel button to the bottom of your window if this is set. If the user clicks Cancel in such a window, nothing else happens besides the fact that the window is closed. However, if the user clicks Ok, the WECPOF runtime will execute the ExpressionAfterModalOk if you have defined it. The ExpressionAfterModalOk is an EAL expression that works on the context exactly as the ExecuteExpression does, but the ExpressionAfterModalOk also brings the variables from the ViewModel UI that were modal and are now closing. This fact enables you to read the values of the window about to close and to apply these values to your current context. The variables taken from the window that is closing are prefixed with "vModalResult_" to avoid name collisions with the calling context.

The MDriven Book

{{image:Can do action -8.png|frameless|388x388px}} This will enable you to create a Seek/Pick/Assign pattern in a WECPOF application. Consider that you have a Car-rental-model, you have a ViewModel that shows the rental contract and you now need to assign a Car to the Contract. The choice to use a ComboBox to scroll and find a car is no good because the number of possible Cars is 500+. Instead, add a ContextAction to the Rental contract ViewModel that brings up the Free-Car-Seeker-ViewModel, and you tick the ViewModelIsModal checkbox. You also define the ExpressionAfterModalOk to be vCurrent_RentalContract.AssignedCar:=vModalResult_Current_PickedCar. Framework Action If you pick a framework action, none of the other expressions will apply. {{image:Can do action -9.png|frameless|418x418px}} The Framework action is added to allow access to functions that operate on a different level than your model. * Save – saves the changed (created, deleted, updated) objects using the persistence mapper you have used in your EcoSpace (Or in Gaffr). * Refresh – calls refresh on your Persistence mapper if it supports Refresh (set up sync server to allow) * Undo/Redo – calls undo or redo on your ecospace. The current WECPOF runtime uses a timer to create new undo-blocks every other second if there are new changes. * Exit – quits the WECPOF application The MDriven Book - Next Chapter: Global actions {{Edited|July|12|2024}}

ViewModelIsModal & ExpressionAfterModalOk

Write the content here to display this box Actions can do a couple of things. ExecuteExpression {{image:Can do action -2.png|frameless|466x466px}} The ExecuteExpression is an Extended-Action-Language expression that is rooted in some context. For GlobalActions, this context is null. What you can do is limited to expressions that stem from a class of your model, like X.allinstances or X.Create. The EAL Editor is displayed by the button next to the expression: {{image:Can do action -1.png|frameless|330x330px}} If the Action is a ContextAction, you get access to the variables that the context defines. These are named: vCurrent_TheViewModelClassName. You can either act on classes as with GlobalActions or act on the current value of variables from the context. The variables follow the selections the users make in ViewModel grids, so you can act on things the user has selected. When you have an object context like this, you can call methods that the objects class defines: {{image:Can do action -5.png|frameless|398x398px}} You can use all the attributes and navigate all the associations in as many steps as you need to get the desired effect. If the action is a ClassAction, the action is rooted in an object of that class, and you use the OCL identifier "self" to operate on this. EnableExpression The EnableExpression is very similar to the ExecuteExpression, except it is not an EAL expression but rather, an OCL Expression. As such, it cannot have side effects (it cannot change anything in your domain of objects). The EnableExpression must also evaluate to a Boolean value of true or false. {{image:Can do action -3.png|frameless|360x360px}} Except for these differences, the expression operates in the same context and can reach the same variables, etc., as

described for the ExecuteExpression. {{image:Can do action -6.png|frameless|376x376px}} The EnableExpression is used to control whether an action should be executable or not. Consider that you have a state machine on class X and you only want the Delete action enabled when X is in the state Deletable: self.oclIsInState(#Deletable) BringUpViewModel & ViewModelRootObjectExpression An action can bring up a ViewModel-defined UI, and when it does, it will assign the Root object of the ViewModel, the result of the ViewModelRootObjectExpression. If the ViewModel RootObject Expression is empty, the Root object of the brought-up ViewModel-defined UI will be unassigned – and this might be perfectly fine for certain types of UIs, like UIs designed to seek persistent storage, for example. ViewModelIsModal & ExpressionAfterModalOk {{image:Can do action -7.png|frameless|368x368px}} If the action is set to bring up a ViewModel-powered UI, you may also choose to bring that UI up as a Modal window (a dialog you need to Ok or Cancel to get away from). The WECPOF runtime will add an OK and Cancel button to the bottom of your window if this is set. If the user clicks Cancel in such a window, nothing else happens besides the fact that the window is closed. However, if the user clicks Ok, the WECPOF runtime will execute the ExpressionAfterModalOk if you have defined it. The ExpressionAfterModalOk is an EAL expression that works on the context exactly as the ExecuteExpression does, but the ExpressionAfterModalOk also brings the variables from the ViewModel UI that were modal and are now closing. This fact enables you to read the values of the window about to close and to apply these values to your current context. The variables taken from the window that is closing are prefixed with "vModalResult_" to avoid name collisions with the calling context.

The MDriven Book

{{image:Can do action -8.png|frameless|388x388px}} This will enable you to create a Seek/Pick/Assign pattern in a WECPOF application. Consider that you have a Car-rental-model, you have a ViewModel that shows the rental contract and you now need to assign a Car to the Contract. The choice to use a ComboBox to scroll and find a car is no good because the number of possible Cars is 500+. Instead, add a ContextAction to the Rental contract ViewModel that brings up the Free-Car-Seeker-ViewModel, and you tick the ViewModelIsModal checkbox. You also define the ExpressionAfterModalOk to be vCurrent_RentalContract.AssignedCar:=vModalResult_Current_PickedCar. Framework Action If you pick a framework action, none of the other expressions will apply. {{image:Can do action -9.png|frameless|418x418px}} The Framework action is added to allow access to functions that operate on a different level than your model. * Save – saves the changed (created, deleted, updated) objects using the persistence mapper you have used in your EcoSpace (Or in Gaffr). * Refresh – calls refresh on your Persistence mapper if it supports Refresh (set up sync server to allow) * Undo/Redo – calls undo or redo on your ecospace. The current WECPOF runtime uses a timer to create new undo-blocks every other second if there are new changes. * Exit – quits the WECPOF application The MDriven Book - Next Chapter: Global actions {{Edited|July|12|2024}}

The MDriven Book

Framework Action

Write the content here to display this box Actions can do a couple of things. ExecuteExpression {{image:Can do action -2.png|frameless|466x466px}} The ExecuteExpression is an Extended-Action-Language expression that is rooted in some context. For GlobalActions, this context is null. What you can do is limited to expressions that stem from a class of your model, like X.allinstances or X.Create. The EAL Editor is displayed by the button next to the expression: {{image:Can do action -1.png|frameless|330x330px}} If the Action is a ContextAction, you get access to the variables that the context defines. These are named: vCurrent_TheViewModelClassName. You can either act on classes as with GlobalActions or act on the current value of variables from the context. The variables follow the selections the users make in ViewModel grids, so you can act on things the user has selected. When you have an object context like this, you can call methods that the objects class defines: {{image:Can do action -5.png|frameless|398x398px}} You can use all the attributes and navigate all the associations in as many steps as you need to get the desired effect. If the action is a ClassAction, the action is rooted in an object of that class, and you use the OCL identifier "self" to operate on this. EnableExpression The EnableExpression is very similar to the ExecuteExpression, except it is not an EAL expression but rather, an OCL Expression. As such, it cannot have side effects (it cannot change anything in your domain of objects). The EnableExpression must also evaluate to a Boolean value of true or false. {{image:Can do action -3.png|frameless|360x360px}} Except for these differences, the expression operates in the same context and can reach the same variables, etc., as

described for the ExecuteExpression. {{image:Can do action -6.png|frameless|376x376px}} The EnableExpression is used to control whether an action should be executable or not. Consider that you have a state machine on class X and you only want the Delete action enabled when X is in the state Deletable: self.oclIsInState(#Deletable) BringUpViewModel & ViewModelRootObjectExpression An action can bring up a ViewModel-defined UI, and when it does, it will assign the Root object of the ViewModel, the result of the ViewModelRootObjectExpression. If the ViewModel RootObject Expression is empty, the Root object of the brought-up ViewModel-defined UI will be unassigned – and this might be perfectly fine for certain types of UIs, like UIs designed to seek persistent storage, for example. ViewModelIsModal & ExpressionAfterModalOk {{image:Can do action -7.png|frameless|368x368px}} If the action is set to bring up a ViewModel-powered UI, you may also choose to bring that UI up as a Modal window (a dialog you need to Ok or Cancel to get away from). The WECPOF runtime will add an OK and Cancel button to the bottom of your window if this is set. If the user clicks Cancel in such a window, nothing else happens besides the fact that the window is closed. However, if the user clicks Ok, the WECPOF runtime will execute the ExpressionAfterModalOk if you have defined it. The ExpressionAfterModalOk is an EAL expression that works on the context exactly as the ExecuteExpression does, but the ExpressionAfterModalOk also brings the variables from the ViewModel UI that were modal and are now closing. This fact enables you to read the values of the window about to close and to apply these values to your current context. The variables taken from the window that is closing are prefixed with "vModalResult_" to avoid name collisions with the calling context.

The MDriven Book

{{image:Can do action -8.png|frameless|388x388px}} This will enable you to create a Seek/Pick/Assign pattern in a WECPOF application. Consider that you have a Car-rental-model, you have a ViewModel that shows the rental contract and you now need to assign a Car to the Contract. The choice to use a ComboBox to scroll and find a car is no good because the number of possible Cars is 500+. Instead, add a ContextAction to the Rental contract ViewModel that brings up the Free-Car-Seeker-ViewModel, and you tick the ViewModelIsModal checkbox. You also define the ExpressionAfterModalOk to be vCurrent_RentalContract.AssignedCar:=vModalResult_Current_PickedCar. Framework Action If you pick a framework action, none of the other expressions will apply. {{image:Can do action -9.png|frameless|418x418px}} The Framework action is added to allow access to functions that operate on a different level than your model. * Save – saves the changed (created, deleted, updated) objects using the persistence mapper you have used in your EcoSpace (Or in Gaffr). * Refresh – calls refresh on your Persistence mapper if it supports Refresh (set up sync server to allow) * Undo/Redo – calls undo or redo on your ecospace. The current WECPOF runtime uses a timer to create new undo-blocks every other second if there are new changes. * Exit – quits the WECPOF application The MDriven Book - Next Chapter: Global actions {{Edited|July|12|2024}}

Defining Main Menu Actions

The MDriven Book

Action names

Write the content here to display this box When declaring Actions in MDriven, you have the option of giving a name (required) and a Presentation (defaults to). The reason for the two different properties is that the ActionName must be globally unique for all actions. It is used for the presentation of the action and as a reference name of the action. The presentation is, however, the text we use when presenting the action in a contextmenu or in the MDriven Prototype left-side action column. Since the Presentation often resembles the Name, we have introduced the following shorthand/activecontent: {| class="wikitable" |ActionName |Presentation value |Presentation result |- |Somename | |Somename |- |SomeName | |Some Name |- |SomeName |Extra1 Extra2 |Extra1 Some Name Extra 2 |- |SomeName | |"DefaultStringRep of context object" |- |SomeName | |Some Name "DefaultStringRep of context object" |- |SomeName | |"Value of attribute1 of context object" |- |SomeName |Extra1 Extra2 Extra3 |Extra1 Some Name Extra 2 "Value of attribute1 of context object" Extra 3 |} The context object is of course the object that the actions are acting on. So it is intended to be used with "Class actions". Constraints descriptions As we introduced the above-mentioned shorthand, we also made the part available in the constraints Description texts. This way your constraints descriptions can get context info. If you have the need for a complex expression, like navigation for pulling attributes from neighbor classes, you are advised to create the expression as a derived attribute that you then can reference from your constraint description text. The MDriven Book - Next Chapter: Microsoft office and OpenDocument as a Report generator

The MDriven Book

Constraints descriptions

Write the content here to display this box When declaring Actions in MDriven, you have the option of giving a name (required) and a Presentation (defaults to). The reason for the two different properties is that the ActionName must be globally unique for all actions. It is used for the presentation of the action and as a reference name of the action. The presentation is, however, the text we use when presenting the action in a contextmenu or in the MDriven Prototype left-side action column. Since the Presentation often resembles the Name, we have introduced the following shorthand/activecontent: {| class="wikitable" |ActionName |Presentation value |Presentation result |- |Somename | |Somename |- |SomeName | |Some Name |- |SomeName |Extra1 Extra2 |Extra1 Some Name Extra 2 |- |SomeName | |"DefaultStringRep of context object" |- |SomeName | |Some Name "DefaultStringRep of context object" |- |SomeName | |"Value of attribute1 of context object" |- |SomeName |Extra1 Extra2 Extra3 |Extra1 Some Name Extra 2 "Value of attribute1 of context object" Extra 3 |} The context object is of course the object that the actions are acting on. So it is intended to be used with "Class actions". Constraints descriptions As we introduced the above-mentioned shorthand, we also made the part available in the constraints Description texts. This way your constraints descriptions can get context info. If you have the need for a complex expression, like navigation for pulling attributes from neighbor classes, you are advised to create the expression as a derived attribute that you then can reference from your constraint description text. The MDriven Book - Next Chapter: Microsoft office and OpenDocument as a Report generator

The MDriven Book

Microsoft Office and OpenDocument as a Report generator

The MDriven Book

A bit hasty and vague

Write the content here to display this box Update: 2024.07.01 Generate Reports Using OpenDocument and Microsoft Office OpenDocument format is an open file format standard for office applications compatible with Microsoft Office and open source applications like LibreOffice and OpenOffice. Common filename extensions used for OpenDocument documents are: * .odt for text documents * .ods for spreadsheet documents MDriven applications allow generating reports from model-driven data using OpenDocument format. Text Document 1. Start by creating an OpenDocument text document using any Office application that supports OpenDocument format. 2. Add %meta% tag within the document as this will be used to print out all the available tags within your ViewModel for printing out model data. NOTE: Make sure to write tags without spaces between the word (meta) and the percentage (%) signs. Saving Strategies a) Temporary Location : (i) Create a folder named temp in your C:/ directory and save your file in the directory as mytemplate.odt. : (ii) Your url path will now be 'c:\\temp\\mytemplate.odt'. b) Permanent Location (AssetsTK Strategy) : (i) This strategy allows your template document to be uploaded with your model onto the server running your Turnkey application during deployment. : (ii) Go to the location where your .modlr file is saved. : (iii) Create a folder with the name in the format of _AssetsTK where is the name of your .modlr file name. : (iv) Within the folder create another folder named content where your save your template document. : (v) Your url path will now be 'http://localhost:8182/content/mytemplate.odt' : (vi) Check here on how to have a dynamic url for when your deploy your application onto the Mdriven Server. 3. Create a ViewModel with the Name

OpenDocumentReportTemplate. This ViewModel will be used as a template for the Model-Driven data to generate reports. 4. Within the ViewModel context menu, click Add column > Add columns needed for your report or create columns TemplateUrl and ReportFileName within the ViewModel. 5. Select the ViewModel to view the settings on the right. Uncheck the Use placing hints section at the top. 6. In the TemplateUrl column expression, enter the url path created earlier using any of the saving strategies above. In the ReportFileName column expression, enter a file name for the new OpenDocument that will be generated in the format '.odt' where is the name of your file. Create a class action within the ViewModel whose data you want to print out using the expression below: self.opendocumentreportshow(.ViewModels.OpenDocumentReportTemplate) where is the root class of the ViewModel. Within OpenDocumentReportTemplate, add columns whose data you want to print out. Trigger print out using the created class action. The first print out will return all tags available for printing out data. Use these tags to format the structure of your report document. If you don't want to use the %meta% tag to print out the available tags first, you may use the column names directly with the same expression %%. To print out nested ViewModel data in a table format, create a table and add the expression in the following format: {| class="wikitable" |%%+%%% |%% |%% |} ViewModel Template for Printing Out Text Open Document {{image:viewmodel-template-for-text-document.png|alt=ViewModel Template for Printing out Model-Driven data to OpenDocument reports|none|thumb|539x539px}} Text Document Template Located at _AssetsTK/Content/mytemplate.odt and accessible at

http://localhost:8182/content/mytemplate.odt

{{image:open-document-text-template.png|alt=Office Text Document for Printing out model-driven data using OpenDocument format|none|thumb|538x538px}} The table format is for both text documents and spreadsheet documents. When using Microsoft Office Word, [https://support.microsoft.com/en-us/office/differences-between-the-opendoc ument-text-odt-format-and-the-word-docx-format-d9d51a92-56d1-4794-8b68-5efb57aebfdc check here] for format styling that is supported in OpenDocument format. Spread Sheet Document 1. Start by creating an OpenDocument spread sheet document using any Office application that supports OpenDocument format. 2. Add %meta% tag within the document as this will be used to print out all the available tags within our ViewModel for printing out model data or use the column names directly with the same expression %% where represents the column name in the ViewModel. 3. Use .ods extension for spread sheet documents and your url path will now be 'http://localhost:8182/content/mytemplate.ods'. ViewModel Template for Printing Out Spread Sheet Open Document {{image:viewmodel-template-for-spreadsheet-open-document.png|alt=Sprea d Sheet Document ViewModel Template|none|thumb|596x596px}}Adding _float at the end of a column name ensures that the output in Excel is processed as a number. Spread Sheet Template Located at _AssetsTK/Content/mytemplate.ods and accessible at http://localhost:8182/content/mytemplate.ods

