CATIA Magic

Custom Reporting: Report Patterns

Daniel Brookshier
Daniel.Brookshier@3ds.com
linkedin.com/in/danielbrookshier/

INCOSE North Texas, October 2022

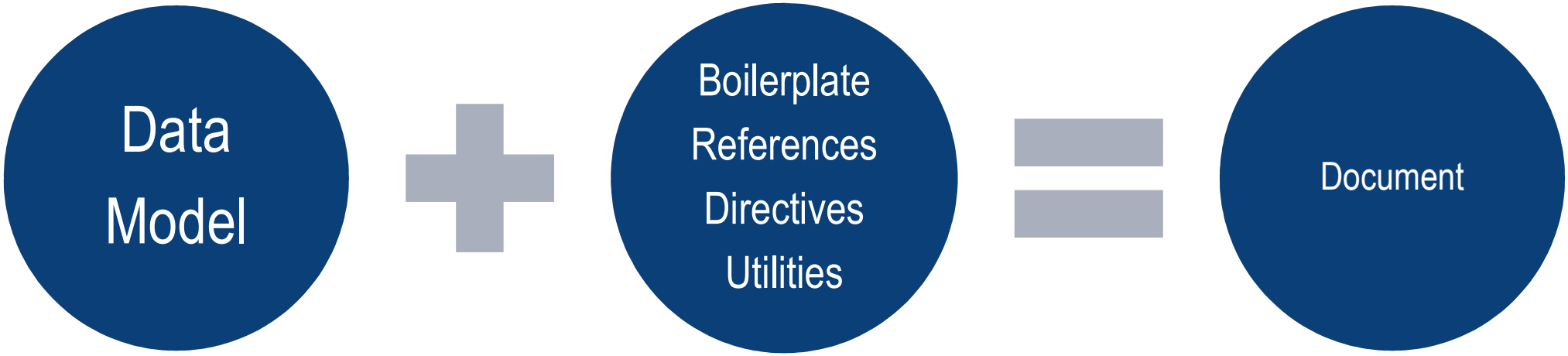
No Magic

DASSAULT SYSTEMES | The 3DEXPERIENCE® Company

Overview

- Quick overview of reports
- About Patterns
- Pattern: Filter-based Section
- Pattern: Tables to Select Report Content
- Pattern: Report Wizard Selected Elements
- Pattern: Smart Package for Report Content Data
- Pattern: Report Model
- Pattern: Use Tables to Drive Reports
- Pattern: Flatten Tables
- Pattern: Reading Writing JSON (Code or Data Generation)
- Anti-pattern: Monolithic Template
- Anti-pattern: Too Many Macros
- Anti-pattern: Hard coded references
- Anti-pattern: Expected data

Velocity merges data from a data model to a template to produce the document.



