

User guide



FICO™ Xpress
Optimization Suite

FICO Optimization Solutions SDK

User Guide

Release 1.1.0

FICO internal documentation

Last update 7 April, 2017

This material is the confidential, proprietary, and unpublished property of Fair Isaac Corporation. Receipt or possession of this material does not convey rights to divulge, reproduce, use, or allow others to use it without the specific written authorization of Fair Isaac Corporation and use must conform strictly to the license agreement.

The information in this document is subject to change without notice. If you find any problems in this documentation, please report them to us in writing. Neither Fair Isaac Corporation nor its affiliates warrant that this documentation is error-free, nor are there any other warranties with respect to the documentation except as may be provided in the license agreement.

©2014–2017 Fair Isaac Corporation. All rights reserved. Permission to use this software and its documentation is governed by the software license agreement between the licensee and Fair Isaac Corporation (or its affiliate). Portions of the program may contain copyright of various authors and may be licensed under certain third-party licenses identified in the software, documentation, or both.

In no event shall Fair Isaac Corporation or its affiliates be liable to any person for direct, indirect, special, incidental, or consequential damages, including lost profits, arising out of the use of this software and its documentation, even if Fair Isaac Corporation or its affiliates have been advised of the possibility of such damage. The rights and allocation of risk between the licensee and Fair Isaac Corporation (or its affiliates) are governed by the respective identified licenses in the software, documentation, or both.

Fair Isaac Corporation and its affiliates specifically disclaim any warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The software and accompanying documentation, if any, provided hereunder is provided solely to users licensed under the Fair Isaac Software License Agreement. Fair Isaac Corporation and its affiliates have no obligation to provide maintenance, support, updates, enhancements, or modifications except as required to licensed users under the Fair Isaac Software License Agreement.

FICO and Fair Isaac are trademarks or registered trademarks of Fair Isaac Corporation in the United States and may be trademarks or registered trademarks of Fair Isaac Corporation in other countries. Other product and company names herein may be trademarks of their respective owners.

FICO Optimization Solutions SDK

Deliverable Version: A

Last Revised: 7 April, 2017

Version 1.1.0

Contents

Introduction	1
What is the Xpress Solutions SDK?	1
What are the prerequisites for using the Solutions SDK?	1
1 Constraints	2
1.1 Overview	2
1.2 Usage examples	2
1.2.1 Integrating the SDK with VDL views	2
1.2.2 Constraint Editor	2
1.2.2.1 Fixed Constraint Editor	3
1.2.2.2 User-Defined Constraint Editor	6
1.2.3 Automatic Suggestions	8
1.2.3.1 Repairing infeasibility	8
1.2.3.2 Improving the results	9
1.2.3.3 Example	9
2 Segmentation	13
2.1 Overview	13
2.2 Usage examples	13
2.2.1 Integrating the SDK with VDL views	14
2.2.2 Database table segmentation	14
2.2.3 Attributes of elements	15
2.2.4 Segment lookup	17
2.2.5 Graphical user interface	17
2.2.5.1 Optimization Modeler Integration	18
2.2.5.2 Segment Editor	18
2.2.5.3 Constraint Editor	19
2.2.5.4 Segment Group Editor	21
2.2.5.5 Tableau Integration	23
2.2.6 Advanced	25
2.3 Advanced concepts	25
2.3.1 Configuration	26
2.3.1.1 Memory and database	26
2.3.1.2 Cache management	26
2.3.1.3 Types	26
2.3.1.4 Operators	27
2.3.1.5 Hiding attributes	27
2.3.1.6 Automated sampling of attribute values	28
2.3.1.7 Multiple Domains	28
3 Utilities	30
3.1 Overview	30
3.2 Usage examples	30
3.2.1 Integrating the SDK with VDL views	31
3.2.2 Message handling	31
3.2.2.1 Logging routines	31