{{image:spreadsheet-document-template.png|alt=Spread Sheet Document Template|none|thumb|594x594px}} Sample Spread Sheet Output {{image:spreadsheet-sample-output.png|alt=SpreadSheet Sample

Output|none|thumb|593x593px}} Other Pages On Printing Out Documents: * OpenDocument * HtmlReport * Use LibreOffice for PDF conversion Older Information - 2018.04.01 Because we thought it would be great if we could generate Word and Excel documents straight from model-driven data, we made it happen. This article explains how to do it. Create a Word template: {{image:Reporting.png|frameless|408x408px}} You can have grids in grids to create structure: {{image:Reporting 1.png|frameless|424x424px}} Save this as an open document file (odt). Save it to a place where you can access it from a URL (maybe you use SharePoint, or just stick it on some website). Now you need data. Declare a ViewModel in Modlr: {{image:Reporting 2.png|frameless|411x411px}} (By now, you know that a ViewModel transforms your model for a specific need – in this case, the OpenDocument Report.) {{image:Reporting 3.png|frameless|384x384px}} Create two extra ViewModelColumns in your ViewModel - TemplateUrl and ReportFileName: {{image:Reporting 4.png|frameless|444x444px}} Make the TemplateUrl column return the URL to where your template from above can be found – maybe in some SharePoint instances, as in this example. Make the ReportFileName column return what the file should be called when produced. And you are done. Execute the report by using the new EAL operator: vSomePumpRev.opendocumentreportshow('PumpRevDeepReport') – this will call the OnOpenDocument event on the new IOpenDocumentService. This happens when each UI platform has its own way of showing things to a user. namespace Eco.Services { // Summary: // OpenDocumentService , ViewModel needs root level column describing url to // template. Column must be namned "TemplateUrl" and be a valid url to a open // document template public interface IOpenDocumentService { event EventHandler

OnOpenDocument; byte[] AsByteArray(IObject vmroot, string viewModelName, out string reportname); void ExecuteOnOpenDocument(IObject vmroot, string viewModelName, byte[] openDocumentData, string reportname); } } In WECPOF for WPF, we do this: File.WriteAllBytes(suggestedfilename, openDocumentData); System.Diagnostics.ProcessStartInfo sInfo = new System.Diagnostics.ProcessStartInfo(suggestedfilename); System.Diagnostics.Process.Start(sInfo); In ASP.NET, you would do something else. If you do not want to open the file – generate the data within – then use the new EAL operator vSomePumpRev.opendocumentreportasblob('PumpRevDeepReport') A Brief Recap (A Bit Hasty and Vague) So you might think I just skimmed over stuff – like: how do you get a hold of the placeholder tags that are replaced with data? Do this by entering the tag %meta% in your template. We will always look for this tag – and when found, we will add all valid tags in their places. How do you create a hierarchical structure in the report? Find it like this in the data: {{image:Reporting 5.png|frameless|387x387px}} And like this in the template: {{image:Reporting 6.png|frameless|378x378px}} The tag %%+Name% acts as a row builder. The following tag %OtherName% is the data in the child. In the Example: %%+ComponentSpecificationRevs% – I stick this as the first thing in a table row and the row will be duplicated for each child. Then the %RevisionNumber% is filled in in the cell. The reporting mechanism also works for Excel. An example of an Excel result report: {{image:Reporting 7.png|frameless|339x339px}} Update 2014-03-06. Qualifications When working with reports, we sometimes do not know in design time what needs the report will have at runtime. To handle this

situation, the template tagging has been extended to allow for qualifications. Let me explain. The smallest possible report sample: {{image:Reporting 8.png|frameless|254x254px}} {{image:Reporting 9.png|frameless|264x264px}} {{image:Reporting 10.png|frameless|376x376px}} We want to allow for picking the correct Class2 in runtime time while working on the template. If we have this Excel template: {{image:Reporting 11.png|frameless|279x279px}} We get this data out: {{image:Reporting 12.png|frameless|268x268px}} The qualification extension is that we can now have Template tags like this: %Class2[Name=Hello1]Name% What this means is that we are navigating to ViewModel column Class2 – but this is a list – and filter the result on the ViewModel column Name of ViewModelClass Class2. Taking the one with value "Hello1" – for that Class2 we use the Name column… Example: {{image:Reporting 13.png|frameless|410x410px}} will give you: {{image:Reporting 14.png|frameless|410x410px}} (Notice that we have different external ids in the two last columns – the first from a Class2 with NameHello1, the other from one with NameHello2) This is useful when you have data in name-value pair patterns. Update 2014-04-04 -- Images in Word reports This is how you can do it. Add some placeholder images in the template: {{image:Reporting 15.png|frameless|375x375px}} On the Image "Format Picture" – Alt Text property – enter the Tag that holds the image blob in your ViewModel: {{image:Reporting 16.png|frameless|404x404px}} Later, in Office(Word), it will look like this: {{image:2022-05-27 17h19 23.png|none|thumb|424x424px}} Now, the image will be replaced with your data. Also, new today is the fact that the Aspect ratio of your data is kept in the final Word (odt) report: {{image:Reporting

The MDriven Book

17.png|frameless|420x420px}} The MDriven Book - Next Chapter: Prototyping

The MDriven Book

Qualifications

Write the content here to display this box Update: 2024.07.01 Generate Reports Using OpenDocument and Microsoft Office OpenDocument format is an open file format standard for office applications compatible with Microsoft Office and open source applications like LibreOffice and OpenOffice. Common filename extensions used for OpenDocument documents are: * .odt for text documents * .ods for spreadsheet documents MDriven applications allow generating reports from model-driven data using OpenDocument format. Text Document 1. Start by creating an OpenDocument text document using any Office application that supports OpenDocument format. 2. Add %meta% tag within the document as this will be used to print out all the available tags within your ViewModel for printing out model data. NOTE: Make sure to write tags without spaces between the word (meta) and the percentage (%) signs. Saving Strategies a) Temporary Location : (i) Create a folder named temp in your C:/ directory and save your file in the directory as mytemplate.odt. : (ii) Your url path will now be 'c:\\temp\\mytemplate.odt'. b) Permanent Location (AssetsTK Strategy) : (i) This strategy allows your template document to be uploaded with your model onto the server running your Turnkey application during deployment. : (ii) Go to the location where your .modlr file is saved. : (iii) Create a folder with the name in the format of _AssetsTK where is the name of your .modlr file name. : (iv) Within the folder create another folder named content where your save your template document. : (v) Your url path will now be 'http://localhost:8182/content/mytemplate.odt' : (vi) Check here on how to have a dynamic url for when your deploy your application onto the Mdriven Server. 3. Create a ViewModel with the Name

OpenDocumentReportTemplate. This ViewModel will be used as a template for the Model-Driven data to generate reports. 4. Within the ViewModel context menu, click Add column > Add columns needed for your report or create columns TemplateUrl and ReportFileName within the ViewModel. 5. Select the ViewModel to view the settings on the right. Uncheck the Use placing hints section at the top. 6. In the TemplateUrl column expression, enter the url path created earlier using any of the saving strategies above. In the ReportFileName column expression, enter a file name for the new OpenDocument that will be generated in the format '.odt' where is the name of your file. Create a class action within the ViewModel whose data you want to print out using the expression below: self.opendocumentreportshow(.ViewModels.OpenDocumentReportTemplate) where is the root class of the ViewModel. Within OpenDocumentReportTemplate, add columns whose data you want to print out. Trigger print out using the created class action. The first print out will return all tags available for printing out data. Use these tags to format the structure of your report document. If you don't want to use the %meta% tag to print out the available tags first, you may use the column names directly with the same expression %%. To print out nested ViewModel data in a table format, create a table and add the expression in the following format: {| class="wikitable" |%%+%%% |%% |%% |} ViewModel Template for Printing Out Text Open Document {{image:viewmodel-template-for-text-document.png|alt=ViewModel Template for Printing out Model-Driven data to OpenDocument reports|none|thumb|539x539px}} Text Document Template Located at _AssetsTK/Content/mytemplate.odt and accessible at

http://localhost:8182/content/mytemplate.odt

{{image:open-document-text-template.png|alt=Office Text Document for Printing out model-driven data using OpenDocument format|none|thumb|538x538px}} The table format is for both text documents and spreadsheet documents. When using Microsoft Office Word, [https://support.microsoft.com/en-us/office/differences-between-the-opendoc ument-text-odt-format-and-the-word-docx-format-d9d51a92-56d1-4794-8b68-5efb57aebfdc check here] for format styling that is supported in OpenDocument format. Spread Sheet Document 1. Start by creating an OpenDocument spread sheet document using any Office application that supports OpenDocument format. 2. Add %meta% tag within the document as this will be used to print out all the available tags within our ViewModel for printing out model data or use the column names directly with the same expression %% where represents the column name in the ViewModel. 3. Use .ods extension for spread sheet documents and your url path will now be 'http://localhost:8182/content/mytemplate.ods'. ViewModel Template for Printing Out Spread Sheet Open Document {{image:viewmodel-template-for-spreadsheet-open-document.png|alt=Sprea d Sheet Document ViewModel Template|none|thumb|596x596px}}Adding _float at the end of a column name ensures that the output in Excel is processed as a number. Spread Sheet Template Located at _AssetsTK/Content/mytemplate.ods and accessible at http://localhost:8182/content/mytemplate.ods

{{image:spreadsheet-document-template.png|alt=Spread Sheet Document Template|none|thumb|594x594px}} Sample Spread Sheet Output {{image:spreadsheet-sample-output.png|alt=SpreadSheet Sample

The MDriven Book

Output|none|thumb|593x593px}} Other Pages On Printing Out Documents: * OpenDocument * HtmlReport * Use LibreOffice for PDF conversion Older Information - 2018.04.01 Because we thought it would be great if we could generate Word and Excel documents straight from model-driven data, we made it happen. This article explains how to do it. Create a Word template: {{image:Reporting.png|frameless|408x408px}} You can have grids in grids to create structure: {{image:Reporting 1.png|frameless|424x424px}} Save this as an open document file (odt). Save it to a place where you can access it from a URL (maybe you use SharePoint, or just stick it on some website). Now you need data. Declare a ViewModel in Modlr: {{image:Reporting 2.png|frameless|411x411px}} (By now, you know that a ViewModel transforms your model for a specific need – in this case, the OpenDocument Report.) {{image:Reporting 3.png|frameless|384x384px}} Create two extra ViewModelColumns in your ViewModel - TemplateUrl and ReportFileName: {{image:Reporting 4.png|frameless|444x444px}} Make the TemplateUrl column return the URL to where your template from above can be found – maybe in some SharePoint instances, as in this example. Make the ReportFileName column return what the file should be called when produced. And you are done. Execute the report by using the new EAL operator: vSomePumpRev.opendocumentreportshow('PumpRevDeepReport') – this will call the OnOpenDocument event on the new IOpenDocumentService. This happens when each UI platform has its own way of showing things to a user. namespace Eco.Services { // Summary: // OpenDocumentService , ViewModel needs root level column describing url to // template. Column must be namned "TemplateUrl" and be a valid url to a open // document template public interface IOpenDocumentService { event EventHandler

The MDriven Book

OnOpenDocument; byte[] AsByteArray(IObject vmroot, string viewModelName, out string reportname); void ExecuteOnOpenDocument(IObject vmroot, string viewModelName, byte[] openDocumentData, string reportname); } } In WECPOF for WPF, we do this: File.WriteAllBytes(suggestedfilename, openDocumentData); System.Diagnostics.ProcessStartInfo sInfo = new System.Diagnostics.ProcessStartInfo(suggestedfilename); System.Diagnostics.Process.Start(sInfo); In ASP.NET, you would do something else. If you do not want to open the file – generate the data within – then use the new EAL operator vSomePumpRev.opendocumentreportasblob('PumpRevDeepReport') A Brief Recap (A Bit Hasty and Vague) So you might think I just skimmed over stuff – like: how do you get a hold of the placeholder tags that are replaced with data? Do this by entering the tag %meta% in your template. We will always look for this tag – and when found, we will add all valid tags in their places. How do you create a hierarchical structure in the report? Find it like this in the data: {{image:Reporting 5.png|frameless|387x387px}} And like this in the template: {{image:Reporting 6.png|frameless|378x378px}} The tag %%+Name% acts as a row builder. The following tag %OtherName% is the data in the child. In the Example: %%+ComponentSpecificationRevs% – I stick this as the first thing in a table row and the row will be duplicated for each child. Then the %RevisionNumber% is filled in in the cell. The reporting mechanism also works for Excel. An example of an Excel result report: {{image:Reporting 7.png|frameless|339x339px}} Update 2014-03-06. Qualifications When working with reports, we sometimes do not know in design time what needs the report will have at runtime. To handle this

situation, the template tagging has been extended to allow for qualifications. Let me explain. The smallest possible report sample: {{image:Reporting 8.png|frameless|254x254px}} {{image:Reporting 9.png|frameless|264x264px}} {{image:Reporting 10.png|frameless|376x376px}} We want to allow for picking the correct Class2 in runtime time while working on the template. If we have this Excel template: {{image:Reporting 11.png|frameless|279x279px}} We get this data out: {{image:Reporting 12.png|frameless|268x268px}} The qualification extension is that we can now have Template tags like this: %Class2[Name=Hello1]Name% What this means is that we are navigating to ViewModel column Class2 – but this is a list – and filter the result on the ViewModel column Name of ViewModelClass Class2. Taking the one with value "Hello1" – for that Class2 we use the Name column… Example: {{image:Reporting 13.png|frameless|410x410px}} will give you: {{image:Reporting 14.png|frameless|410x410px}} (Notice that we have different external ids in the two last columns – the first from a Class2 with NameHello1, the other from one with NameHello2) This is useful when you have data in name-value pair patterns. Update 2014-04-04 -- Images in Word reports This is how you can do it. Add some placeholder images in the template: {{image:Reporting 15.png|frameless|375x375px}} On the Image "Format Picture" – Alt Text property – enter the Tag that holds the image blob in your ViewModel: {{image:Reporting 16.png|frameless|404x404px}} Later, in Office(Word), it will look like this: {{image:2022-05-27 17h19 23.png|none|thumb|424x424px}} Now, the image will be replaced with your data. Also, new today is the fact that the Aspect ratio of your data is kept in the final Word (odt) report: {{image:Reporting

17.png|frameless|420x420px}} **The MDriven Book - Next Chapter: Prototyping**

Images in Word reports

Write the content here to display this box Update: 2024.07.01 Generate Reports Using OpenDocument and Microsoft Office OpenDocument format is an open file format standard for office applications compatible with Microsoft Office and open source applications like LibreOffice and OpenOffice. Common filename extensions used for OpenDocument documents are: * .odt for text documents * .ods for spreadsheet documents MDriven applications allow generating reports from model-driven data using OpenDocument format. Text Document 1. Start by creating an OpenDocument text document using any Office application that supports OpenDocument format. 2. Add %meta% tag within the document as this will be used to print out all the available tags within your ViewModel for printing out model data. NOTE: Make sure to write tags without spaces between the word (meta) and the percentage (%) signs. Saving Strategies a) Temporary Location : (i) Create a folder named temp in your C:/ directory and save your file in the directory as mytemplate.odt. : (ii) Your url path will now be 'c:\\temp\\mytemplate.odt'. b) Permanent Location (AssetsTK Strategy) : (i) This strategy allows your template document to be uploaded with your model onto the server running your Turnkey application during deployment. : (ii) Go to the location where your .modlr file is saved. : (iii) Create a folder with the name in the format of _AssetsTK where is the name of your .modlr file name. : (iv) Within the folder create another folder named content where your save your template document. : (v) Your url path will now be 'http://localhost:8182/content/mytemplate.odt' : (vi) Check here on how to have a dynamic url for when your deploy your application onto the Mdriven Server. 3. Create a ViewModel with the Name