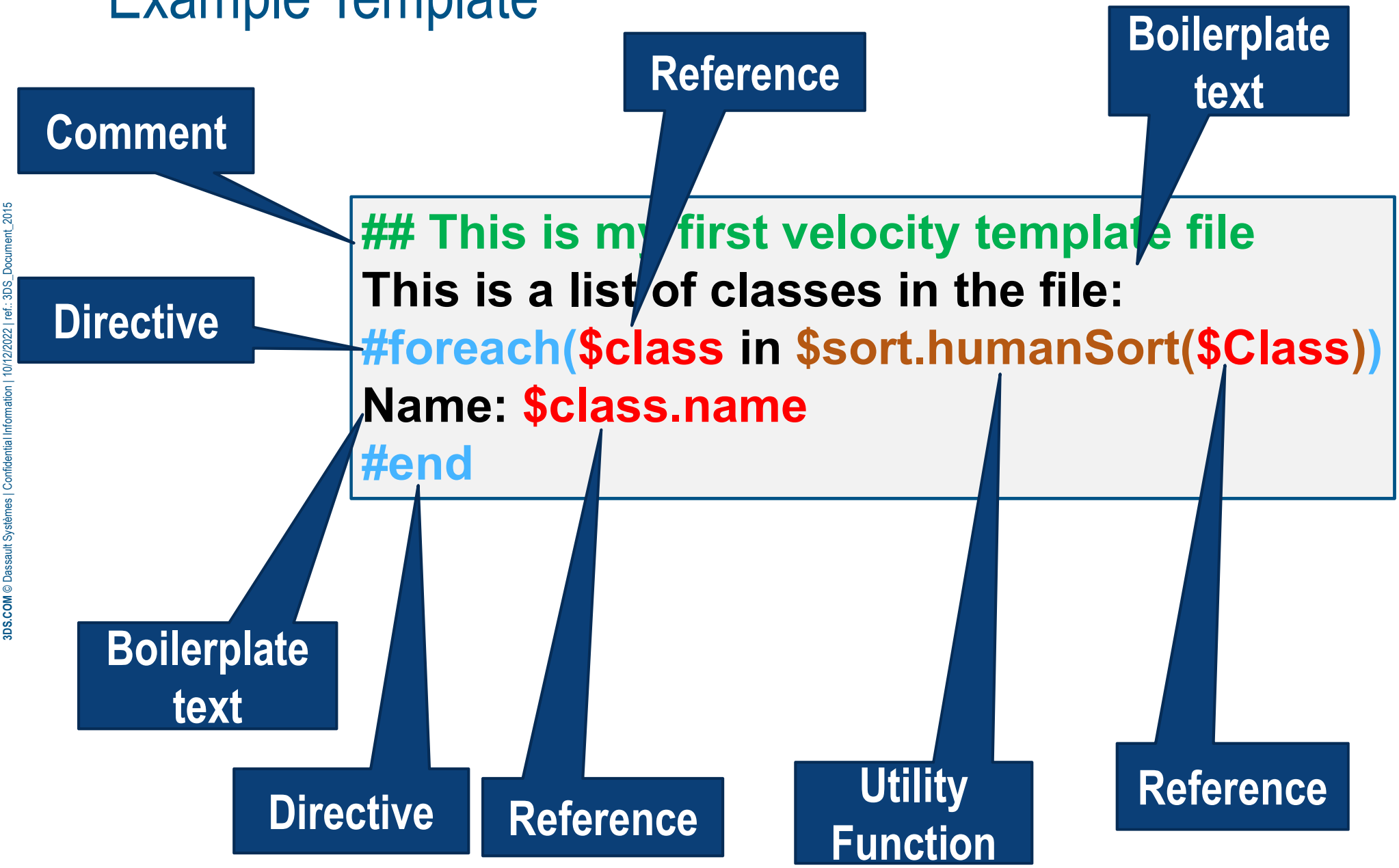
Velocity Templates and Report Wizard

- Velocity Template Language (VTL)
  - Established template language for Java (CATIA Magic/Cameo/MagicDraw are Java apps)
  - Extensible
  - Very fast
  - Works even when data is incomplete
- Issues with VTL
  - VTL was written for text templates (HTML/email)
  - Not designed for reporting on complex data models like UML/SysML
  - Not intended for complex processing (this is why it is so fast)
  - Interpreted (all errors are runtime)
  - Works even when data is incomplete

Example Template

```
## This is my first velocity template file
This is a list of classes in the file:
#foreach($class in $sort.humanSort($Class))
Name: $class.name
#end
```

Example Template

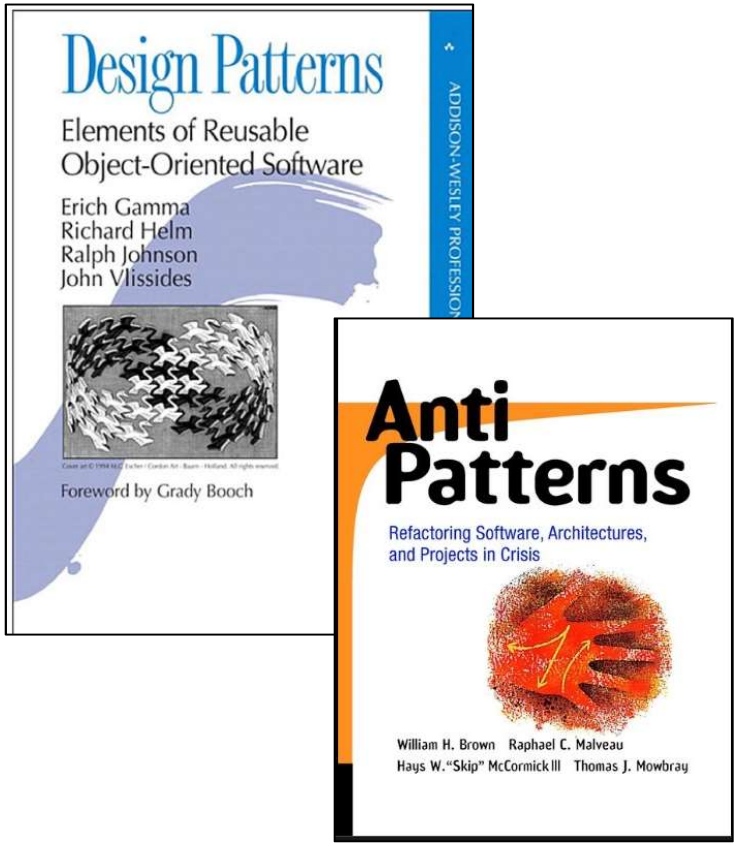


No Magic

DASSAULT SYSTEMES | The 3DEXPERIENCE® Company

About Patterns

- Design Patterns are a common method to match problems with solutions
- Not necessarily a reusable code because solutions are often specific to application
- Knowing pattern reduce the effort of designing the solution
- Anti-patterns are a method of detecting problems and suggesting a solution (refactoring)
- Combining patterns and anti-patterns avoids re-inventing solutions
- Caution: Not all patterns are good for your needs or simple to implement.



Why Patterns in Reporting

- Models change over time
- Templates are a view of the changing model
- Rare that two companies or two stakeholders have the same model structure or formatting of template results.
- Templates and source models may become standardized, **but not today**.
- Translation:

All templates are just an application of a pattern to create a current view of the model with current stakeholder viewpoint.

- Understand Patterns
  - Match pattern to needs
  - Understand variations
  - Manage templates over time
  - Use simplified templates with example patterns as a starting point for production templates
- Understand Anti-patterns
  - When are you in trouble?
  - Why do templates break?
  - Match ant-pattern to patterns to recover.

The Big Issues:

- Reports are hard
- Critics are harder
- Responding to critics is even harder

3DEXPERIENCE®

Pattern: Filter-based Section

Print Properties By Kind

Pattern: Filter-based Section

- Problem:
  - Many metatypes of UML, SysML, and other profiles are similar
  - Need to separate by metatype because they represent very different concepts
- Examples:
  - Attributes
  - Ports
  - SysML:
    - Ports: standard, full, and proxy
    - Constraint Properties
    - SysML Value Properties
- Solution: Print filters of attributes by metatype name.
  - Macro to Print type by name
    - Name is used to filter: 'Value Property', 'Full Port Property',etc.
    - The section is printed
    - The section contents are then printed
  - Macro to print list of named types (Could also use array of strings)
    - Used to designate the name and order or the sections.
    - The example calls the display macro

Filter Tool

- The filter tool reduces a collection based on a criteria
- All Classes (Class, Block, Interface Block, Performer, etc.) have properties (ownedProperty), but these properties are grouped in compartments.
- For example, a Class has:
  - Properties
  - Ports
- A Block has more:
  - Value properties
  - Reference properties
  - Constraint properties
  - Full Ports
  - Proxy Ports
  - Flow properties
- How do we print these in a template?