OpenDocumentReportTemplate. This ViewModel will be used as a template for the Model-Driven data to generate reports. 4. Within the ViewModel context menu, click Add column > Add columns needed for your report or create columns TemplateUrl and ReportFileName within the ViewModel. 5. Select the ViewModel to view the settings on the right. Uncheck the Use placing hints section at the top. 6. In the TemplateUrl column expression, enter the url path created earlier using any of the saving strategies above. In the ReportFileName column expression, enter a file name for the new OpenDocument that will be generated in the format '.odt' where is the name of your file. Create a class action within the ViewModel whose data you want to print out using the expression below: self.opendocumentreportshow(.ViewModels.OpenDocumentReportTemplate) where is the root class of the ViewModel. Within OpenDocumentReportTemplate, add columns whose data you want to print out. Trigger print out using the created class action. The first print out will return all tags available for printing out data. Use these tags to format the structure of your report document. If you don't want to use the %meta% tag to print out the available tags first, you may use the column names directly with the same expression %%. To print out nested ViewModel data in a table format, create a table and add the expression in the following format: {| class="wikitable" |%%+%%% |%% |%% |} ViewModel Template for Printing Out Text Open Document {{image:viewmodel-template-for-text-document.png|alt=ViewModel Template for Printing out Model-Driven data to OpenDocument reports|none|thumb|539x539px}} Text Document Template Located at _AssetsTK/Content/mytemplate.odt and accessible at

http://localhost:8182/content/mytemplate.odt

{{image:open-document-text-template.png|alt=Office Text Document for Printing out model-driven data using OpenDocument format|none|thumb|538x538px}} The table format is for both text documents and spreadsheet documents. When using Microsoft Office Word, [https://support.microsoft.com/en-us/office/differences-between-the-opendocument-text-odt-format-and-the-word-docx-format-d9d51a92-56d1-4794-8b68-5efb57aebfdc check here] for format styling that is supported in OpenDocument format. Spread Sheet Document 1. Start by creating an OpenDocument spread sheet document using any Office application that supports OpenDocument format. 2. Add %meta% tag within the document as this will be used to print out all the available tags within our ViewModel for printing out model data or use the column names directly with the same expression %% where represents the column name in the ViewModel. 3. Use .ods extension for spread sheet documents and your url path will now be 'http://localhost:8182/content/mytemplate.ods'. ViewModel Template for Printing Out Spread Sheet Open Document {{image:viewmodel-template-for-spreadsheet-open-document.png|alt=Spread Sheet Document ViewModel Template|none|thumb|596x596px}}Adding _float at the end of a column name ensures that the output in Excel is processed as a number. Spread Sheet Template Located at _AssetsTK/Content/mytemplate.ods and accessible at http://localhost:8182/content/mytemplate.ods

{{image:spreadsheet-document-template.png|alt=Spread Sheet Document Template|none|thumb|594x594px}} Sample Spread Sheet Output {{image:spreadsheet-sample-output.png|alt=SpreadSheet Sample

Output|none|thumb|593x593px}} Other Pages On Printing Out Documents: * OpenDocument * HtmlReport * Use LibreOffice for PDF conversion Older Information - 2018.04.01 Because we thought it would be great if we could generate Word and Excel documents straight from model-driven data, we made it happen. This article explains how to do it. Create a Word template: {{image:Reporting.png|frameless|408x408px}} You can have grids in grids to create structure: {{image:Reporting 1.png|frameless|424x424px}} Save this as an open document file (odt). Save it to a place where you can access it from a URL (maybe you use SharePoint, or just stick it on some website). Now you need data. Declare a ViewModel in Modlr: {{image:Reporting 2.png|frameless|411x411px}} (By now, you know that a ViewModel transforms your model for a specific need – in this case, the OpenDocument Report.) {{image:Reporting 3.png|frameless|384x384px}} Create two extra ViewModelColumns in your ViewModel - TemplateUrl and ReportFileName: {{image:Reporting 4.png|frameless|444x444px}} Make the TemplateUrl column return the URL to where your template from above can be found – maybe in some SharePoint instances, as in this example. Make the ReportFileName column return what the file should be called when produced. And you are done. Execute the report by using the new EAL operator: vSomePumpRev.opendocumentreportshow('PumpRevDeepReport') – this will call the OnOpenDocument event on the new IOpenDocumentService. This happens when each UI platform has its own way of showing things to a user. namespace Eco.Services { // Summary: // OpenDocumentService , ViewModel needs root level column describing url to // template. Column must be namned "TemplateUrl" and be a valid url to a open // document template public interface IOpenDocumentService { event EventHandler

OnOpenDocument; byte[] AsByteArray(IObject vmroot, string viewModelName, out string reportname); void ExecuteOnOpenDocument(IObject vmroot, string viewModelName, byte[] openDocumentData, string reportname); } } In WECPOF for WPF, we do this: File.WriteAllBytes(suggestedfilename, openDocumentData); System.Diagnostics.ProcessStartInfo sInfo = new System.Diagnostics.ProcessStartInfo(suggestedfilename); System.Diagnostics.Process.Start(sInfo); In ASP.NET, you would do something else. If you do not want to open the file – generate the data within – then use the new EAL operator vSomePumpRev.opendocumentreportasblob('PumpRevDeepReport') A Brief Recap (A Bit Hasty and Vague) So you might think I just skimmed over stuff – like: how do you get a hold of the placeholder tags that are replaced with data? Do this by entering the tag %meta% in your template. We will always look for this tag – and when found, we will add all valid tags in their places. How do you create a hierarchical structure in the report? Find it like this in the data: {{image:Reporting 5.png|frameless|387x387px}} And like this in the template: {{image:Reporting 6.png|frameless|378x378px}} The tag %%+Name% acts as a row builder. The following tag %OtherName% is the data in the child. In the Example: %%+ComponentSpecificationRevs% – I stick this as the first thing in a table row and the row will be duplicated for each child. Then the %RevisionNumber% is filled in in the cell. The reporting mechanism also works for Excel. An example of an Excel result report: {{image:Reporting 7.png|frameless|339x339px}} Update 2014-03-06. Qualifications When working with reports, we sometimes do not know in design time what needs the report will have at runtime. To handle this

situation, the template tagging has been extended to allow for qualifications. Let me explain. The smallest possible report sample: {{image:Reporting 8.png|frameless|254x254px}} {{image:Reporting 9.png|frameless|264x264px}} {{image:Reporting 10.png|frameless|376x376px}} We want to allow for picking the correct Class2 in runtime time while working on the template. If we have this Excel template: {{image:Reporting 11.png|frameless|279x279px}} We get this data out: {{image:Reporting 12.png|frameless|268x268px}} The qualification extension is that we can now have Template tags like this: %Class2[Name=Hello1]Name% What this means is that we are navigating to ViewModel column Class2 – but this is a list – and filter the result on the ViewModel column Name of ViewModelClass Class2. Taking the one with value "Hello1" – for that Class2 we use the Name column… Example: {{image:Reporting 13.png|frameless|410x410px}} will give you: {{image:Reporting 14.png|frameless|410x410px}} (Notice that we have different external ids in the two last columns – the first from a Class2 with NameHello1, the other from one with NameHello2) This is useful when you have data in name-value pair patterns. Update 2014-04-04 -- Images in Word reports This is how you can do it. Add some placeholder images in the template: {{image:Reporting 15.png|frameless|375x375px}} On the Image "Format Picture" – Alt Text property – enter the Tag that holds the image blob in your ViewModel: {{image:Reporting 16.png|frameless|404x404px}} Later, in Office(Word), it will look like this: {{image:2022-05-27 17h19 23.png|none|thumb|424x424px}} Now, the image will be replaced with your data. Also, new today is the fact that the Aspect ratio of your data is kept in the final Word (odt) report: {{image:Reporting

17.png|frameless|420x420px}} The MDriven Book - Next Chapter: Prototyping

Prototyping

The MDriven Book

This is how you do Prototyping with MDriven

Write the content here to display this box Used in all lines of engineering, prototyping is the process of whipping something together quickly as a mock-up to show or test to learn something that otherwise would be hard to know. Prototyping for software has a unique position. The prototype is composed of the same elements as the finished product: logical rules. This is untrue for any other practice of engineering. We have a unique position where we can harvest to make the real deal – the finished product – just as easily as if we were building a prototype. Many experienced developers frown at this. They know how prudent you must be to build something robust and solid. They also know how quick and dirty the prototype is when you're putting it together. Why does this huge gap exist? My answer: normal coding leaves too many degrees of freedom for the task at hand. Normal coding uses only one tool – code – for handling, presenting, and navigating data. In addition, you will mix things up when you are in a hurry. If we can separate things like the information we handle, from the transformation of that information into views and from the navigation between these views – then we have something that will almost fly on its own – and building something will be like prototyping it. Think of it as an autopilot; it will protect you against making stupid mistakes, help you fly straight, and give you time to talk on the radio or more – you only need to tell it where you want to fly. With MDriven, we use the model to tell the autopilot what information to handle, what the perspectives of the information should be, and how to navigate between the views of information. Having instructed the autopilot, we can take off and see if we have the right model or not. If not, we churn away and

fix it. **Many developers also frown at the idea of autopilot since it will emasculate them and stop them from flexing their muscles. I need to point out that what makes MDrivenFramework great is that you can turn off the autopilot for any portion of the flight so to speak. You can still do some cool looping in front of a gaping audience whenever you feel like it, before turning on autopilot to do the mundane flight back home. This will give you freedom, speed, and fewer accidents. This is How You Do It With MDriven 1. Model what you know so far. In the example below, I am prototyping for a Car Rental Service. {{image:Prototyping - 1.png|none|frame|527x527px}} 2. Think about what user stories or requirements you have:As a Customer, I want to see what cars you can rent.'As a Customer, I want to know what they cost per day and what my total cost will be.'As a Rental worker, I want to be able to hand customers rental contracts to sign.As a Rental worker, I want to be able to find free cars.3. Then, create some ViewModels that can cover these user stories. When prototyping, make use of the scaffolding user interface hints that place out UI controls on a screen surface that matches the types of the Viewmodels properties. I end up with 4 ViewModels:Search for a Car: {{image:Prototyping - 2.png|none|frame|478x478px}} Search for a Customer:{{image:Prototyping - 3.png|none|frame|479x479px}} View or edit a customer: {{image:Prototyping - 5.png|none|frame|483x483px}} View or edit a Rental contract: {{image:Prototyping - 6.png|none|frame|485x485px}} 4. What actions do we need to expose to the user for navigation between these views? For this, start by clicking Create/Init standard actions in the ActionsDefinition dialog. This gives you ordinary actions like save, quit, undo, and redo. It also picks up on the ViewModels you have and adds some actions for them. {{image:Prototyping - 7.png|none|frame|481x481px}} Remove the**

actions created for **ViewAndEditCustomer** and **ViewAndEditRentalContract** since these are rooted views that require a root object (a customer or a Rental contract) to have something to show. Instead, add a ViewModelActions in the SearchForCustomer ViewModel that creates a new Customer and one that creates new rental contracts: {{image:Prototyping - 8.png|none|frame|473x473px}} Also, add class actions that can show an existing customer and rental contract: {{image:Prototyping - 9.png|none|frame|478x478px}} 6. I could think a bit harder, but the whole point with prototyping is that it should be easy to test if we are done or not. So I hit the start prototype symbol: {{image:Prototyping - 10.png|none|frame|482x482px}} I am presented with a choice of how to store data for this prototype. We will choose XML for starters: {{image:Prototyping - 11.png|none|frame|408x408px}} I start the prototype and find the main menu items: {{image:Prototyping - 12.png|none|frame}} I pick "Search for customers" and I search. When the system finds none, I click the action new customer and I get the ViewAndEditCustomer view. I enter a name and save and click 'back'. Now, the search result sees a customer: {{image:Prototyping - 13.png|none|frame}} Note to self: I must have a Search for rental contracts. I create a new rental contract and notice that the button "Assign customer" does nothing. Note to self: I must bring up "Search for Customer" on "Assign Customer" and let the user pick 1 customer to return back with and set them on the "rental contract". Note to self: I must do the same for "Assign Car" and bring up "Search for cars". Note to self: I do not have any data for cars or brands and no UI to enter it with. Note to self: I should do this in the debugger window for now. This is how a typical prototyping session goes and how it reveals the obvious things we need to do. Let's do them now. Start

with using the debugger for adding cars and brands: {{image:Prototyping - 14.png|none|frame|457x457px}} I add a brand as well then save the data to the prototyping XML file by switching to dirty objects: {{image:Prototyping - 15.png|none|frame|472x472px}} I click to open the Autoform of a Car-object. I drag a Brand object to the Brand field: {{image:Prototyping - 16.png|none|frame|533x533px}} That takes care of test data for Cars and Brands. I still have these two items: Note to self: I must bring up "search for customer" on "assign customer" – then let the user pick 1 customer to return back with and set on the "rental contract". Note to self: I must do the same for "assign car" – I must bring up "search for cars". I do this by adding a ViewModelAction for the ViewOrEditRentalContract: {{image:Prototyping - 17.png|none|frame|444x444px}} The Action should bring up the search for Customer. It should be a modal action and be fine to press "ok" once a Customer is selected. When Ok is executed, we assign the picked customer to our rental contract. The same more or less for picking a car: {{image:Prototyping - 18.png|none|frame|425x425px}} I choose to hook these actions up to the buttons I put in the ViewModel: {{image:Prototyping - 19.png|none|frame|439x439px}} I also have this one left: Note to self: I must have a Search for rental contracts. Yet another ViewModel. Once I have it done, I use the shortcut action to create a global action to show it: {{image:Prototyping - 20.png|none|frame|438x438px}} Then I press Play again: This time, I can assign a Customer and assign a Car. The search for Car comes up with an Ok/Cancel button because it was brought up by a modal action: {{image:Prototyping - 21.png|none|frame|431x431px}} I search, pick a car, and press ok: {{image:Prototyping - 22.png|none|frame|460x460px}} I still have a few user stories left. I will need to change the model somewhat,

amend the ViewModels a bit, and maybe create some actions. All in all, this is a very straightforward way to work. You get instant gratification when you see your model and logic come to life. This is something that will also trigger ideas for further things your users will need or want. Churn on like this for an hour or two and you will have done more than what you would do in a day with traditional specification work or coding. The Look The prototyper window uses Windows Presentation Foundation (WPF). You are free to change the used style sheet as you see fit: {{image:Prototyping - 23.png|none|frame|488x488px}} Or: {{image:Prototyping - 24.png|none|frame|493x493px}} Or maybe you need to stress that you are "just prototyping" by using something really bubbly: {{image:Prototyping - 25.png|none|frame|501x501px}}The MDriven Book - Next Chapter: Available Actions

The MDriven Book

The look

Write the content here to display this box Used in all lines of engineering, prototyping is the process of whipping something together quickly as a mock-up to show or test to learn something that otherwise would be hard to know. Prototyping for software has a unique position. The prototype is composed of the same elements as the finished product: logical rules. This is untrue for any other practice of engineering. We have a unique position where we can harvest to make the real deal – the finished product – just as easily as if we were building a prototype. Many experienced developers frown at this. They know how prudent you must be to build something robust and solid. They also know how quick and dirty the prototype is when you're putting it together. Why does this huge gap exist? My answer: normal coding leaves too many degrees of freedom for the task at hand. Normal coding uses only one tool – code – for handling, presenting, and navigating data. In addition, you will mix things up when you are in a hurry. If we can separate things like the information we handle, from the transformation of that information into views and from the navigation between these views – then we have something that will almost fly on its own – and building something will be like prototyping it. Think of it as an autopilot; it will protect you against making stupid mistakes, help you fly straight, and give you time to talk on the radio or more – you only need to tell it where you want to fly. With MDriven, we use the model to tell the autopilot what information to handle, what the perspectives of the information should be, and how to navigate between the views of information. Having instructed the autopilot, we can take off and see if we have the right model or not. If not, we churn away and

fix it. Many developers also frown at the idea of autopilot since it will emasculate them and stop them from flexing their muscles. I need to point out that what makes MDrivenFramework great is that you can turn off the autopilot for any portion of the flight so to speak. You can still do some cool looping in front of a gaping audience whenever you feel like it, before turning on autopilot to do the mundane flight back home. This will give you freedom, speed, and fewer accidents. This is How You Do It With MDriven 1. Model what you know so far. In the example below, I am prototyping for a Car Rental Service. {{image:Prototyping - 1.png|none|frame|527x527px}} 2. Think about what user stories or requirements you have:As a Customer, I want to see what cars you can rent.'As a Customer, I want to know what they cost per day and what my total cost will be.'As a Rental worker, I want to be able to hand customers rental contracts to sign.As a Rental worker, I want to be able to find free cars.3. Then, create some ViewModels that can cover these user stories. When prototyping, make use of the scaffolding user interface hints that place out UI controls on a screen surface that matches the types of the Viewmodels properties. I end up with 4 ViewModels:Search for a Car: {{image:Prototyping - 2.png|none|frame|478x478px}} Search for a Customer:{{image:Prototyping - 3.png|none|frame|479x479px}} View or edit a customer: {{image:Prototyping - 5.png|none|frame|483x483px}} View or edit a Rental contract: {{image:Prototyping - 6.png|none|frame|485x485px}} 4. What actions do we need to expose to the user for navigation between these views? For this, start by clicking Create/Init standard actions in the ActionsDefinition dialog. This gives you ordinary actions like save, quit, undo, and redo. It also picks up on the ViewModels you have and adds some actions for them. {{image:Prototyping - 7.png|none|frame|481x481px}} Remove the