Filter Tool API

- $report.filterDiagram(diagramList, diagramTypes)
  - Filter diagrams
  - $report.filterDiagram($Diagram, 'Block Diagram')
- $report.filterElement(elementList, humanTypes)
  - Filter elements by human type name (same as user interface)
  - $report.filterElement($list, 'Interface Block')
- $report.filterElementType(elementList, elementType)
  - Filter elements by official type name
- $report.filter(elementList, propertyName, propertyValue)
  - Filter properties by type

Macro: propertyCompartment

```
#macro(propertyCompartment $compartmentName)
#set($sectionData =
$report.filterElement($block.ownedAttribute,
[$compartmentName]))
#if(!$sectionData.isEmpty())
  $compartmentName:
#foreach($pp in $sectionData)
#if($pp.isConjugated)~#end$pp.name: $pp.type.name
#end
#end
#end##macro
```

Macro: printBlockProperties

```
#macro(printBlockProperties $block)
#if($block.humanType=='Block'||$block.humanType=='Interfac
e Block')
```

$block.humanType: $block.name

```
#propertyCompartment('Part Property')
#propertyCompartment('Reference Property')
#propertyCompartment('Value Property')
#propertyCompartment('Constraint Property')
#propertyCompartment('Flow Property')
#propertyCompartment('Full Port')
#propertyCompartment('Proxy Port')
#propertyCompartment('Flow Port')
#propertyCompartment('Port')
#end
#end
```


No Magic

DASSAULT SYSTEMES | The 3DEXPERIENCE® Company

Other Benefits

- Can be used to turn on/off various levels of content
- Easier to test.
- Centralizes formatting of similar types

Other Variants

- Variation: Package Filter
  - Any element, package, or type can be used in a similar way.
  - For example: A list of named packages
- Variation: Entry #IF filter
  - Use an if statement to detect type to enable or disable content.
  - Calling template calls macro without knowing if macro will print
  - Can be used to call a macro for content we could print without checking in main body of the template
  - Helps with order of content without adding type and content checks to main body.
  - On the surface, very wasteful, but cleans up template significantly
  - Compartmentalization of type, sorting, and checks for missing data.

Variant

```
#macro(printBlockProperties $block $sectionNames)
#if($block.humanType=='Block'||$block.humanType==
'Interface Block')
```

$block.humanType: $block.name

#foreach($sec in $sectionNames)

```
#propertyCompartment($sec)
#end
#end
#end
```


No Magic

Variant and Simple Use

```
## Set the value names array in the order of sections
#set($sections = ['Part Property', 'Reference Property',
'Value Property', 'Constraint Property', 'Flow Property',
'Full Port', 'Proxy Port', 'Flow Port', 'Port'])
#foreach($block in $Block)
#printBlockProperties($block,$sectionNames)
#end
#foreach($block in $Block)
Section: Blocks
#printBlockProperties($block,$sectionNames)
#end
#foreach($block in $InterfaceBlock)
Section: Interface Blocks
#printBlockProperties($block,$sectionNames)
#end
```

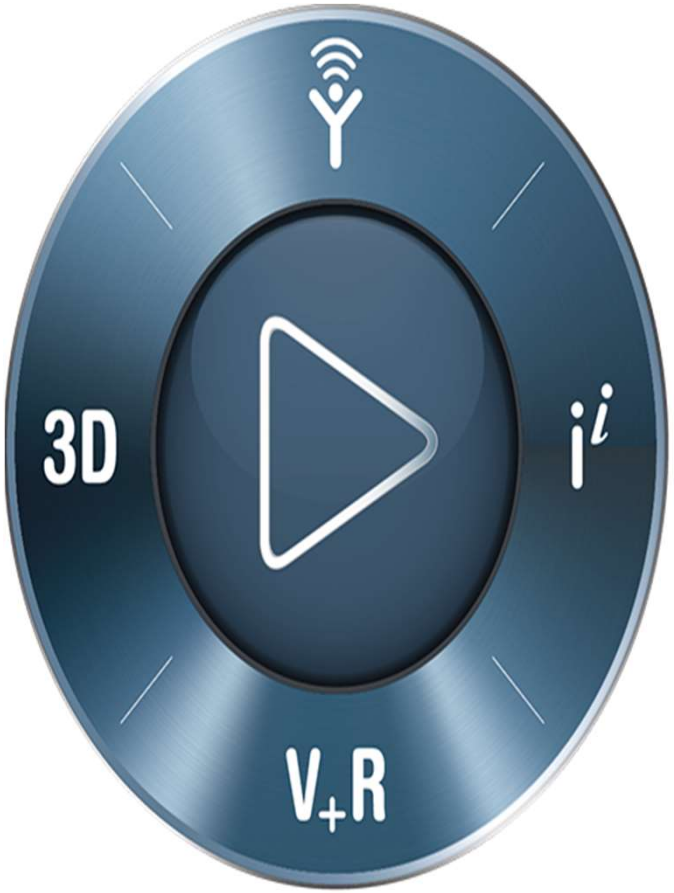
PATTERN: TABLES TO SELECT REPORT CONTENT

Pattern: Tables to Select Report Content (1 of 2)

- **Pattern: Use tables to**
  - Select data to report via table scope
  - To select details to report by what columns are displayed
  - To do complex queries and navigation between element
- **Good for: Everything!**
- **For example: Tables for structure of concept, logical, and physical models**
  - Use separate tables for each set
  - Include in scope only what needs to be documented
  - Only display columns pertinent to report
  - Use GenericTable to get data into report. Optionally use the Flattened Table Pattern to display all data and labels dynamically.