actions created for **ViewAndEditCustomer** and **ViewAndEditRentalContract** since these are rooted views that require a root object (a customer or a Rental contract) to have something to show. Instead, add a ViewModelActions in the SearchForCustomer ViewModel that creates a new Customer and one that creates new rental contracts: {{image:Prototyping - 8.png|none|frame|473x473px}} Also, add class actions that can show an existing customer and rental contract: {{image:Prototyping - 9.png|none|frame|478x478px}} 6. I could think a bit harder, but the whole point with prototyping is that it should be easy to test if we are done or not. So I hit the start prototype symbol: {{image:Prototyping - 10.png|none|frame|482x482px}} I am presented with a choice of how to store data for this prototype. We will choose XML for starters: {{image:Prototyping - 11.png|none|frame|408x408px}} I start the prototype and find the main menu items: {{image:Prototyping - 12.png|none|frame}} I pick "Search for customers" and I search. When the system finds none, I click the action new customer and I get the ViewAndEditCustomer view. I enter a name and save and click 'back'. Now, the search result sees a customer: {{image:Prototyping - 13.png|none|frame}} Note to self: I must have a Search for rental contracts. I create a new rental contract and notice that the button "Assign customer" does nothing. Note to self: I must bring up "Search for Customer" on "Assign Customer" and let the user pick 1 customer to return back with and set them on the "rental contract". Note to self: I must do the same for "Assign Car" and bring up "Search for cars". Note to self: I do not have any data for cars or brands and no UI to enter it with. Note to self: I should do this in the debugger window for now. This is how a typical prototyping session goes and how it reveals the obvious things we need to do. Let's do them now. Start

with using the debugger for adding cars and brands: {{image:Prototyping - 14.png|none|frame|457x457px}} I add a brand as well then save the data to the prototyping XML file by switching to dirty objects: {{image:Prototyping - 15.png|none|frame|472x472px}} I click to open the Autoform of a Car-object. I drag a Brand object to the Brand field: {{image:Prototyping - 16.png|none|frame|533x533px}} That takes care of test data for Cars and Brands. I still have these two items: Note to self: I must bring up "search for customer" on "assign customer" – then let the user pick 1 customer to return back with and set on the "rental contract". Note to self: I must do the same for "assign car" – I must bring up "search for cars". I do this by adding a ViewModelAction for the ViewOrEditRentalContract: {{image:Prototyping - 17.png|none|frame|444x444px}} The Action should bring up the search for Customer. It should be a modal action and be fine to press "ok" once a Customer is selected. When Ok is executed, we assign the picked customer to our rental contract. The same more or less for picking a car: {{image:Prototyping - 18.png|none|frame|425x425px}} I choose to hook these actions up to the buttons I put in the ViewModel: {{image:Prototyping - 19.png|none|frame|439x439px}} I also have this one left: Note to self: I must have a Search for rental contracts. Yet another ViewModel. Once I have it done, I use the shortcut action to create a global action to show it: {{image:Prototyping - 20.png|none|frame|438x438px}} Then I press Play again: This time, I can assign a Customer and assign a Car. The search for Car comes up with an Ok/Cancel button because it was brought up by a modal action: {{image:Prototyping - 21.png|none|frame|431x431px}} I search, pick a car, and press ok: {{image:Prototyping - 22.png|none|frame|460x460px}} I still have a few user stories left. I will need to change the model somewhat,

amend the ViewModels a bit, and maybe create some actions. All in all, this is a very straightforward way to work. You get instant gratification when you see your model and logic come to life. This is something that will also trigger ideas for further things your users will need or want. Churn on like this for an hour or two and you will have done more than what you would do in a day with traditional specification work or coding. The Look The prototyper window uses Windows Presentation Foundation (WPF). You are free to change the used style sheet as you see fit: {{image:Prototyping - 23.png|none|frame|488x488px}} Or: {{image:Prototyping - 24.png|none|frame|493x493px}} Or maybe you need to stress that you are "just prototyping" by using something really bubbly: {{image:Prototyping - 25.png|none|frame|501x501px}}The MDriven Book - Next Chapter: Available Actions
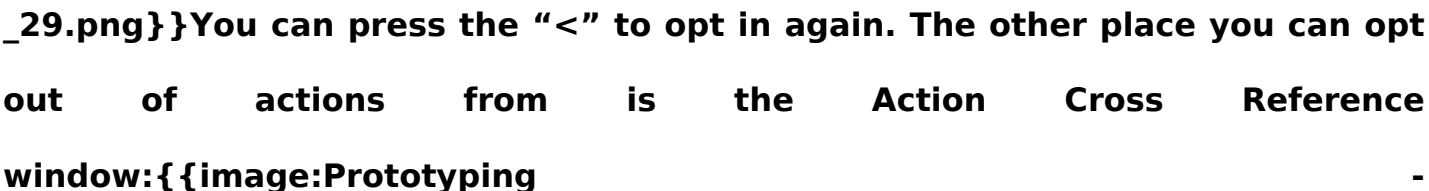
Available Actions

Write the content here to display this box The actions shown to the left in the image below are calculated by the Framework. You can have class actions – actions associated with a class in your model. These will show up whenever an object of that class is shown in your view. * You can also have ViewModel actions and these only show up in the view they are defined for. * You have the ability to instruct the logic to make exceptions for the calculated display of actions. You can opt out of the presentation of actions per view. * Actions may be used for navigation, but they may also perform something – like calling a method on an object. In our example, we already have some actions – both ViewModel actions, Class actions, and Global actions (the ones that build up the main menu).{{image:Prototyping - 26.png|none|frame|link=https://wiki.mdriven.net/index.php/File:Prototyping_-_26.png}}Look closely at the ShowRentalContract action. This is a Class action – available everywhere a RentalContract is shown. However, there are situations where we do not want it to show – like when we already are in the view that is brought up by the action:{{image:Prototyping - 27.png|none|frame|link=https://wiki.mdriven.net/index.php/File:Prototyping_-_27.png}}To remove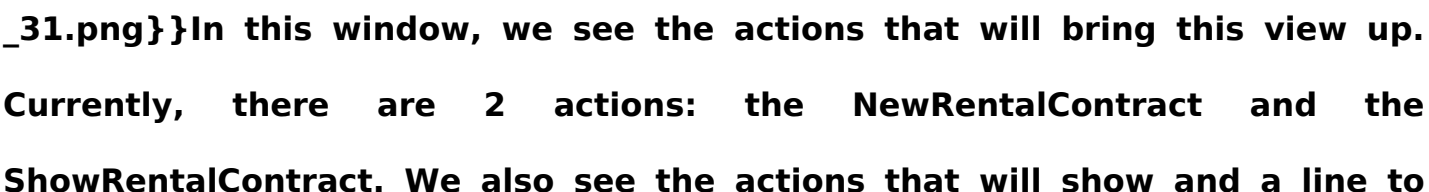 it, we want to opt out. There are two tools in MDrivenDesigner that are good to use for this. The first one is the ViewModelEditor:{{image:Prototyping - 28.png|none|frame|link=htt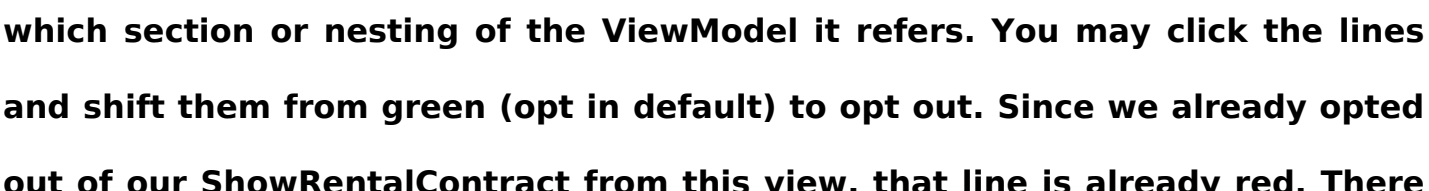ps://wiki.mdriven.net/index.php/File:Prototyping_-_28.png}}Pressing the ">" button will move it to the opt out column:{{image:Prototyping - 29.png|none|frame|link=https://wiki.mdriven.net/index.php/File:Prototyping_-
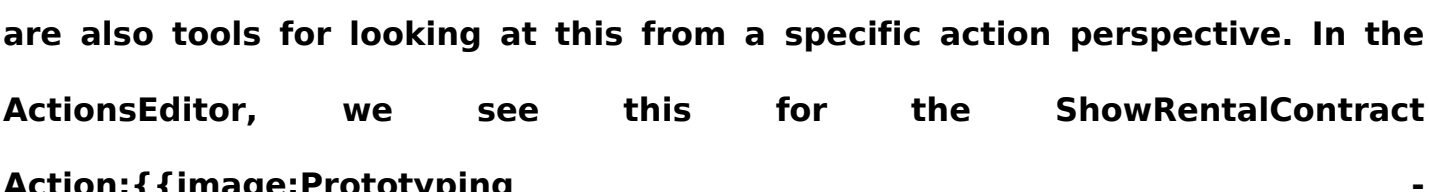
29.png}}You can press the "<" to opt in again. The other place you can opt out of actions from is the Action Cross Reference window:{{image:Prototyping - 30.png|none|frame|link=https://wiki.mdriven.net/index.php/File:Prototyping- _30.png}}{{image:Prototyping - 31.png|none|frame|link=https://wiki.mdriven.net/index.php/File:Prototyping_- _31.png}}In this window, we see the actions that will bring this view up. Currently, there are 2 actions: the NewRentalContract and the ShowRentalContract. We also see the actions that will show and a line to which section or nesting of the ViewModel it refers. You may click the lines and shift them from green (opt in default) to opt out. Since we already opted out of our ShowRentalContract from this view, that line is already red. There are also tools for looking at this from a specific action perspective. In the ActionsEditor, we see this for the ShowRentalContract Action:{{image:Prototyping - 32.png|none|frame|link=https://wiki.mdriven.net/index.php/File:Prototyping_- _32.png}}It says "Action shows in 2 places. 1 Opted out" followed by a dialog button. Click that and you will see this:{{image:Prototyping - 33.png|none|frame|link=https://wiki.mdriven.net/index.php/File:Prototyping_- _33.png}}Here we see that it is opt out in the ViewAndEditRentalContract view, but it shows 2 times in the SearchForRentalContracts view. I switch back to prototyping to verify this:{{image:Prototyping - 34.png|none|frame|link=https://wiki.mdriven.net/index.php/File:Prototyping_- _34.png}}And find that this is true. The reason for this is the fact that this ViewModel defines two areas that have the type RentalContract: one for the root and one for the Grid-nesting that shows the results from the variable

vSeekerResult.{{image:Prototyping - 35.png|none|frame|link=https://wiki.mdriven.net/index.php/File:Prototyping_-_35.png}}The root instance will always be null since this ViewModel does not require a root object to function. ViewModels designed for seeking seldom do. If it always will be null then there is little point in having an action that will be enabled only if it is assigned to an object. We should opt that root action out. Click the line to toggle the opt out state:{{image:Prototyping - 36.png|none|frame|link=https://wiki.mdriven.net/index.php/File:Prototyping_-_36.png}}The ActionEditor was updated as well:{{image:Prototyping - 37.png|none|frame|link=https://wiki.mdriven.net/index.php/File:Prototyping_-_37.png}}It now says the Action shows in 1 place – 2 opted out. I have had my prototyping session running all along – but it still uses the model we had prior to our changes. I can now restart the prototyping by clicking Play again, bringing up a new prototype window. Or I can just reread the model to the one I have. I see that the ShowRentalContract action is shown only once.{{image:Prototyping - 38.png|frame|link=https://wiki.mdriven.net/index.php/File:Prototyping_-_38.png|none}}If you are alone for a day and you have lots of ideas, I assure you that you can model them in MDriven Designer and verify them in MDriven Prototyper. When you are able to try and verify ideas rapidly, you will find many "think bugs" early. Bugs that make you say things like: "Oh, no – that is not a good way of doing it!". Since the MDriven environment works as your autopilot and it has tens of thousands of flight hours under its belt, you will be freer in the creative part of your work. The MDriven Book - Next Chapter: MDriven Server Introduction

The MDriven Book

Introducing MDriven Server

The MDriven Book

Security concerns for MDriven Server

Write the content here to display this box When you install your MDriven Server, access it by registering a new user. There are more things to consider, however. To secure your model and data and system, you can: # Make sure you communicate with MDriven Server over HTTPS so that no one sees your passwords and other data that will go over the wire. # Limit what an unauthenticated user of MDriven Server can do. {{image:MDriven security 01.png|frameless|252x252px}} ☞ {{image:MDriven security 02.png|frameless|363x363px}} {{image:MDriven security 03.png|frameless}} ☞ {{image:MDriven security 04.png|frameless|422x422px}} In the user admin dialog, state that the Admin UI requires identification. If you do this – and you should at some point – make sure you make yourself SuperAdmin so you do not lock yourself out. You can also state whether the services exposed by the MDriven Server via various web interfaces require authentication or not. You will likely begin with a relaxed attitude to security - this will put fewer requirements on the users you engage in prototyping etc. Understand that no security limitations are enforced as long as you run your server in HTTP mode – because this would force us to send passwords over an open wire which is considered unsafe since it may implicate other services you have. The MDriven Book - Next Chapter: MDrivenServer Summarized {{Edited|July|12|2024}}

The MDriven Book

MDrivenServer Summarized

Write the content here to display this box The MDrivenServer receives your model from MDrivenDesigner. As it does, it will create or evolve the database it uses to store your data. This database is default a SQLServer Compact edition that gets installed along the MDrivenServer. You can however change the database used for your data to a SQLServer in Azure. You change the database used 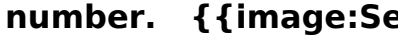simply by providing a connection string like this: {{image:MDrivenServerSummarized.png|none|thumb|696x696px}} The MDriven Server allows you to access the data in your database through various interfaces. The most advanced interface is the MDriven Framework PersistenceMapper API. This API is easily used with MDriven Framework-built applications. The MDriven Frameworks PersistenceMapper API is a secure and robust way to query and retrieve your data in a multi-user environment. It allows for functions like optimistic locking, client synchronization, CRUD operations in transactions, OCL queries executed as SQL in the database, etc. The MDrivenServer also offers alternate methods to get access to your data via Json objects so that you can build non-MDriven, savvy applications that do CRUD operations on the data in your model. As the MDrivenServer has interfaces to receive model updates from the MDrivenDesigner, the development loop from idea to deploy can be very quick. The MDriven Book - Next Chapter: MDrivenServer periodic server-side actions {{Edited|July|12|2024}}

The MDriven Book

MDrivenServer periodic server-side actions

Write the content here to display this box Running Background Jobs on the Server A recurring pattern when building multi-user software systems is the need to execute periodic actions. With MDrivenServer, you can execute recurring actions or periodic actions. A Periodic action is defined by a selection expression that selects what objects to act on. For each object selected for a periodic action, we want to do the "action" and that probably needs a cluster of objects to do different stuff to evolve some object state. To enable efficient load for such an object cluster, you must define and associate a ViewModel with each periodic action. * The periodic action logic will load the ViewModel for your object and loop through all actions it finds in its root ViewModel class. When all actions are executed, the periodic action logic will save any changed state that was the result of your actions. * Periodic actions can be used to "automatically" step your information from one state to the next – given that the circumstances are correct. This ability may take some getting used to, but it can be used for things like assigning a unique number to an order or an article in your domain model – or for actions that need to be done serverside for some particular reason. Define the periodic actions in the ViewModelEditor: {{image:ServerSide Actions 01.png|frameless|246x246px}} ☛ {{image:ServerSide Actions 02.png|frameless|464x464px}} With this technique, you can change any state of the objects in your database efficiently with very little code. A common case is the need to assign a unique number. I will demonstrate this. The client alone cannot be used to guarantee that the next number in a sequence is taken, since there may be multiple users trying to do this at the same time. You need to ensure you serialize all

user requests for a new number. This is commonly done with a database lock. There is, however, a nice alternative to a harsh DB-locking technique and that is to serialize via a server-side action. Consider having this State machine: {{image:ServerSide Actions 03.png|frameless|388x388px}} The ViewModel associated with the periodic action on the server can now assign your number. {{image:ServerSide Actions 04.png|frameless|550x550px}} The client will set the AssignNumber state. The Server side action will look for Order.allinstances- >select(o|o.State='AssignNumber') When objects are found, they will be assigned a new number by the actions in the ServerSide ViewModel and saved. Combined with a periodic client action that calls selfVM.Refresh, we will see the client as soon as the state is changed to NumberAssigned. Since we do not want the client to poll selfVM.Refresh all the time, we set the EnableExpression in this action to self.State='AssignNumber'. Following this pattern, we get the user to push a button to get a new number and we can show some info about working. The client starts refreshing, the server does its job – the client gets the new info by its refresh – the info about progress can be hidden as a consequence. This way, we have serialized an action in our system with model-driven techniques that will scale well and work for any number of users of your system. New Recommendation on Number Assignment Pattern Do as shown above but mark the state attribute and the number attribute with tagged value Realtime. This removes the need for periodic action with Refresh on the client. Combine this with triggering via SysAsyncTicket and lower/remove the frequency of the ServerSide-job execution. Other Uses of Server-side Actions We have also implemented several additional common actions you can have the MDrivenServer perform for you. Emailing from the Server This topic is
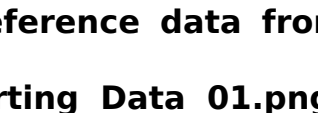
The MDriven Book

described here: Emailing from an app using MDrivenServer Importing Data From Other SQL Sources This topic is described here: Import_data_from_other_SQL_servers Exporting Files From MDriven Server This topic is described here: More about exporting Catching Errors and Debug Info For Server-side Actions Debugging serverside is covered here: Debugging MDrivenServer Serverside actions Environment Specific Execution Read on Server-wide variables here: Server_Wide_Variables The MDriven Book - Next Chapter: SQLExport from MDriven Server