Pattern: Tables to Select Report Content (2 of 2)

- **Advantages:**
  - Reduce complexity of VTL code because all work is done in model and Table diagram
  - Custom columns can be used to generate data via complex query/script but VTL code is just GenericTable tool
  - Data in a table can use scope, or added via drag and drop, and edited to remove rows (Similar to SmartPackage)
- **Disadvantages**
  - Layout of report using Generic Table may not be easy for certain kinds of data(see Flattened Table)
  - Sorting can become difficult if you are trying to tell a story with elements in a specific order in document
- **Recommendations:**
  - Mix with other patterns like Report Model and Flattened Table.
- **Variation**
  - Matrix to Select Report Content



3DEXPERIENCE®

Pattern: ReoportWizard Selected Elements

Pattern: ReoportWizard Selected Elements

- Pattern: Use ReportWizard, the Selected Elements to filter elements
- Good for: When reporting a catalog of the model elements.
- For example, $Class will only have selected scope
- Advantages:
  - Reports are usually categorized by element type
  - Most often used pattern, so many examples
  - Can be used to select root objects like Package or SmartPackage that can then be iterated.
- Disadvantages
  - Layout of report is fixed and usually by type
  - Usually recursive, so hard to pick and choose specific content
  - Difficult to select a meaningful scope
  - Difficult to tell a story
- Recommendations:
  - Mix with other patterns

PATTERN: SMART PACKAGE FOR REPORT CONTENT DATA

Pattern: Smart Package for Report Content Data

- Need:
  - Fine control over what is included in a report
  - Model Architecture is in flux
  - Selection of data from ReportWizard inadequate or complex
  - Not all data is ready for inclusion in reports
  - Changing the order or nesting of data in a report often changes
- Solution:
  - SmartPackages used to choose data and define section contents
  - SmartPackage hierarchy used to control nesting of paragraphs.
  - Independence from Model refactoring
  - Only selected data appears in report.
  - Independent model using target model can be used to prevent accidental changes
  - Can use drag/drop or queries to populate SmartPackage
  - Other patterns can be used
- Benefits
  - Report configuration's is easier to maintain
  - Generally immune to refactoring of the model
- Recommendations
  - Nested SmartPackage with queries can cause performance issues
  - Keep package hierarchy flat! Avoid mirroring deep hierarchies in model
  - Avoid complex queries
  - Use a report model to limit access to report maintainers
  - Use ReportWizard to select SmartPackages to include in report

PATTERN: REPORT MODEL

Pattern: Report Model

- Problems:
  - Model and report management are mixed into same project
  - User accidentally break integrity of the model during refactoring or changes to tables/matrix diagrams intended for report
  - Adding report template data is cut/paste into existing models
  - Multiple configurations of reports and types of reports litter the model
- Solution Details:
  - Create a model for selecting data and managing control/content
  - Multiple report models for different reports
  - Report models can be created from model templates
    - Ensure consistency
    - Can be documented
    - Reduce access to SmartPackage, Tables,

Report Model

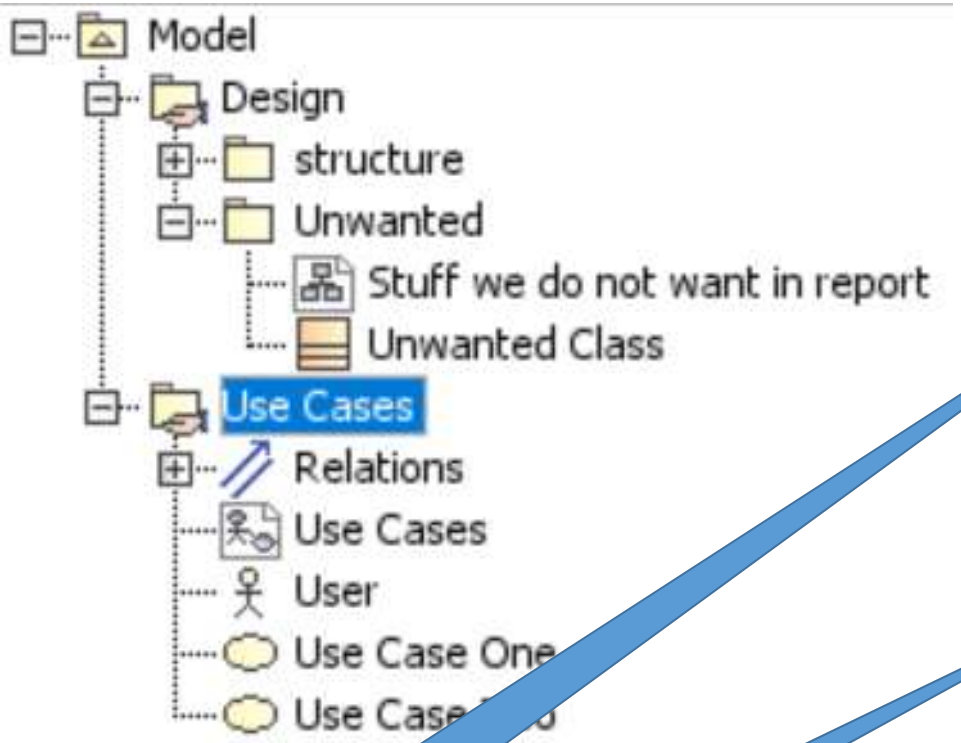
- Problems:
  - Model and report management are mixed into same project
  - User accidentally break integrity of the model during refactoring or changes to tables/matrix diagrams intended for report
  - Adding report template data is cut/paste into existing models
  - Multiple configurations of reports and types of reports litter the model
- Solution:
  - Create a model for selecting data and managing control/content
  - Multiple report models for different reports
  - Report models can be created from model templates
    - Ensure consistency
    - Can be documented
    - Reduce access to SmartPackage, Tables, Matrix, and diagrams to report model
  - Create Template to show user what is expected
- Issues
  - Cost of updating referenced model (switching versions)
  - More models to maintain
  - Need to ensure compliance to the report model pattern with templates and examples.