Other uses of Server side Actions

The MDriven Book

Emailing from the server

The MDriven Book

Importing data from other SQL sources

Write the content here to display this box MDriven Server has been extended with functionality to import data from other SQL-based systems. The MDriven Server is designed to take care of the repetitive common tasks that always seem to come back and haunt system developers. MDriven Server takes a strictly model-driven approach to help you with the work. Earlier, I described how to export files: Exporting files from MDriven Server#Producing export files from MDriven Server and here is a link to where I explain more about the concept of periodic actions: MDrivenServer periodic server-side actions What is new today is the ability to read from an external SQL server and import that data – strictly by using MDriven techniques - with zero need for external programs. Let me show you. Suppose I have this model and I want Class1 to be reference data from an external database: {{image:ServerSide Actions Importing Data 01.png|frameless|257x257px}} I declare a ViewModel that looks like this: {{image:ServerSide Actions Importing Data 02.png|frameless|469x469px}} It defines 4 columns with data and 2 actions: 1=ViewModel / new Nesting – the name of yet another ViewModel that will act as an importer of the SQL result set, this can, in recent versions, be replaced with Nesting and then you are expected to have a connected nesting with that name in the ViewModel that has the Import action. This Nesting is then defining the columns to import. {{image:2019-02-05 15h42 14.png|none|thumb|This shows how the Nesting column ties to Nesting inside the same ViewModel. }} 2=Connectionstring- the external database (Update 2018-10-18: you can now use 'connectionstringodbc' and the logic will use ODBC connection instead of sqlserver.) 3=Query – the SQL query (remember

that you can build it with data from the rest of your model). 4=Key – if we want the import to be able to update Class1, we need to explain what the key is in the class. The value of the key should be a string with the value of the name of the attribute we want to use as a key in the class. This attribute must also be the result of the first column on the import nesting. And the actions: SQLImport – using this name will trigger the import function in MDriven Server. Finished – this is a generic action that executes the expression - in this case setting Class2.Attribute1 to 'Done'. So the returned SQL data looks like this: {{image:ServerSide Actions Importing Data 03.png|frameless|323x323px}} And the ViewModel that is going to act as the import template – called "TheImporter" in the example above - looks like this: {{image:ServerSide Actions Importing Data 04.png|frameless}} I now declare the ServerSide job in MDriven Server: {{image:ServerSide Actions Importing Data 05.png|frameless|422x422px}} Now the MDriven Server will check every 20 seconds if the expression: Class2.allinstances- >select(attribute1='todo') returns any rows. If it does, it fetches at most 2 of these and executes all the actions found in TheServerSideJob. In MDriven Designer, I can create a Class2 with the debugger and save it: {{image:ServerSide Actions Importing Data 06.png|frameless|324x324px}} And then I check the MDriven Server log: {{image:ServerSide Actions Importing Data 07.png|frameless|493x493px}} I check my Class2: {{image:ServerSide Actions Importing Data 08.png|frameless|448x448px}} Attribute1 is now 'Done' – so the serverside job relaxes and will not find anything more to do for now. Rewriting From Old Style 2 ViewModels to New Style 1 ViewModel with Nesting The old style was to have a ViewModel column pointing out the description of the import row. The new more compact style is to instead refer to a Nesting within the main

ViewModel doing the import. Example of the new style: {{image:2019-05-15 10h38 46.png|none|thumb|610x610px}} Example of the old style: {{image:2019-05-15 10h41 20.png|none|thumb|980x980px}}Make sure the columns in the import are not left as read-only since they are created that way by default. If they are read-only, they will not receive any values: {{image:2020-04-21 11h47 05.png|none|thumb|939x939px}}The MDriven Book - Next Chapter: Exporting files from MDriven Server

The MDriven Book

Producing export files from MDriven Server

Write the content here to display this box {| class="wikitable" !Special column name ! |- |filename ! |- |path | |- |xslt |also see the XsltTransformXml OCL operator |- |filedata | |- |savefile | |} **Producing Export Files From MDriven Server You can also write files by adding an action column called "savefile" (case sensitive). When the periodic action supervisor in MDrivenServer sees this action column, it looks for two additional columns: path and filename (case insensitive). If these are found, the complete result of the ViewModel is streamed out as XML by using the ViewModelXMLUtils.GetDataAsXml method. The XML is then saved to path/filename. Like this: {{image:Exporting files from MDriven Server.png|frameless|360x360px}} In a real-world application, it looks like this: {{image:Exporting files from MDriven Server 02.png|frameless|362x362px}} Notice that both the path and filename are evaluated expressions. Furthermore, the Action UpdateExportTime is executed last – and it changes some data. Since the whole execution is wrapped in a memory transaction, we make sure that we do not update the export time unless the file was written and everything is ok. The resulting file looks like this: {{image:Exporting files from MDriven Server 03.png|frameless|409x409px}} Shaping and Transforming Export Files This SaveFile action has been updated to look for a column named XSLT. If it is found, the contents of the field are assumed to be a valid XSLT transformation. The XML from the ViewModel is transformed with the XSLT and the result is saved as above. So if your XSLT is… , … then your output would be… NO101087,Cosmica Tygg 90 ST… New Addition March 2019: Arbitrary Filedata Add a ViewModel column named "filedata". This results in**

an Image, Blob, or other byte array type, and this data is saved rather than the ViewModel-data-as-XML. If the "filedata"-column is of type string, the string content is written to the file. The MDriven Book - Next Chapter: SQLExport

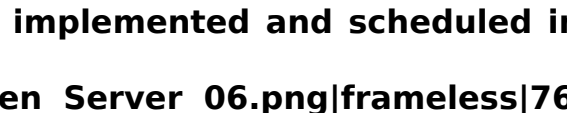# The MDriven Book

Shaping and transforming export files

Write the content here to display this box {| class="wikitable" !Special column name ! |- |filename ! |- |path | |- |xslt |also see the XsltTransformXml OCL operator |- |filedata | |- |savefile | |} Producing Export Files From MDriven Server You can also write files by adding an action column called "savefile" (case sensitive). When the periodic action supervisor in MDrivenServer sees this action column, it looks for two additional columns: path and filename (case insensitive). If these are found, the complete result of the ViewModel is streamed out as XML by using the ViewModelXMLUtils.GetDataAsXml method. The XML is then saved to path/filename. Like this: {{image:Exporting files from MDriven Server.png|frameless|360x360px}} In a real-world application, it looks like this: {{image:Exporting files from MDriven Server 02.png|frameless|362x362px}} Notice that both the path and filename are evaluated expressions. Furthermore, the Action UpdateExportTime is executed last – and it changes some data. Since the whole execution is wrapped in a memory transaction, we make sure that we do not update the export time unless the file was written and everything is ok. The resulting file looks like this: {{image:Exporting files from MDriven Server 03.png|frameless|409x409px}} Shaping and Transforming Export Files This SaveFile action has been updated to look for a column named XSLT. If it is found, the contents of the field are assumed to be a valid XSLT transformation. The XML from the ViewModel is transformed with the XSLT and the result is saved as above. So if your XSLT is… , … then your output would be… NO101087,Cosmica Tygg 90 ST… New Addition March 2019: Arbitrary Filedata Add a ViewModel column named "filedata". This results in

an Image, Blob, or other byte array type, and this data is saved rather than the ViewModel-data-as-XML. If the "filedata"-column is of type string, the string content is written to the file. The MDriven Book - Next Chapter: SQLExport

The MDriven Book

SQLExport serverside action

Write the content here to display this box SQLExport ServerSide Action The SQLExport action is used to write data from your model into another external SQL database. It is an alternative way to export data that needs no other external components or maintenance than what MDriven Server provides. SQLExport requires 'connectionstring' on root, 'data' (column name must start with 'data', case insensitive) as a Nesting that contains the data row(s) to insert or update, and on data - 'queryforinsert','queryforupdate' and 'queryforselectcount' as attributes (if queryforselectcount returns 0 the queryforinsert is used, else update). Update 2018-10-18: You can now use 'connectionstringodbc' and the logic will use ODBC connection instead. The very common case of "replicating" some model-driven data to another database is implemented and scheduled in minutes. {{image:Exporting files from MDriven Server 06.png|frameless|769x769px}} Common Gotchas The data nesting must be collected even when you work in 1 object -> do self->asset The MDriven Book - Next Chapter: OCL Expressions

Object Constraint Language

What is Object Constraint Language

The MDriven Book

Certain important constructs

Write the content here to display this box Some constructs are more returning than others as an everyday business developer with MDriven. Your favorite ways to express yourself may be different from mine but these are some of my returning expressions: {| class="wikitable" ! ! |- |let z= Things.allinstances- >select(x|x.someInt>3)->size in (If z>4 then 'There are more than 4 Things with SomeInt>3' else 'There are '+z.asstring+' Things with SomeInt>3' endif) |I use the "let" construct to assign a result of an expression to a temporary variable. This is so I do not need to repeat myself in the testing of z>4 and the z.asstring. |- |Thing.allinstances->groupby(x|x.SomeValue) |Groupby - this expression has the type Collection(SomeValue:ValueStore+List:Collection(Thing)) so I get a list of SomeValue and for each a list of the things that use it. |- |Thing.allinstances->collect(x|x.SomeValue.UsedB   y.SomeInt->sum,   x.SomeValue.UsedBy->collect(y|x, y.Details)) |Nested collecting. This expression gets the type Collection(Part1:System.Int32+Part2:Collection(Thing:Thing+Det ails:Collection(Detail))). The ability to nest collections is very powerful. In this case, I start with all Things – grab the SomeValue valueStore– check what other things have this set via the association MultiValuePick, and for these, I sum up all SomeValue plus grab the Details. This kind of multi-level collect-usage is very handy when summarizing deep object hierarchies on different levels. |- |' '.Chars(0) |This expression returns System.Char. Since OCL has no literal way to input a single character – it is always interpreted as string – this trick will help when calling certain .net functions that take characters as arguments. |- |if true then 'this returns a string' else 0.asstring

endif |All return paths must result in the same type. Since OCL is a functional language we must be consistent. This is one way to get the expression correct; add.asstring after the zero |- |Thing.allinstances- >select(someint>3) ValueStore.allinstances- >select(usedby- >notEmpty).Thing Like this: Thing.allinstances- >select(someint>3) ->intersection(ValueStore.allinstances- >select(usedby- >notEmpty).Thing) |Working with OCLps – expressions that will be translated into SQL and executed in a database- it is sometimes easier to do one expression per complex constraint and at the end intersect all the expressions together. |- |self |When you are in the context of an object, you can use the variable self to access the properties of this. |} You may define methods in classes too and implement these with OCL: {{image:Constructs image 1.png|frameless|402x402px}} You will in the OCL implementation in the Body-property: {{image:Constructs image 2.png|frameless|459x459px}} Notice that since this was a method, MDriven will treat your OCL as EAL – something that has side effects. In this case, our method does not have any side effects and I may want to be able to use this method in OCL. But trying to use it in OCL will not succeed. Methods with side effects are not recognized by OCL. There is a flag on the Method definition called IsQuery and if this is set we "promise" that it does not have intentional side effects. Now it is seen by OCL: {{image:Constructs image 3.png|frameless|464x464px}} We can then use our IsQuery method in any expression in OCL. Thing.allinstances->select(x|x.MyMethod(x.SomeInt)) Summary OCL I am often asked if OCL is capable of doing everything we need to do in a line of business application. The answer is that as long as the arguments and result are representable in your model – yes it will do anything. Sometimes you have external or ambient data not accessible from

the model – then you cannot use OCL – until you make that data available. Not only can you do everything you need – it also comes out in small easily interpreted snippets of text that very much looks just like the requirements you are set to implement. I like to compare OCL and modeling with Calculus. In math you can discuss numbers and operators on those numbers in plain language – but you seldom do since it will be error prone and require you to use a lot of words for even simple things. Instead, everyone actually doing math uses calculus notation to write up expressions. The expressions are often reduced to the smallest possible – so that they are easily understood and ready to be used for a purpose. Use OCL for the same reason but not on only numbers but on all your designed information. Imagine a world without a good way to declaratively work with math. In this world, we would probably not have been able to do much cool technology. The ability to convey compact math between people is very good for mankind. I am certain that a good compact way to convey rules on information is equally important – if not even more usable – for mankind. The MDriven Book - Next Chapter: Seeker view

The MDriven Book

EAL differences

OCLps differences

The MDriven Book

Summary OCL

Write the content here to display this box Some constructs are more returning than others as an everyday business developer with MDriven. Your favorite ways to express yourself may be different from mine but these are some of my returning expressions: {| class="wikitable" ! ! |- |let z= Things.allinstances- >select(x|x.someInt>3)->size in (If z>4 then 'There are more than 4 Things with SomeInt>3' else 'There are '+z.asstring+' Things with SomeInt>3' endif) |I use the "let" construct to assign a result of an expression to a temporary variable. This is so I do not need to repeat myself in the testing of z>4 and the z.asstring. |- |Thing.allinstances- >groupby(x|x.SomeValue) |Groupby - this expression has the type Collection(SomeValue:ValueStore+List:Collection(Thing)) so I get a list of SomeValue and for each a list of the things that use it. |- |Thing.allinstances- >collect(x|x.SomeValue.UsedB   y.SomeInt->sum,   x.SomeValue.UsedBy- >collect(y|x, y.Details)) |Nested collecting. This expression gets the type Collection(Part1:System.Int32+Part2:Collection(Thing:Thing+Det ails:Collection(Detail))). The ability to nest collections is very powerful. In this case, I start with all Things – grab the SomeValue valueStore– check what other things have this set via the association MultiValuePick, and for these, I sum up all SomeValue plus grab the Details. This kind of multi-level collect-usage is very handy when summarizing deep object hierarchies on different levels. |- |' '.Chars(0) |This expression returns System.Char. Since OCL has no literal way to input a single character – it is always interpreted as string – this trick will help when calling certain .net functions that take characters as arguments. |- |if true then 'this returns a string' else 0.asstring

endif |All return paths must result in the same type. Since OCL is a functional language we must be consistent. This is one way to get the expression correct; add.asstring after the zero |- |Thing.allinstances- >select(someint>3) ValueStore.allinstances- >select(usedby- >notEmpty).Thing Like this: Thing.allinstances- >select(someint>3) ->intersection(ValueStore.allinstances- >select(usedby- >notEmpty).Thing) |Working with OCLps – expressions that will be translated into SQL and executed in a database- it is sometimes easier to do one expression per complex constraint and at the end intersect all the expressions together. |- |self |When you are in the context of an object, you can use the variable self to access the properties of this. |} You may define methods in classes too and implement these with OCL: {{image:Constructs image 1.png|frameless|402x402px}} You will in the OCL implementation in the Body-property: {{image:Constructs image 2.png|frameless|459x459px}} Notice that since this was a method, MDriven will treat your OCL as EAL – something that has side effects. In this case, our method does not have any side effects and I may want to be able to use this method in OCL. But trying to use it in OCL will not succeed. Methods with side effects are not recognized by OCL. There is a flag on the Method definition called IsQuery and if this is set we "promise" that it does not have intentional side effects. Now it is seen by OCL: {{image:Constructs image 3.png|frameless|464x464px}} We can then use our IsQuery method in any expression in OCL. Thing.allinstances->select(x|x.MyMethod(x.SomeInt)) Summary OCL I am often asked if OCL is capable of doing everything we need to do in a line of business application. The answer is that as long as the arguments and result are representable in your model – yes it will do anything. Sometimes you have external or ambient data not accessible from

the model – then you cannot use OCL – until you make that data available. Not only can you do everything you need – it also comes out in small easily interpreted snippets of text that very much looks just like the requirements you are set to implement. I like to compare OCL and modeling with Calculus. In math you can discuss numbers and operators on those numbers in plain language – but you seldom do since it will be error prone and require you to use a lot of words for even simple things. Instead, everyone actually doing math uses calculus notation to write up expressions. The expressions are often reduced to the smallest possible – so that they are easily understood and ready to be used for a purpose. Use OCL for the same reason but not on only numbers but on all your designed information. Imagine a world without a good way to declaratively work with math. In this world, we would probably not have been able to do much cool technology. The ability to convey compact math between people is very good for mankind. I am certain that a good compact way to convey rules on information is equally important – if not even more usable – for mankind. The MDriven Book - Next Chapter: Seeker view