Report Model Example

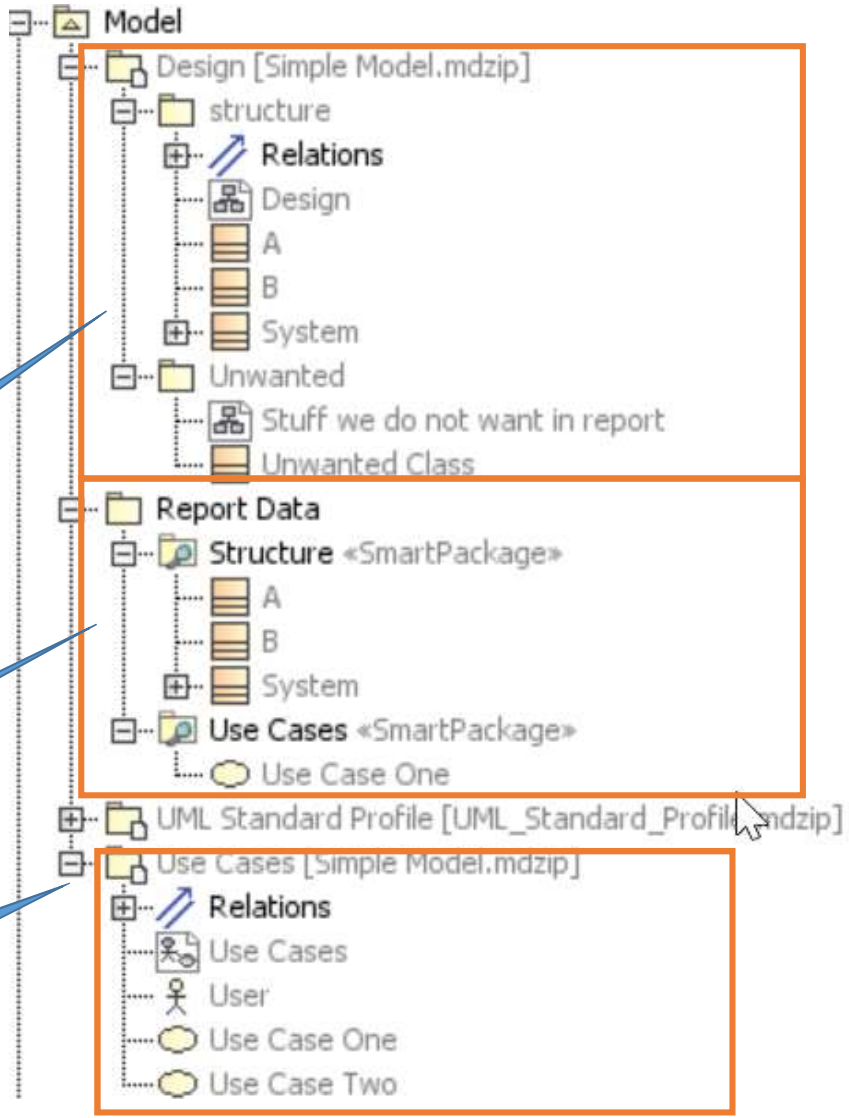
- Subject model is referenced
- Report model is configuration of report
  - SmartPackage Report Data to scope what is shown in report
  - SmartPackage is section name
- Note that this example limits the scope to only part of the sujext model that needs to be in the report

Report Model Example Using SmartPackage

Model of Interest



Report Model



Reported Model

Report Model Configuration

Reported Model

PATTERN: USE TABLES TO DRIVE REPORTS

Pattern: Use Tables to Drive Reports

- Generic Table tool can be used to display any table
- Replicates columns and headers
- Simplifies formatting
- Can be extended for specific types
- Most a simple use of the same text in the document
- New headers and column data can be created with Create Custom
- Advantages
  - Reduced formatting
  - Data alignment in table is easier to read
  - Edit table in the model, not the table in the report
  - What you see is what you get (mostly), so few surprises
- Disadvantages
  - Wide tables won't fit (See flattened table pattern)
  - One-to-many relationships may need special handling

PATTERN: FLATTEN TABLES

Pattern: Flatten Tables
Reporting via Flattening of Tables

- Problem: Reports are usually a collection of characteristics of a subject that require many steps:
  - Need to collect subject elements
  - Navigation from subject to characteristics
  - Paragraph headers need to be built
  - Create labels for characteristics to be displayed.
  - Format of each characteristic
- Disadvantages:
  - Difficult to maintain formatting versus content
    - Add or remove subject elements
    - Add/ or remove characteristic
  - Complex navigation of relationships from subject to related characteristic targets (for example requirements of a block)
  - Format of each characteristic is duplicated for each characteristic which can cause formatting to become out of synch.
- Bottom line: This brute force approach is costly to create and maintain.

Flattening of Tables

- Table becomes driver of all or part of a report
- Less need for editing of the report, just edit scope and columns of table
- Dynamically control scope of report by controlling scope of the table
- Dynamically control characteristics by controlling the columns in the table
- Commoditized and simplified formatting for a wider range of data
- Custom Column used to add new data and/or change the label or paragraph name
- Custom columns instead of pre-processing in template
- Users can see what will be reported in the tool prior to report generation.
- Disadvantages:
  - Complex template, but less complex than brute force methods.
  - Must have tables for the data
  - Tables and Columns need to be managed to ensure reports are compliant

Improvements to Flattened Tables

- Formatting can be commoditized by type (current example is based on strings)
- Table to Table relationships (via type or other method).
  - Current template is just one table
  - Could associate a table with a feature (Element type) and use another table and template. For example, a column type is requirement, another table and template can be used to render the data. This allows requirements to be flat or a table and to control the details of a usually one to many set of content.
- Lookups mapping columns to label/paragraphs can be used versus creating a new column for just that purpose.

A Flat Example: Setup

```
## Import the GenericTableTool
#import('generic','com.nomagic.reportwizard.tools.GenericTableTool')
## example shows how the GenericTable view allows us to create a table based on an existing table ##diagram and dynamically fill in the header and rows of data.
##
## Create an array to store our data from the table (we are using a variety for the example.
#set($tableTypes = ["Generic Table","Requirement Table","Glossary Table", "Instance Table"])
#set($genericTableArray = $array.createArray())
## Get each of the tables that are of type "Generic Table"
#foreach($diagram in $Diagram)
#if($tableTypes.contains($diagram.diagramType))
## Assignment to temp is needed because add returns a boolean.
#set($temp = $genericTableArray.add($diagram))
#end
#end
```

A Flat Example: Table Processing

```
## For each table, show the flattened data
#foreach($t in $genericTableArray)
##=== $t.diagramType ====
#end
#foreach($gt in $sorter.humanSort($genericTableArray))
## Print the name of the Table
$gt.name
## get the reference to the generic table
#set($table = $generic.getTable($gt))
## Get the visible columns
#set($cols = $table.getVisibleColumnIds())
## Remove column zero because this has the row counter
#set($temp = $cols.remove(0))
```

A Flat Example: Row/Column

```
## For each row/column get the contents (note: not using forrow/forcol
## because there is no table object).
#foreach($row in $table.getRows())
#foreach($col in $cols)
#if($table.getColumn($col)==$table.getColumn(1))
## Column zero is used as the paragraph name
$table.getValueAsString($row, $col)
##elseif($table.getColumn($col[1])==$table.getColumn($col))
## Column one is used as the name documenting the element
##$table.getValueAsString($row, $col)
#else
## Print the column header as the label, then the data of the cell.
## Note: we may need to handle collections for columns that are
## one to many relationships.
$table.getColumn($col): $table.getValueAsString($row, $col)#end

#end##col
#end##row
#end## End of the loop for each Generic Table
```

PATTERN: PREPROCESS COMPLEX DATA

Pattern: Preprocess complex data

- Problem: Need to pre-process data prior to use in template.
- Symptoms:
  - Data and VTL to process data can is complicated
  - Complex VTL in line with template
  - Templates are fragile and hard to debug
- Examples:
  - Most templates that are not table-based have this issue
- Solution(s):
  - Use scripting to pre-process
  - Preprocess data into arrays and hash maps then use the simplified structure in body of template (use macros to isolate)
  - Use table or flattened table
- Benefits:
  - Decouples processing from formatting tasks
  - Simplifies complex formatting like that used for tables
- Disadvantages
  - VTL for pre-processing is harder to manage (use tables instead)

Output of Improved Flat Table

- Note that formatting the table contents would be much harder in prior example.

Amplified Audio (Block)

Namespace	Interfaces
Documentation	
Part	Audio Signal : dbV
Owned Attribute	Audio Signal : dbV

Amplified Audio IF (Interface Block)

Namespace	Interfaces
Documentation	
Part	
Owned Attribute	out AudioSignal : Amplified Audio

Analog Audio IF (Interface Block)

Namespace	Interfaces
Documentation	
Part	
Owned Attribute	out AudioSignal : Line Audio

READING WRITING JSON (CODE OR DATA GENERATION)

PATTERN: READING AND WRITING SCHEMAS

- Problem: Generation of code, schemas, or complex configuration data
- Solution: Use templates to transform a model to code or config file
- Example: Read and write JSON
- JSON can be read or written via many open source libraries for Java or JS-233
- The following example is written to load a JSON file for **APACHE AVRO**
- Apache Avro is a data serialization framework that is used for many different applications
- The code only covers some of the core data model, but can be expanded.
- Note that this application requires that datatypes be created with the same names as the Avro types.
- Also include is a test file to read, a Report Wizard Template to write the model to JSON and an example output.
- Warning: This is just an example and not a complete solution.