Seeker view

Write the content here to display this box Definition A Seeker is a special ViewModel for searching and finding things in the database. Introduction to SeekLogic To get anything done, you need to find the necessary things. A typical software system has the same need. How do we go about declaring a non-limiting, multi-variable, user-friendly seeker into a generic model-driven system like the ones we build with MDriven? This is what we need: # We need unrooted (i.e. not having anything to start with) persistent storage evaluated OCL expressions to execute a search. # We need user input on how to limit the search. # We need to allow the user to use different limiting criteria as he or she sees fit; after all, the One-Field-Matches-Everything tactic that Google uses does not cut the mustard in enterprise applications. Users will want to limit the search to "Only this department", "Only things from yesterday" etc, and even in Google, you need to use an extended syntax to get this done. # We need to allow multiple sets of search definitions per seeker interface. If the user does not get a hit using the filters with the first set, it is natural for the user to "try again". Then, we want to use another set of search criteria; this has been tested on real users and many find it intuitive and obvious that it should work this way. Consider this model: {{image:Model image.png|frameless|356x356px}} This is how we can define a Seeker: {{image:Image 3 model.png|frameless|564x564px}} #1 We declare variables that hold the user input – one variable is a string and the other is an Object reference of type ReferenceClass (from our model). #2 We add Columns that use these variables so that we get some UI for the user to enter the criteria into. The Reference value we set up as a PickList. We create a

The MDriven Book

SearchExpression - right-click menu on the ViewModelColumn - Add nested – Add search expr. When adding the first search expression to the ViewModel, the Designer will also add the default implementation details with a vSeekerResult variable and a SeekString. This is intended to help - there is nothing magical about the added widgets. The only important thing in a Seeker is that there is a variable named vSeekerResult, which is of type collection. #3 Add two criteria. The result of the two criteria will be intersected (on the server side). Here is an important fact: Since the result of the criteria will be intersected, we need some way to say if a criterion is Active or not – after all, it is up to the user to limit the search on either or both of the two criteria. This is how the activation of a Criteria is done: {{image:Image 4 model.png|frameless|390x390px}} and {{image:Image 5 model.png|frameless|390x390px}} The Active Expression is optional. If you leave it blank, it defaults to true – always on. #4 The second batch of search expressions is executed the second time the user clicks the search button BUT only if the search variables have not changed. You can have as many search batches as you need, and they are Round-robin-used whenever the user clicks the search button with untouched variables. Whenever a variable is changed, the Round-robin is reset and the first batch is used again. Even if these rules may seem complex, they are intuitive for the user – especially if you use the search batches to filter for the same data as the resulting columns show. For example, the user enters someone's first name, but your first batch filters on the last name – the wrong people come up for the first search, and the user hits search again. Now we use the second batch where you filter on the first name – voila! This was just an example. You can create a filter expression that unions the first name and the last name results.

Person.allinstances->select(a|a.FirstName.SqlLike(vSeekString+'%'))->union(

Person.allinstances-   >select(a|a.LastName.SqlLike(vSeekString+'%')))

Another example might be that the user enters a number. First, we try to match it with a product code, the user hits search again, and we try to match it with the order number. The user is still not happy so he hits search again. Now we match it with the phone number of the customer – the user is happy. In this example, we could have chosen to create a detailed search interface with 3 text boxes – one for the product code, one for the order number, and one for the customer phone number – just as valid. Or we could do a union expression as above – just as valid. Choose the strategy that sits best with your users, but I urge you to test the simple interface with a single or only a few input boxes - it is user-friendly. Databases Use SQL The search expressions for OCL criteria are different from those of other ViewModel column expressions in one important aspect: they are executed against persistent storage (your database). If your database is an SQL-Server, these expressions will be translated to SQL and sent to the database for evaluation and execution. The database is much better and faster at handling huge volumes of data than MDriven. This means that we can filter out specific objects in our model from a nearly unlimited-sized database. The multi-variable seekers described in this chapter will be the natural starting point for your users' work in your system. They search – they find – and then they work. The work part is often rooted in a specific found object where your other ViewModels expand the neighboring information. Useful Search Examples Dates Dates are a little tricky because SQL servers encode dates in many formats. We need to pass dates to search for, not as text, but as values of the date type. So, when filtering on dates, use variables of the type

The MDriven Book

DateTime. We've created a variable vAfterDate here and are using it to search.

Person.allInstances->select(p|p.Registrered>=vAfterDate)->orderBy(p|p.Registrered) This will find all Persons registered after the given date and order them on the registered date. Need to filter on dates without the user seeing it? (To limit the search result maybe?) Set the DateTime variable before selfVM.Search on the search button like this: vAfterDate:= DateTime.Today.addMonths(-6); selfVM.Search Limiting the Number of Results By adding the tagged value MaxFetch, you can change the number of records shown before asking the user to extend their search. Also, see SeekMore logic with paging Search_result_pages. What Type is Searched - What Type is vSeekerResult? The vSeekerResult will be created for you as a collection of RootType of the ViewModel. If you want another type of vSeekerResult, you can declare the vSeekerResult as Collection(YourDesiredType), and then seeker logic will assume the search type is YourDesiredType. Order, OrderBy, OrderDescending in Seeker - Extensions added 2021-01-16 Since the MultiVariable seeker may combine expressions with ->intersection, it has not been very easy to get a good "order by" expression to work consistently over the many possible combinations of final expressions used. Furthermore, OclPS has been limited to using only the attributes on the main class for order by. To remedy these 2 challenges, we have extended OclPS - orderby and orderDescending - to support single link navigations to attributes used in order. We have introduced a new type of nesting in the ViewModel. This Nesting is a SearchExpression-nesting with a name starting with OrderExpression. If the SearchLogic finds this, we now grab the first active criteria from there and

extract the OrderBy/OrderDescending expression-part - this part is appended to the combined resulting expression from the original part of the searchlogic. The video discussing this can be found here: [https://www.youtube.com/watch?v=55K0irODBRE Wednesdays-with-MDriven | View Model Editor 2021: Part 3] {{image:2021-09-27 16h09 061.png|none|thumb|585x585px}}Seek form (web) * Use the tagged value Eco.HiliteGridColumn to signal the user to sort order/searched column that... * Find it by clicking on the orange search-expression and using the "Tagged value" button or the property inspector. The MDriven Book - Next Chapter: Efficient ViewModel fetch

Databases use SQL

Write the content here to display this box Definition A Seeker is a special ViewModel for searching and finding things in the database. Introduction to SeekLogic To get anything done, you need to find the necessary things. A typical software system has the same need. How do we go about declaring a non-limiting, multi-variable, user-friendly seeker into a generic model-driven system like the ones we build with MDriven? This is what we need: # We need unrooted (i.e. not having anything to start with) persistent storage evaluated OCL expressions to execute a search. # We need user input on how to limit the search. # We need to allow the user to use different limiting criteria as he or she sees fit; after all, the One-Field-Matches-Everything tactic that Google uses does not cut the mustard in enterprise applications. Users will want to limit the search to "Only this department", "Only things from yesterday" etc, and even in Google, you need to use an extended syntax to get this done. # We need to allow multiple sets of search definitions per seeker interface. If the user does not get a hit using the filters with the first set, it is natural for the user to "try again". Then, we want to use another set of search criteria; this has been tested on real users and many find it intuitive and obvious that it should work this way. Consider this model: {{image:Model image.png|frameless|356x356px}} This is how we can define a Seeker: {{image:Image 3 model.png|frameless|564x564px}} #1 We declare variables that hold the user input – one variable is a string and the other is an Object reference of type ReferenceClass (from our model). #2 We add Columns that use these variables so that we get some UI for the user to enter the criteria into. The Reference value we set up as a PickList. We create a

The MDriven Book

SearchExpression - right-click menu on the ViewModelColumn - Add nested – Add search expr. When adding the first search expression to the ViewModel, the Designer will also add the default implementation details with a vSeekerResult variable and a SeekString. This is intended to help - there is nothing magical about the added widgets. The only important thing in a Seeker is that there is a variable named vSeekerResult, which is of type collection. #3 Add two criteria. The result of the two criteria will be intersected (on the server side). Here is an important fact: Since the result of the criteria will be intersected, we need some way to say if a criterion is Active or not – after all, it is up to the user to limit the search on either or both of the two criteria. This is how the activation of a Criteria is done: {{image:Image 4 model.png|frameless|390x390px}} and {{image:Image 5 model.png|frameless|390x390px}} The Active Expression is optional. If you leave it blank, it defaults to true – always on. #4 The second batch of search expressions is executed the second time the user clicks the search button BUT only if the search variables have not changed. You can have as many search batches as you need, and they are Round-robin-used whenever the user clicks the search button with untouched variables. Whenever a variable is changed, the Round-robin is reset and the first batch is used again. Even if these rules may seem complex, they are intuitive for the user – especially if you use the search batches to filter for the same data as the resulting columns show. For example, the user enters someone's first name, but your first batch filters on the last name – the wrong people come up for the first search, and the user hits search again. Now we use the second batch where you filter on the first name – voila! This was just an example. You can create a filter expression that unions the first name and the last name results.

```
Person.allinstances->select(a|a.FirstName.SqlLike(vSeekString+'%'))->union(
Person.allinstances-          >select(a|a.LastName.SqlLike(vSeekString+'%')))
```

Another example might be that the user enters a number. First, we try to match it with a product code, the user hits search again, and we try to match it with the order number. The user is still not happy so he hits search again. Now we match it with the phone number of the customer – the user is happy. In this example, we could have chosen to create a detailed search interface with 3 text boxes – one for the product code, one for the order number, and one for the customer phone number – just as valid. Or we could do a union expression as above – just as valid. Choose the strategy that sits best with your users, but I urge you to test the simple interface with a single or only a few input boxes - it is user-friendly. Databases Use SQL The search expressions for OCL criteria are different from those of other ViewModel column expressions in one important aspect: they are executed against persistent storage (your database). If your database is an SQL-Server, these expressions will be translated to SQL and sent to the database for evaluation and execution. The database is much better and faster at handling huge volumes of data than MDriven. This means that we can filter out specific objects in our model from a nearly unlimited-sized database. The multi-variable seekers described in this chapter will be the natural starting point for your users' work in your system. They search – they find – and then they work. The work part is often rooted in a specific found object where your other Vie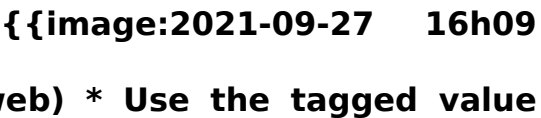wModels expand the neighboring information. Useful Search Examples Dates Dates are a little tricky because SQL servers encode dates in many formats. We need to pass dates to search for, not as text, but as values of the date type. So, when filtering on dates, use variables of the type

DateTime. We've created a variable vAfterDate here and are using it to search.

Person.allInstances->select(p|p.Registrered>=vAfterDate)->orderBy(p|p.Registrered) This will find all Persons registered after the given date and order them on the registered date. Need to filter on dates without the user seeing it? (To limit the search result maybe?) Set the DateTime variable before selfVM.Search on the search button like this: vAfterDate:= DateTime.Today.addMonths(-6); selfVM.Search Limiting the Number of Results By adding the tagged value MaxFetch, you can change the number of records shown before asking the user to extend their search. Also, see SeekMore logic with paging Search_result_pages. What Type is Searched - What Type is vSeekerResult? The vSeekerResult will be created for you as a collection of RootType of the ViewModel. If you want another type of vSeekerResult, you can declare the vSeekerResult as Collection(YourDesiredType), and then seeker logic will assume the search type is YourDesiredType. Order, OrderBy, OrderDescending in Seeker - Extensions added 2021-01-16 Since the MultiVariable seeker may combine expressions with ->intersection, it has not been very easy to get a good "order by" expression to work consistently over the many possible combinations of final expressions used. Furthermore, OclPS has been limited to using only the attributes on the main class for order by. To remedy these 2 challenges, we have extended OclPS - orderby and orderDescending - to support single link navigations to attributes used in order. We have introduced a new type of nesting in the ViewModel. This Nesting is a SearchExpression-nesting with a name starting with OrderExpression. If the SearchLogic finds this, we now grab the first active criteria from there and

The MDriven Book

extract the OrderBy/OrderDescending expression-part - this part is appended to the combined resulting expression from the original part of the searchlogic. The video discussing this can be found here: [https://www.youtube.com/watch?v=55K0irODBRE Wednesdays-with-MDriven | View Model Editor 2021: Part 3] {{image:2021-09-27 16h09 061.png|none|thumb|585x585px}}Seek form (web) * Use the tagged value Eco.HiliteGridColumn to signal the user to sort order/searched column that... * Find it by clicking on the orange search-expression and using the "Tagged value" button or the property inspector. The MDriven Book - Next Chapter: Efficient ViewModel fetch

The MDriven Book

Efficient fetch - real case (advanced - skip until you have the need)

The MDriven Book

Introducing MDriven Turnkey

The MDriven Book

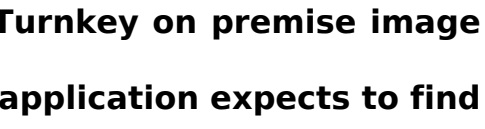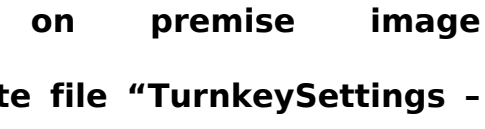## Creating your own MDriven Turnkey instance in your Azure account
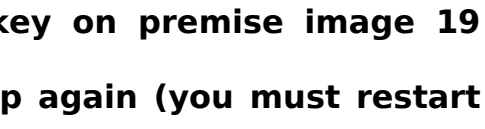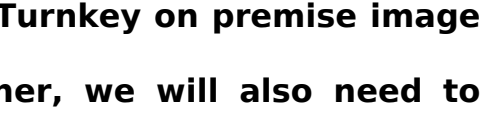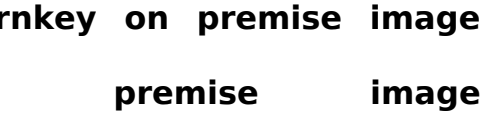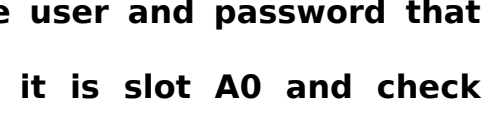
The MDriven Book

Set up MDriven Turnkey on premise

Write the content here to display this box There can be several reasons for running the MDriven Turnkey software on premise rather than in the cloud - and it is easy to achieve. To run MDriven Turnkey on your own IIS, do the following: 1. Create a new IIS application: {{image:Turnkey on premise image 1.png|frameless}} 2. Download these application packages, MDriven Server and MDriven Turnkey: {{image:Turnkey on premise image 2.png|frameless}} 3. Make sure you understand how to make the IIS server use https and how you can either self-sign or acquire your own certificate. This is key for making the internal WCF communication in MDriven Turnkey work. 4. Import the Application Package for MDriven Turnkey into your IIS app: {{image:Turnkey on premise image 3.png|frameless}} {{image:Turnkey on premise image 4.png|frameless}} {{image:Turnkey on premise image 5.png|frameless}} {{image:Turnkey on premise image 6.png|frameless}} Now import the package for MDriven Server into your first application: {{image:Turnkey on premise image 7.png|frameless}} Name the application __MDrivenServer – with two underscores: {{image:Turnkey on premise image 8.png|frameless}} In the end, you will have this: {{image:Turnkey on premise image 9.png|frameless}} Configure for https (not described in this book – IIS-specific knowledge) Notice the __MDrivenServer/logs catalog; things will work better if your application can access this. Find it in File Explorer and give read/write access to "IIS AppPool\NameOfTheAppPool". To configure your system navigate to the __MDrivenServer application: https://localhost/MyMDrivenTurnkeyApp1/__MDrivenServer/ You should see this: {{image:Turnkey on premise image 10.png|frameless}} This is the
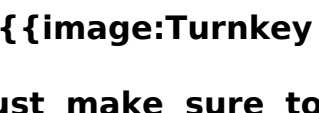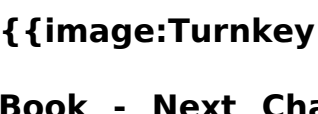
server's response to not finding the expected database. It creates it and sets things up to the best of its ability. Click Index to go back to the default page: {{image:Turnkey on premise image 11.png|frameless}} We are prompted with a login – but we need to register a user that the MDriven Turnkey app will use to access the MDriven Server. You will also want to access the MDriven Server in order to upload and evolve models – so create a user "b" and remember the password: {{image:Turnkey on premise image 12.png|frameless}} Now, log into MDriven Server with this user: {{image:Turnkey on premise image 13.png|frameless}} In the admin area, choose user admin: {{image:Turnkey on premise image 14.png|frameless}} In UserAdmin, you can control the existing users on your MDrivenServer – we need an account to be used by your Turnkey site. To tighten security for our MDrivenServer, we are going to: # Assign the users we want to have access to the role SuperAdmin # Check the box for Admin require Identification # Check the box for Services require identification Check the box for Turn off registration: {{image:Turnkey on premise image 15.png|frameless|306x306px}} This way, we will enforce user identification and turn off the ability to register as a new user. It is important that you remember the password for the users you have – otherwise, you will be locked out. When you have accidentally locked yourself out, you can find the admin database, and open it with SqlServerCompact tools. Find table admin_GlobalSettings and set column TurnOffRegistration to 0 (false). Now, you may register a new user and regain control. {{image:Turnkey on premise image 16.png|frameless}} The database is located in __MDrivenServer\App_Data\DatabaseCompact.sdf Now the MDrivenServer is fully configured and we will look at the MDriven Turnkey application that will

use it. Pointing our browser at the Turnkey application will give us this: https://localhost/MyMDrivenTurnkeyApp1 {{image:Turnkey on premise image 17.png|frameless}} The reason is that the Turnkey application expects to find a file with information about what user and password to use to access __MDrivenServer. The file should be in the App_Data folder and its name must be TurnkeySettings.xml. {{image:Turnkey on premise image 18.png|frameless}} As we see, there is a template file "TurnkeySettings – NotInEffect.xml" – copy this and name it "TurnkeySettings.xml" This file is normally managed from the LicenseAndTicket website – but since we are now installing Turnkey locally, the LicenseAndTicket has no access to help us with this. We need to edit it manually. Open the file and fill in the username and password you picked for MDriven: {{image:Turnkey on premise image 19 .png|frameless}} Once this is done, we try our app again (you must restart your app-pool so that the file is re-read): {{image:Turnkey on premise image 19.png|frameless}} When we use MDriven Designer, we will also need to enter the address to our MDriven Server manually – so skip the first login page and go to the second page: {{image:Turnkey on premise image 20.png|frameless}} {{image:Turnkey on premise image 22.png|frameless|301x301px}} We must fill in the user and password that has access to MDriven Server. We need to say it is slot A0 and check "Automated Deploy" – that means an upload of the model is automatically deployed. The server address is the same one we use to access MDriven Server: https://localhost/MyMDrivenTurnkeyApp1/__MDrivenServer And now we have everything set up to run the development loop or production in-house – nothing on in the cloud. To update the site with fresh binaries for MDriven Turnkey or MDriven server, we can simply reapply fresh packages

from the build server. {{image:Turnkey on premise image 4.png|frameless|333x333px}} {{image:Turnkey on premise image 24.png|frameless|333x333px}} Just make sure to choose "Append files" – because if you "delete", you will lose the settings we just set up and the database with any models. {{image:Turnkey on premise image 25.png|frameless}} The MDriven Book - Next Chapter: MDriven Turnkey Architecture

MDriven Turnkey Architecture

Responsibilities

Data roundtrip

Security

The MDriven Book

Information Security

Write the content here to display this box IT security covers the security of your Information Technology. A natural subset of IT security is information security - secure your information, in this context, as part of an IT system. I often argue that nothing is as easy to sell as fear. Fear of anything. Fear of lacking IT security and Information Security is no different. It is an easy sell. You just open with the line, "Are you sure your data is secure – because I think it might not be?" Deal! Show me! Help me! In my opinion, these are the obvious IT security hygienic must-haves: * Make sure you authenticate users. * Have an order to your authorization processes. * Keep your computers clean from unwanted software. Beyond the obvious IT security must-haves, the delivered "Show me", or "Help me" very seldom come to any real practical effect, other than "You should not have data worth stealing – and if you do, you should not let anyone come near it – not even your staff". Well, thank you, Mr. IT-security expert. Not helpful at all. From my experience in business, government, and military, the latter two take IT security painfully seriously. This does not automatically mean that they are safe, but they spend a lot of effort aiming to be safe. There is a tradeoff between protecting data and making it easy for trusted users to work with data. You must find a level in this tradeoff that reduces risk and does not come in the way of work. There is no such thing as eliminating risk. Trying to eliminate risk will paralyze you and you will get nothing done. Decide what risk level is acceptable – and note that this level may contrast with the different types of data that you have. The risk level may also vary not only on data type and data value but also on data aggregation; you want to protect all the data

more than the individual parts. The Basics of IT Security It is pretty simple really. Think of a PC as a piece of luggage at the airport. "Sir, did you pack this bag yourself? Have you watched it all the time since you closed it? Can you guarantee that there is nothing in here you received from others?" When it comes to your laptop – or the laptop you got from work – you must say "No!" Your computer is not to be trusted. Period. This does not mean that there is anything wrong with it, but we cannot be sure. It is easy to sell this kind of fear. When it comes to the servers for your company that are placed under lock and key – patched and maintained by educated personnel – we might say "Well, we sure hope we did not receive anything we did not want." So your servers might be safe – and they are easier to trust than user-pcs. IT departments will try to toughen your PC up. Again, with the luggage metaphor, they might make it solid, making it impossible to store anything inside, safer – but you are not helped by a solid piece of luggage or a computer you cannot store anything in. Anything in between fully functional and solid is possible, but they can only do so much, and almost anything they do limits the degree of freedom you have with the computer. The IT department is most afraid that your computer contains Trojan software that infects others at work – and increases the threat against the servers. Trojans can act as beach-heads inside your company from which hackers may have a line of sight to your servers. Trojans may also be a nuance or even hold your computer for ransom – but a professional attacking Trojan says nothing; it just steals your data – for weeks or years. Once infected, it is really hard to trust a complex IT environment again because there are so many places where the Trojans may "hide" during cleaning. The cleaning will be extensive and this motivates great precautions to avoid infection. This text, however,

will not deal with that kind of IT security. I just had to state the facts, to make sure we are on the same page. This text is about how we should build systems in this corrosive and hostile environment to control the risk as we expose our data to authorized users. Building Safer Software Systems The thing with building protected software systems is that part of the software will implement the lock that protects it. I will call "the lock" the AccessControlSystem. If the Access Control System protects the data from being exposed to the wrong user – then who protects the AccessControlSystem from being bypassed? This is the heart of the matter. Did someone tell you that your system is protected by the Active Directory in your company? Wrong – the ActiveDirectory may be the one handing out the keys to the lock – but the system itself is responsible for implementing the lock. An Access Control System is almost always part of the system itself (true for all nontrivial access control), just like the lock on your front door is mounted as a part of the door. So what is the best way to protect the lock? Let the lock stay on the server! This is the conclusion that most software architects have reached. Since the lock is a part of the system – the system must stay on the server as well. This is the main reason for moving away from rich fat clients on Windows to a typically worse user experience with web technology like MVC even for in-house systems. Another way to keep the system on the server is to use terminal server solutions. The important thing is that the Access Control System filters out the slice of data that is acceptable to show an individually authenticated user. Filtering out means that you have a bigger volume to filter from – a volume to reduce to a subset. The larger volume is what the Access Control System protects. If the Access Control System is going to protect this volume and filter ok data from it, then

it must have access to the whole. This is simple logic. Having stated these facts, we have a clear view of our goal: to deliver approved slices of data to authenticated users – our users are not physically on the servers – they are on their laptops. Although the MVC web approach does this, many have found the reduced user experience is not ideal for prolonged use in an office. Bulky postbacks and the tendency to reduce the amount of data on a single screen to avoid lags often make web-based systems click-intensive and filled with stressful waiting. To mitigate this, many software architects have started to use Ajax and Javascript to build a richer user experience to replace the plain data presentation offered by MVC. When doing so, they might start to hold data locally in the browser, and gradually, they may start to lose track of our simple goal – maybe implementing filtering in the browser – maybe keeping data between screen navigations – maybe doing some work that was already part of the Access Control System. No one is certain, because it is difficult to see what a software system does without high-effort reviewing of actual code. How MDriven Turnkey Does It – Every Time To avoid the risk of muddling the definition of the Access Control System while still offering a rich user experience, MDriven Turnkey does 4 things: # Lets the ViewModel define the slice of data we agree with showing an authenticated user with authorization for a specific use case. The user cannot see more by definition in the framework. # Lets the ViewModel reduction of data from the complete model happen on the server and since each ViewModel is fully implemented with the declarative Object Constraints Language (OCL), you also have an exact and easily understood definition of the information subset it contains. # Defines the rules that build up the Access Control Systems in the model declaratively – with static verification – and the ability to make the rules

depend on any data in your model. # Streams the resulting slice of data – and any changes to it – to a client to produce a rich client experience without slow roundtrips and exposes nothing else. MDriven Turnkey comes with the Access Control System already in place – on the server – you merely need to define your keys depending on your rules based on your model. Even if you build a WPF, AngularJS, or Android client – the Access Control System is on the server adhering to your rules. Since the platform has the Access Control System in-built, developers can make fewer mistakes. The verification of the rules can now be done statically in the model and we do not need to review code to detect the tendencies of muddling the simple goal of ensuring information filtering is happening on the server only. MDriven Turnkey user authentication is done with standard OAuth2 allowing for implementations with optional social login or Single sign-on (SSO) with OpenId. Multifactor authentication can easily be made part of your Access Control System. Server-to-server authentication using OAuth2 is also supported as shown here. All communication is done over SSL and if you have stronger crypto needs, any tunnel can be used to protect traffic between the user and server. We claim two important things: * Building software systems with MDriven Turnkey will make them intrinsically safe – meaning that even the lowest effort will expose only what you intend to expose. * Using MDriven Turnkey will make your complete Access Control System possible to inspect even for non-coders – meaning that security officers can understand what risk level individual screens pose even in an evolving system. Both these properties are highly sought after by organizations striving to minimize the risk of unwanted information leakage – whether it is from coding mistakes, architectural mistakes, misinterpreted requirements, or simply plain laziness of either

developers or reviewers. The MDriven Book - Next Chapter: Access control system in MDriven {{Edited|July|12|2024}}

The MDriven Book

The basics of IT security

Write the content here to display this box IT security covers the security of your Information Technology. A natural subset of IT security is information security - secure your information, in this context, as part of an IT system. I often argue that nothing is as easy to sell as fear. Fear of anything. Fear of lacking IT security and Information Security is no different. It is an easy sell. You just open with the line, "Are you sure your data is secure – because I think it might not be?" Deal! Show me! Help me! In my opinion, these are the obvious IT security hygienic must-haves: * Make sure you authenticate users. * Have an order to your authorization processes. * Keep your computers clean from unwanted software. Beyond the obvious IT security must-haves, the delivered "Show me", or "Help me" very seldom come to any real practical effect, other than "You should not have data worth stealing – and if you do, you should not let anyone come near it – not even your staff". Well, thank you, Mr. IT-security expert. Not helpful at all. From my experience in business, government, and military, the latter two take IT security painfully seriously. This does not automatically mean that they are safe, but they spend a lot of effort aiming to be safe. There is a tradeoff between protecting data and making it easy for trusted users to work with data. You must find a level in this tradeoff that reduces risk and does not come in the way of work. There is no such thing as eliminating risk. Trying to eliminate risk will paralyze you and you will get nothing done. Decide what risk level is acceptable – and note that this level may contrast with the different types of data that you have. The risk level may also vary not only on data type and data value but also on data aggregation; you want to protect all the data

more than the individual parts. The Basics of IT Security It is pretty simple really. Think of a PC as a piece of luggage at the airport. "Sir, did you pack this bag yourself? Have you watched it all the time since you closed it? Can you guarantee that there is nothing in here you received from others?" When it comes to your laptop – or the laptop you got from work – you must say "No!" Your computer is not to be trusted. Period. This does not mean that there is anything wrong with it, but we cannot be sure. It is easy to sell this kind of fear. When it comes to the servers for your company that are placed under lock and key – patched and maintained by educated personnel – we might say "Well, we sure hope we did not receive anything we did not want." So your servers might be safe – and they are easier to trust than user-pcs. IT departments will try to toughen your PC up. Again, with the luggage metaphor, they might make it solid, making it impossible to store anything inside, safer – but you are not helped by a solid piece of luggage or a computer you cannot store anything in. Anything in between fully functional and solid is possible, but they can only do so much, and almost anything they do limits the degree of freedom you have with the computer. The IT department is most afraid that your computer contains Trojan software that infects others at work – and increases the threat against the servers. Trojans can act as beach-heads inside your company from which hackers may have a line of sight to your servers. Trojans may also be a nuance or even hold your computer for ransom – but a professional attacking Trojan says nothing; it just steals your data – for weeks or years. Once infected, it is really hard to trust a complex IT environment again because there are so many places where the Trojans may "hide" during cleaning. The cleaning will be extensive and this motivates great precautions to avoid infection. This text, however,

will not deal with that kind of IT security. I just had to state the facts, to make sure we are on the same page. This text is about how we should build systems in this corrosive and hostile environment to control the risk as we expose our data to authorized users. Building Safer Software Systems The thing with building protected software systems is that part of the software will implement the lock that protects it. I will call "the lock" the AccessControlSystem. If the Access Control System protects the data from being exposed to the wrong user – then who protects the AccessControlSystem from being bypassed? This is the heart of the matter. Did someone tell you that your system is protected by the Active Directory in your company? Wrong – the ActiveDirectory may be the one handing out the keys to the lock – but the system itself is responsible for implementing the lock. An Access Control System is almost always part of the system itself (true for all nontrivial access control), just like the lock on your front door is mounted as a part of the door. So what is the best way to protect the lock? Let the lock stay on the server! This is the conclusion that most software architects have reached. Since the lock is a part of the system – the system must stay on the server as well. This is the main reason for moving away from rich fat clients on Windows to a typically worse user experience with web technology like MVC even for in-house systems. Another way to keep the system on the server is to use terminal server solutions. The important thing is that the Access Control System filters out the slice of data that is acceptable to show an individually authenticated user. Filtering out means that you have a bigger volume to filter from – a volume to reduce to a subset. The larger volume is what the Access Control System protects. If the Access Control System is going to protect this volume and filter ok data from it, then

it must have access to the whole. This is simple logic. Having stated these facts, we have a clear view of our goal: to deliver approved slices of data to authenticated users – our users are not physically on the servers – they are on their laptops. Although the MVC web approach does this, many have found the reduced user experience is not ideal for prolonged use in an office. Bulky postbacks and the tendency to reduce the amount of data on a single screen to avoid lags often make web-based systems click-intensive and filled with stressful waiting. To mitigate this, many software architects have started to use Ajax and Javascript to build a richer user experience to replace the plain data presentation offered by MVC. When doing so, they might start to hold data locally in the browser, and gradually, they may start to lose track of our simple goal – maybe implementing filtering in the browser – maybe keeping data between screen navigations – maybe doing some work that was already part of the Access Control System. No one is certain, because it is difficult to see what a software system does without high-effort reviewing of actual code. How MDriven Turnkey Does It – Every Time To avoid the risk of muddling the definition of the Access Control System while still offering a rich user experience, MDriven Turnkey does 4 things: # Lets the ViewModel define the slice of data we agree with showing an authenticated user with authorization for a specific use case. The user cannot see more by definition in the framework. # Lets the ViewModel reduction of data from the complete model happen on the server and since each ViewModel is fully implemented with the declarative Object Constraints Language (OCL), you also have an exact and easily understood definition of the information subset it contains. # Defines the rules that build up the Access Control Systems in the model declaratively – with static verification – and the ability to make the rules

depend on any data in your model. # Streams the resulting slice of data – and any changes to it – to a client to produce a rich client experience without slow roundtrips and exposes nothing else. MDriven Turnkey comes with the Access Control System already in place – on the server – you merely need to define your keys depending on your rules based on your model. Even if you build a WPF, AngularJS, or Android client – the Access Control System is on the server adhering to your rules. Since the platform has the Access Control System in-built, developers can make fewer mistakes. The verification of the rules can now be done statically in the model and we do not need to review code to detect the tendencies of muddling the simple goal of ensuring information filtering is happening on the server only. MDriven Turnkey user authentication is done with standard OAuth2 allowing for implementations with optional social login or Single sign-on (SSO) with OpenId. Multifactor authentication can easily be made part of your Access Control System. Server-to-server authentication using OAuth2 is also supported as shown here. All communication is done over SSL and if you have stronger crypto needs, any tunnel can be used to protect traffic between the user and server. We claim two important things: * Building software systems with MDriven Turnkey will make them intrinsically safe – meaning that even the lowest effort will expose only what you intend to expose. * Using MDriven Turnkey will make your complete Access Control System possible to inspect even for non-coders – meaning that security officers can understand what risk level individual screens pose even in an evolving system. Both these properties are highly sought after by organizations striving to minimize the risk of unwanted information leakage – whether it is from coding mistakes, architectural mistakes, misinterpreted requirements, or simply plain laziness of either

developers or reviewers. The MDriven Book - Next Chapter: Access control system in MDriven {{Edited|July|12|2024}}