Avro Schema.mdzip


AvroRead.rb


avro_test.json


AvroTemplateFile.json


test_out.avro.json

EXAMPLE: READ AVRO JRUBY

Files provided

```
################################################################################
##############
# Created by Daniel Brookshier (Daniel.Brookshier@3ds.com,
Daniel.Brookshier@gmail.com)
# Read Avro Schema JSON files
# Warning, model must be writable!
# Current capabilities:
#  Assumes that Avro type names are available in the datamodel (mostly extended
datatypes and a few that need to be created).
#  Assume schema namespace is a packege under root model.
#   Does not care where existing enumes are, but will creat in same package as
Record
# Todo: Array
#    Alias
#       Union
#       Need to synch the enumeration literals. Right now only creates if new
#    Better annotation to support round trip and compatibility with Avro Report
Template
#    Currently the type of the field is not synchronized on second read.
#    Need to set default value
################################################################################
##################
require 'java'
require 'json'
AutomatonMacroAPI = com.nomagic.magicdraw.automaton.AutomatonMacroAPI
Application = com.nomagic.magicdraw.core.Application
SessionManager = com.nomagic.magicdraw.openapi.uml.SessionManager
ParameterDirectionKindEnum =
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.ParameterDirectionKindEnum
VisibilityKindEnum =
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.VisibilityKindEnum
$log = Application.getInstance().getGUILog()

################################################################
# Get model root element.
def getModelRoot()
        return Application.getInstance().getProject().getPrimaryModel();
end
=begin
Find the first occurrence of the listed element under the supplied owner,
inclusive of the owner (basically one level? Hard to tell...)

   type  Type of desired element
   name  Name of desired element
   owner Element under which you search for the indicated element
=end
def getElement( type,  name,  owner)
    if (owner == nil)
        owner = getModelRoot();
    end
    elements = Array.new
    elements << owner;## the '<<' adds an element to an array.
    elements.concat(owner.getOwnedElement());
    current = nil;
    for current in elements
  #Application.getInstance().getGUILog().log(current.getHumanName())
        if (current.getHumanName() == (type + (name.empty? ? "" : " " + name)))
            return current;
        end
        if current.hasOwnedElement()
            elements.concat(current.getOwnedElement());
     end
    end
    return nil;
end
```

```
=begin
Get elements from a root (including root)
   parent Scope element under which you search for the indicated element
=end
def getElements( parent )
    elements = Array.new
    elements << parent;
    elements.concat(parent.getOwnedElement());

    for current in elements
        if current.hasOwnedElement()
            owned = current.getOwnedElement();
            for elm in owned
                if (!elements.contains(elm))
                    elements.concat(elm);
                end
            end
        end
    end
    return elements;
end
################################################################
def findPackage(packageName)
   return getElement( "Package",  packageName,  getModelRoot())
end
def readAvro( parsed, namespaceDefault)
   case parsed["type"]
     when "record"
        # create a package for the name space (need to refine and find, not just
create)
        # use default for the namespace if none exists
        if parsed["namespace"] == nil
          parsed["namespace"] = namespaceDefault
        end     pack = getElement( "Package",  parsed["namespace"],
getModelRoot())
        if pack == nil
          pack = $factory.createPackageInstance();
          pack.setName(parsed["namespace"]);
          pack.setOwner($project.getModel());
        end
        # create a class for the name space (need to refine and find, not just
create)
        clazz = getElement( "Class",  parsed["name"],  pack)
        if clazz == nil
          clazz = $factory.createClassInstance();
          clazz.setName(parsed["name"]);
          clazz.setOwner(pack);
        end
        parsed["fields"].each do |field|
          $log.log( "Creating field:"+field["name"]);
          property = getElement( "Property",  field["name"],  clazz)
          if property == nil
            property = $factory.createPropertyInstance() ;
            property.setName(field["name"]);
            property.setOwner(clazz);
          end
          #set the type for the field
          if field["type"] == "enum"
            $log.log("enum:"+field["name"])
            enum = getElement( "Enumeration",  field["name"],  getModelRoot())#we
don't care where enums are stored

            if enum != nil
              $log.log(enum.getName)
              end
```

```
            if enum == nil
              enum = $factory.createEnumerationInstance()
              enum.setName(field["name"])
              enum.setOwner(pack);
              # todo: Need to synch the literals. Right now only creates if new
              field["symbols"].each do |sym|
                literal = $factory.createEnumerationLiteralInstance()
                literal.setName(sym)
                literal.setOwner(enum)
              end
            end

            property.setType(enum)
          elsif $avroType[field["type"]] != nil
            $log.log("set type of field");
            property.setType($avroType[field["type"]])
            # set default todo
            literalString =
$project.getElementsFactory().createLiteralStringInstance();
            literalString.setValue(field["default"]);

            #property.setDefault(literalString);
          else#nested class
            $log.log("Nested class found")
            clazzz = readAvro(field["type"],parsed["namespace"])
            property.setType(clazzz)
          end
        end
      else
      end
    return clazz
end
################################################################
# Main Application
################################################################
begin
   #SessionManager.getInstance().closeSession();
   SessionManager.getInstance().createSession("Automaton_Macro_Script_Execute")
   $log.log("------ initializing logging script! ------------")
   $log.log("Ruby Version:  " +RUBY_VERSION)
   $log.log("JRuby Version: " +JRUBY_VERSION)
   string =
'{"desc":{"someKey":"someValue","anotherKey":"value"},"main_item":{"stats":{"a":8,"
b":12,"c":10}}}'
   parsed = JSON.parse(string) # returns a hash

   # Read JSON-based Avro schema from a file, iterate over objects
   $project = Application.getInstance().getProject();
   $factory = $project.getElementsFactory();
   ## Here we are getting the primitive types from the model to type the items read.
   $avroType = { "null" => getElement( "Data Type",  "null",  getModelRoot()),
     "boolean" => getElement( "Data Type",  "boolean",  getModelRoot()),
     "int" => getElement( "Data Type",  "int",  getModelRoot()),
     "long" => getElement( "Data Type",  "long",  getModelRoot()),
     "float" => getElement( "Data Type",  "float",  getModelRoot()),
     "double" => getElement( "Data Type",  "double",  getModelRoot()),
     "bytes" => getElement( "Data Type",  "btyes",  getModelRoot()),
     "string" => getElement( "Primitive Type",  "string",  getModelRoot())}
   file = File.read('C:/Users/DBR2/Documents/Example Models/Data Modeling and
SQL/Avro Schema/avro_test.json')
   parsed = JSON.parse(file)
   readAvro(parsed,"avroNamespace")
ensure
   SessionManager.getInstance().closeSession()
end
```

SETUP FOR JRUBY FOR JSON

Files provided

- These are instructions for installing the latest version of Jruby for use in the tool
- Similar instructions can be used for other JSR-233 scripting languages.

Introduction

These are instructions for setting up CATIA Magic tools (MagicDraw, Cameo, Magic Cyber, etc. to have JSON capability.

The reason that this is installing a separate instance for JRuby is to make it easier to add Gem files. Normally the tool is installed in a read only directory which can cause issues. With this method one file is changed and then we can add Gem files without further read access to the installation files.

Setup Steps

Install the latest JRuby

Please see the latest instructions for your version. These are based on 2021x, **MagicDraw MacroEngine UserGuide.pdf**, section 3.4 Installing Gems for JRuby.
https://www.magicdraw.com/files/manuals/MagicDraw MacroEngine UserGuide.pdf

Change the Plugin.xml to point to the installed JRuby.