The MDriven Book

Building safer software systems

Write the content here to display this box IT security covers the security of your Information Technology. A natural subset of IT security is information security - secure your information, in this context, as part of an IT system. I often argue that nothing is as easy to sell as fear. Fear of anything. Fear of lacking IT security and Information Security is no different. It is an easy sell. You just open with the line, "Are you sure your data is secure – because I think it might not be?" Deal! Show me! Help me! In my opinion, these are the obvious IT security hygienic must-haves: * Make sure you authenticate users. * Have an order to your authorization processes. * Keep your computers clean from unwanted software. Beyond the obvious IT security must-haves, the delivered "Show me", or "Help me" very seldom come to any real practical effect, other than "You should not have data worth stealing – and if you do, you should not let anyone come near it – not even your staff". Well, thank you, Mr. IT-security expert. Not helpful at all. From my experience in business, government, and military, the latter two take IT security painfully seriously. This does not automatically mean that they are safe, but they spend a lot of effort aiming to be safe. There is a tradeoff between protecting data and making it easy for trusted users to work with data. You must find a level in this tradeoff that reduces risk and does not come in the way of work. There is no such thing as eliminating risk. Trying to eliminate risk will paralyze you and you will get nothing done. Decide what risk level is acceptable – and note that this level may contrast with the different types of data that you have. The risk level may also vary not only on data type and data value but also on data aggregation; you want to protect all the data

more than the individual parts. The Basics of IT Security It is pretty simple really. Think of a PC as a piece of luggage at the airport. "Sir, did you pack this bag yourself? Have you watched it all the time since you closed it? Can you guarantee that there is nothing in here you received from others?" When it comes to your laptop – or the laptop you got from work – you must say "No!" Your computer is not to be trusted. Period. This does not mean that there is anything wrong with it, but we cannot be sure. It is easy to sell this kind of fear. When it comes to the servers for your company that are placed under lock and key – patched and maintained by educated personnel – we might say "Well, we sure hope we did not receive anything we did not want." So your servers might be safe – and they are easier to trust than user-pcs. IT departments will try to toughen your PC up. Again, with the luggage metaphor, they might make it solid, making it impossible to store anything inside, safer – but you are not helped by a solid piece of luggage or a computer you cannot store anything in. Anything in between fully functional and solid is possible, but they can only do so much, and almost anything they do limits the degree of freedom you have with the computer. The IT department is most afraid that your computer contains Trojan software that infects others at work – and increases the threat against the servers. Trojans can act as beach-heads inside your company from which hackers may have a line of sight to your servers. Trojans may also be a nuance or even hold your computer for ransom – but a professional attacking Trojan says nothing; it just steals your data – for weeks or years. Once infected, it is really hard to trust a complex IT environment again because there are so many places where the Trojans may "hide" during cleaning. The cleaning will be extensive and this motivates great precautions to avoid infection. This text, however,

will not deal with that kind of IT security. I just had to state the facts, to make sure we are on the same page. This text is about how we should build systems in this corrosive and hostile environment to control the risk as we expose our data to authorized users. Building Safer Software Systems The thing with building protected software systems is that part of the software will implement the lock that protects it. I will call "the lock" the AccessControlSystem. If the Access Control System protects the data from being exposed to the wrong user – then who protects the AccessControlSystem from being bypassed? This is the heart of the matter. Did someone tell you that your system is protected by the Active Directory in your company? Wrong – the ActiveDirectory may be the one handing out the keys to the lock – but the system itself is responsible for implementing the lock. An Access Control System is almost always part of the system itself (true for all nontrivial access control), just like the lock on your front door is mounted as a part of the door. So what is the best way to protect the lock? Let the lock stay on the server! This is the conclusion that most software architects have reached. Since the lock is a part of the system – the system must stay on the server as well. This is the main reason for moving away from rich fat clients on Windows to a typically worse user experience with web technology like MVC even for in-house systems. Another way to keep the system on the server is to use terminal server solutions. The important thing is that the Access Control System filters out the slice of data that is acceptable to show an individually authenticated user. Filtering out means that you have a bigger volume to filter from – a volume to reduce to a subset. The larger volume is what the Access Control System protects. If the Access Control System is going to protect this volume and filter ok data from it, then

it must have access to the whole. This is simple logic. Having stated these facts, we have a clear view of our goal: to deliver approved slices of data to authenticated users – our users are not physically on the servers – they are on their laptops. Although the MVC web approach does this, many have found the reduced user experience is not ideal for prolonged use in an office. Bulky postbacks and the tendency to reduce the amount of data on a single screen to avoid lags often make web-based systems click-intensive and filled with stressful waiting. To mitigate this, many software architects have started to use Ajax and Javascript to build a richer user experience to replace the plain data presentation offered by MVC. When doing so, they might start to hold data locally in the browser, and gradually, they may start to lose track of our simple goal – maybe implementing filtering in the browser – maybe keeping data between screen navigations – maybe doing some work that was already part of the Access Control System. No one is certain, because it is difficult to see what a software system does without high-effort reviewing of actual code. How MDriven Turnkey Does It – Every Time To avoid the risk of muddling the definition of the Access Control System while still offering a rich user experience, MDriven Turnkey does 4 things: # Lets the ViewModel define the slice of data we agree with showing an authenticated user with authorization for a specific use case. The user cannot see more by definition in the framework. # Lets the ViewModel reduction of data from the complete model happen on the server and since each ViewModel is fully implemented with the declarative Object Constraints Language (OCL), you also have an exact and easily understood definition of the information subset it contains. # Defines the rules that build up the Access Control Systems in the model declaratively – with static verification – and the ability to make the rules

depend on any data in your model. # Streams the resulting slice of data – and any changes to it – to a client to produce a rich client experience without slow roundtrips and exposes nothing else. MDriven Turnkey comes with the Access Control System already in place – on the server – you merely need to define your keys depending on your rules based on your model. Even if you build a WPF, AngularJS, or Android client – the Access Control System is on the server adhering to your rules. Since the platform has the Access Control System in-built, developers can make fewer mistakes. The verification of the rules can now be done statically in the model and we do not need to review code to detect the tendencies of muddling the simple goal of ensuring information filtering is happening on the server only. MDriven Turnkey user authentication is done with standard OAuth2 allowing for implementations with optional social login or Single sign-on (SSO) with OpenId. Multifactor authentication can easily be made part of your Access Control System. Server-to-server authentication using OAuth2 is also supported as shown here. All communication is done over SSL and if you have stronger crypto needs, any tunnel can be used to protect traffic between the user and server. We claim two important things: * Building software systems with MDriven Turnkey will make them intrinsically safe – meaning that even the lowest effort will expose only what you intend to expose. * Using MDriven Turnkey will make your complete Access Control System possible to inspect even for non-coders – meaning that security officers can understand what risk level individual screens pose even in an evolving system. Both these properties are highly sought after by organizations striving to minimize the risk of unwanted information leakage – whether it is from coding mistakes, architectural mistakes, misinterpreted requirements, or simply plain laziness of either

developers or reviewers. The MDriven Book - Next Chapter: Access control system in MDriven {{Edited|July|12|2024}}

The MDriven Book

How MDriven Turnkey does it - every time

Write the content here to display this box IT security covers the security of your Information Technology. A natural subset of IT security is information security - secure your information, in this context, as part of an IT system. I often argue that nothing is as easy to sell as fear. Fear of anything. Fear of lacking IT security and Information Security is no different. It is an easy sell. You just open with the line, "Are you sure your data is secure – because I think it might not be?" Deal! Show me! Help me! In my opinion, these are the obvious IT security hygienic must-haves: * Make sure you authenticate users. * Have an order to your authorization processes. * Keep your computers clean from unwanted software. Beyond the obvious IT security must-haves, the delivered "Show me", or "Help me" very seldom come to any real practical effect, other than "You should not have data worth stealing – and if you do, you should not let anyone come near it – not even your staff". Well, thank you, Mr. IT-security expert. Not helpful at all. From my experience in business, government, and military, the latter two take IT security painfully seriously. This does not automatically mean that they are safe, but they spend a lot of effort aiming to be safe. There is a tradeoff between protecting data and making it easy for trusted users to work with data. You must find a level in this tradeoff that reduces risk and does not come in the way of work. There is no such thing as eliminating risk. Trying to eliminate risk will paralyze you and you will get nothing done. Decide what risk level is acceptable – and note that this level may contrast with the different types of data that you have. The risk level may also vary not only on data type and data value but also on data aggregation; you want to protect all the data

more than the individual parts. The Basics of IT Security It is pretty simple really. Think of a PC as a piece of luggage at the airport. "Sir, did you pack this bag yourself? Have you watched it all the time since you closed it? Can you guarantee that there is nothing in here you received from others?" When it comes to your laptop – or the laptop you got from work – you must say "No!" Your computer is not to be trusted. Period. This does not mean that there is anything wrong with it, but we cannot be sure. It is easy to sell this kind of fear. When it comes to the servers for your company that are placed under lock and key – patched and maintained by educated personnel – we might say "Well, we sure hope we did not receive anything we did not want." So your servers might be safe – and they are easier to trust than user-pcs. IT departments will try to toughen your PC up. Again, with the luggage metaphor, they might make it solid, making it impossible to store anything inside, safer – but you are not helped by a solid piece of luggage or a computer you cannot store anything in. Anything in between fully functional and solid is possible, but they can only do so much, and almost anything they do limits the degree of freedom you have with the computer. The IT department is most afraid that your computer contains Trojan software that infects others at work – and increases the threat against the servers. Trojans can act as beach-heads inside your company from which hackers may have a line of sight to your servers. Trojans may also be a nuance or even hold your computer for ransom – but a professional attacking Trojan says nothing; it just steals your data – for weeks or years. Once infected, it is really hard to trust a complex IT environment again because there are so many places where the Trojans may "hide" during cleaning. The cleaning will be extensive and this motivates great precautions to avoid infection. This text, however,

will not deal with that kind of IT security. I just had to state the facts, to make sure we are on the same page. This text is about how we should build systems in this corrosive and hostile environment to control the risk as we expose our data to authorized users. Building Safer Software Systems The thing with building protected software systems is that part of the software will implement the lock that protects it. I will call "the lock" the AccessControlSystem. If the Access Control System protects the data from being exposed to the wrong user – then who protects the AccessControlSystem from being bypassed? This is the heart of the matter. Did someone tell you that your system is protected by the Active Directory in your company? Wrong – the ActiveDirectory may be the one handing out the keys to the lock – but the system itself is responsible for implementing the lock. An Access Control System is almost always part of the system itself (true for all nontrivial access control), just like the lock on your front door is mounted as a part of the door. So what is the best way to protect the lock? Let the lock stay on the server! This is the conclusion that most software architects have reached. Since the lock is a part of the system – the system must stay on the server as well. This is the main reason for moving away from rich fat clients on Windows to a typically worse user experience with web technology like MVC even for in-house systems. Another way to keep the system on the server is to use terminal server solutions. The important thing is that the Access Control System filters out the slice of data that is acceptable to show an individually authenticated user. Filtering out means that you have a bigger volume to filter from – a volume to reduce to a subset. The larger volume is what the Access Control System protects. If the Access Control System is going to protect this volume and filter ok data from it, then

it must have access to the whole. This is simple logic. Having stated these facts, we have a clear view of our goal: to deliver approved slices of data to authenticated users – our users are not physically on the servers – they are on their laptops. Although the MVC web approach does this, many have found the reduced user experience is not ideal for prolonged use in an office. Bulky postbacks and the tendency to reduce the amount of data on a single screen to avoid lags often make web-based systems click-intensive and filled with stressful waiting. To mitigate this, many software architects have started to use Ajax and Javascript to build a richer user experience to replace the plain data presentation offered by MVC. When doing so, they might start to hold data locally in the browser, and gradually, they may start to lose track of our simple goal – maybe implementing filtering in the browser – maybe keeping data between screen navigations – maybe doing some work that was already part of the Access Control System. No one is certain, because it is difficult to see what a software system does without high-effort reviewing of actual code. How MDriven Turnkey Does It – Every Time To avoid the risk of muddling the definition of the Access Control System while still offering a rich user experience, MDriven Turnkey does 4 things: # Lets the ViewModel define the slice of data we agree with showing an authenticated user with authorization for a specific use case. The user cannot see more by definition in the framework. # Lets the ViewModel reduction of data from the complete model happen on the server and since each ViewModel is fully implemented with the declarative Object Constraints Language (OCL), you also have an exact and easily understood definition of the information subset it contains. # Defines the rules that build up the Access Control Systems in the model declaratively – with static verification – and the ability to make the rules

depend on any data in your model. # Streams the resulting slice of data – and any changes to it – to a client to produce a rich client experience without slow roundtrips and exposes nothing else. MDriven Turnkey comes with the Access Control System already in place – on the server – you merely need to define your keys depending on your rules based on your model. Even if you build a WPF, AngularJS, or Android client – the Access Control System is on the server adhering to your rules. Since the platform has the Access Control System in-built, developers can make fewer mistakes. The verification of the rules can now be done statically in the model and we do not need to review code to detect the tendencies of muddling the simple goal of ensuring information filtering is happening on the server only. MDriven Turnkey user authentication is done with standard OAuth2 allowing for implementations with optional social login or Single sign-on (SSO) with OpenId. Multifactor authentication can easily be made part of your Access Control System. Server-to-server authentication using OAuth2 is also supported as shown here. All communication is done over SSL and if you have stronger crypto needs, any tunnel can be used to protect traffic between the user and server. We claim two important things: * Building software systems with MDriven Turnkey will make them intrinsically safe – meaning that even the lowest effort will expose only what you intend to expose. * Using MDriven Turnkey will make your complete Access Control System possible to inspect even for non-coders – meaning that security officers can understand what risk level individual screens pose even in an evolving system. Both these properties are highly sought after by organizations striving to minimize the risk of unwanted information leakage – whether it is from coding mistakes, architectural mistakes, misinterpreted requirements, or simply plain laziness of either

developers or reviewers. The MDriven Book - Next Chapter: Access control system in MDriven {{Edited|July|12|2024}}

The MDriven Book

How the access control system is constructed in MDriven

Write the content here to display this box First, we must agree that the reasons for restricting access can be manifold. This may be as simple as avoiding confusing a user with too many options at the wrong time in the process – or it may be as crucial as protecting highly sensitive information from getting into the wrong hands. Any practical Access control is probably based on static information as "users belonging to group Fishermen should not have access to view Treasury", but also dynamic information as "if a Fisherman has been granted payment but has not collected it yet, he should be able to open the view Treasury". Another aspect is that we sometimes want to show that actions are available – but not enabled to you at this particular moment. At other times, we do not even want to show you that a given action exists in the system. These are all common requirements when dealing with access control. Since MDriven solves all user interaction through ViewModels and Actions, it is natural that these are our targets for Access Control. We can set any expression in the Enable expression of an action – this is one way to disable actions. If we have a rule that is the same for many actions, we may consolidate the rule to an AccessGroup: {{image:Access control system in MDriven 01.png|frameless|365x365px}} {{image:Accessgroups.png|frameless|429x429px}} AccessGroups are useful for static rules like checking if the logged-in user is part of the fisherman group:

SysSingleton.oclSingleton.CurrentUser.SysRoles->exists(name='fisherman')

For actions, we can set the Enable expression for more precision: {{image:Access control system in MDriven.png|frameless|426x426px}} We

may also want to give the User a hint as to why the action is disabled. This ability has reduced the support requests a lot in all the projects we have done: {{image:Access control system in MDriven 03.png|frameless|413x413px}} For views, refine the access control per placed widget by setting the ReadOnly expression or/and the Visible expression: {{image:Access system in MDriven 04.png|frameless|467x467px}} But in views, you can also work on the whole view: {{image:Access system in MDriven 05.png|frameless|460x460px}} Setting the Readonly expression on this level affects all the widgets in the view. Setting the access expression will control if the view is shown at all. If the view's Access expression evaluates to false, the system will look for a ViewModel named AccessDenied and if found, will show this instead. If no AccessDenied view is found, then a blank screen will show. To allow you to disable an action that would bring up a view with an access expression evaluating to false given a certain root object, check the result of the Access expression before bringing up the view. Do this with the OCL operator canAccess(vmname):bool. This enables you to disable actions that will end up showing AccessDenied and stop reports from being created based on a ViewModel and root object that is not allowed based on the Access expression of the ViewModel. Working with the described levels of expression, you can tailor a perfect-fit access control system that evaluates in the safe realm of your server. All OCL rules use a few OCL operators but mostly names from your model that are probably in a ubiquitous language shared with the security officers of your domain. The transparency and fine-grained control of this access control system is precisely what many organizations need to protect their information and still allow for fast-paced development. NOTE:

The MDriven Book

This is the last chapter of The MDriven Book. See also: AccessGroups,_InterestGroups_and_ViewModel-Enable, Access_groups

The MDriven Book

Table of Contents

The MDriven Book

The MDriven Book