Edit <Intstall Dir> \plugins\com.nomagic.magicdraw.automaton\plugin.xml

For example, given install of Jruby 9.3.1 in E directory:

Change default:

```
<library name="lib/engines/jruby-9.2.14.0/lib/jruby.jar"/>
```

To (note the use of forward slash):

```
<library name="e:/jruby-9.3.1.0/lib/jruby.jar"/>
```

Install the json-jruby Gem by opening the command console and running:

```
jruby -S gem install json-jruby
```

Add the following (based on your installation) to the JAVA_ARGS in your application properties file (cameoea.properties, magicdraw.properties, etc.). The application properties are in the bin directory of your CATIA Magic installation directory. For example, given install of Jruby 9.3.1 in E directory:

```
-Djruby.home\="e:/jruby-9.3.1.0" -Djruby.lib\="e:/jruby-9.3.1.0/lib"
```

Relaunch the tool and test your Ruby script

OUTPUT OF AVRO WITH REPORT WIZARD TEMPLATE

Files provided

```
## $importer indicates the context to print.
## Create Avro schema from UML-based schema
## Created by Daniel Brookshier
## daniel.brookshier@3ds.com.
daniel.brookshier@gmail.com
## Note that this is a work in progress and is not
yet complete.
## Partially completed: Record, Enum
##
## Todo
##fixed, array, needs to be based on selected.
##
## list of standard types
#set($avroTypes  =
['null','boolean','int','long','float','double','bytes',
'string'])
################################
#
## Macro for default values.          ##
################################
#
#macro(defaultValue $prop)
#if($prop.defaultValue)
## When the default value is the Enumeration
Literal element.
#if($prop.defaultValue.class.name ==
'com.nomagic.uml2.ext.magicdraw.classes.mdker
nel.impl.InstanceValueImpl')
##Get the literal value
"$prop.defaultValue.instance.name"#else
##When String, Integer, real, etc.
#if($prop.type.class.name ==
'com.nomagic.uml2.ext.magicdraw.classes.mdker
nel.impl.LiteralStringImp')
"$prop.defaultValue.value"#else$prop.defaultVal
ue.value#end
#end
#else
"NONE"#end
#end## macro
################################
#########
## Return the assigned type or union. ##
################################
#########
#macro(typeOrUnion $prop)
#set($aliases=$report.getStereotypeProperty($p
rop, 'Union', 'aliases'))
#set($size=$report.getStereotypeProperty($prop
, 'typeModifier', 'typeModifier'))
#if($avroTypes.contains($prop.type.name))
"$prop.type.name"##print normal type
#elseif($prop.type.humanName.contains("Enume
ration"))## contain needed on humanName
because it includes class.
"enum", "symbols" : [#foreach($lit in
$prop.type.ownedLiteral)"$lit.name"#if(
$foreach.hasNext ), #end#end]##Enumeration
print
#else
#record($prop.type)##Nested Type
#end
#if($size.size()>0),
"size":$size[0].replace('[','').replace(']','')#end
#if(!$list.isEmpty($aliases))
, "aliases": [#foreach($alias in
$aliases)"$alias"#if( $foreach.hasNext ),
#end#end
]#end
#end## end of macro
############################
##  Write record schema ##
############################
##
#macro(record $class)
{
  "type":"record",
  "namespace": "$class.namespace.name",
  "name":"$class.name",
  "doc" : "$class.documentation",
  "fields":[
  #foreach($prop in $class.ownedAttribute)
    {"name":"$prop.name","type":
#typeOrUnion($prop),"default":
#defaultValue($prop)}#if( $foreach.hasNext ),
    #end
  #end
  ]
}
#end## macro
## This runs the template
#record($importer)
```

Anti-patterns

"*Seemed like a good idea at the time.*"
– Architect of the Titanic

Why Anti-patterns?

- Create an understanding of a what can go wrong
- Avoid the anti-pattern versus a better pattern
- Recognize anti-patterns in existing templates
- Avoids making the error as your first step.
- Includes pattern used to recover from anti-pattern

Anti-pattern: Monolithic Template

- A template for a large document with many sections and formats
- Symptoms:
  - Single template file
  - Hard to read
  - Embedded VTL code everywhere
- Examples:
  - Everywhere!

- Recovery from Monolithic Templates
  - May be easier to start over
  - Any report pattern will be better
  - Possibly refactor to macros or section includes

Anti-pattern: Too Many Macros

- Symptoms:
  - Macros are dominant in template
  - Hard to read intent of template
  - Difficult to debug
  - Macros are a fix between pre-processing, formatting, and template
- Disadvantages:
  - Takes longer to write a good macro
  - Because macros are parameterized templates, they are not functions and thus not easily used as functions in a library
  - Reusable macros are hard to write
  - Formatting and processing are usually wildly different depending on use
- Recovery from Too Many Macros Anti-pattern
  - Use recommended patterns
  - Avoid deep call depth of macros
  - Eliminate the need for pre-processing
    - Flattened Tables
    - Report Model

Anti-pattern: Hard coded references

- Problem: Direct references used to locate data to be included in report
- Examples:
  - Report uses search of named elements
  - Report uses URI to locate elements
- Symptoms:
  - Report breaks after refactoring
  - Report does not work for different models
  - Report must be edited often
  - There is no effect from changing the data scope of the report
- Solution Patterns:
  - Use Selected Elements
  - Smart Packages
  - Flattened Tables
  - Report Model

Anti-pattern: Expected Data

- Problem: Report looks incomplete or fails for incomplete data
- Examples:
  - Template relies on elements that may not be complete or do not exist
  - Related anti-pattern: Hard coded references
- Symptoms:
  - Report works for example, but not current state of project
  - Project is agile
  - Report does not work for different models
  - Report must be edited often
- Solution Patterns:
  - Ensure only completed elements are reported
    - Use Selected Elements
    - Smart Packages
    - Flattened Tables
    - Report Model
  - Rewrite report to call out incompleteness (expensive)

Summary

- Recognizing Patterns and anti-patterns are a good way to design reports
- These patterns are only the beginning
- If you have a pattern, let me know

Questions?

No Magic