3.2.2.2	Messaging notification in the Optimization Modeler UI	32
3.2.3	Error message dictionary	34
3.2.4	Timers	35
4	History	36
4.1	Overview	36
4.2	Usage examples	36
5	Data Validation	40
5.1	Overview	40
5.2	Usage examples	40
5.2.1	Setting up the page	40
5.2.2	Adding a validation report	41
5.2.3	Adding a data validation editor	41
5.2.4	Validation rules in the Mosel model	42
6	File Browser	43
6.1	Overview	43
6.2	Usage examples	43
6.2.1	VDL and Javascript	43
6.2.2	Mosel model updates for using a file browser	44
6.2.3	Selecting an attachment as input file	45
6.2.4	Selecting an input file from a standard file system	47
6.2.5	Selecting and downloading an input file from an HTTP server	49
6.2.6	Selecting and downloading an input file from S3	50
7	Data Explorer	52
7.1	Overview	52
7.2	Usage examples	52
7.2.1	General data explorer	52
7.2.2	Element-wise data explorer	52
	Appendix	54
A	Contacting FICO	54
	Product support	54
	Product education	54
	Product documentation	54
	Sales and maintenance	55
	Related services	55
	About FICO	55
	Index	56

Introduction

What is the Xpress Solutions SDK?

The Xpress Solutions SDK is a set of modules that can be integrated into solutions based on the FICO Xpress Optimization Suite to add advanced capabilities. Each functional module is comprised of a Mosel package and corresponding JavaScript library, used to augment the application model and user interface respectively. This user guide documents usage examples for each of the modules. The *SDK Mosel Packages reference manual* provides a detailed reference for the Mosel packages.

The examples in this manual are organized by SDK module:

- Chapter 1: Constraint handling (*sdkconstraint*)
- Chapter 2: Segmentation (*sdksegment*)
- Chapter 3: Utilities (*sdkutil*)
- Chapter 4: History (*sdkutil*)
- Chapter 5: Data Validation Rule (*sdkdatarule*)
- Chapter 6: File Browser (*sdkfilebrowser*)
- Chapter 7: Data Explorer

What are the prerequisites for using the Solutions SDK?

The Solutions SDK requires the latest versions of the FICO Xpress and FICO Optimization Modeler software. Some examples feature Tableau workbooks and thus require the Tableau for FICO server software compatible with the latest version of Optimization Modeler.

The developer of a solution using the Solutions SDK will need knowledge of the Mosel language and of custom view development for Optimization Modeler. For more information please see the FICO Optimization Modeler Developer Guide.

CHAPTER 1

Constraints

1.1 Overview

The SDK Constraints module can be used to extend an optimization application with user defined constraint handling. The *sdkconstraint* package provides a standardized framework for handling user defined constraints within a Mosel model, and the corresponding JavaScript library provides the means to add a basic constraint editor to the Optimization Modeler user interface.

Furthermore, the module can also be used to build an infeasibility finder that can identify which constraints are causing infeasibility and suggest remedial corrections.

The following subsections will cover in detail the various use cases of the module.

1.2 Usage examples

This section documents several standard usage examples of the User Constraints module.

- Section 1.2.1 shows how to integrate the Solutions SDK into your application to add constraint editors to the web application views.
- Section 1.2.2 describes how to use the constraint editor in an Optimization Modeler application.
- Section 1.2.3 presents the automatic suggestion mechanism and its integration in an application.

1.2.1 Integrating the SDK with VDL views

To include the SDK into a VDL page, you must include *sdk.vdl* file within your *client-resources* folder. You must then include the *sdk* into your *vd1* page using a *vd1-include* tag:

```
<vd1-include src="sdk.vdl"></vd1-include>
```

The *sdk.vdl* file relies on the *sdk.min.js* and *sdk.css* files to be present in the *js* and *css* subfolders of *client-resources* respectively

To active the constraint editor on a particular *vd1-table*, you must add it as a modifier to the table as shown in the following examples.

1.2.2 Constraint Editor

A *constraint editor* is a specialized type of *vd1-table* that allows the index set of the constraint to be extended on the fly.

1.2.2.1 Fixed Constraint Editor

A *fixed constraint editor* works with pre-defined constraints. The user has the flexibility to change the logical operators and the right-hand side values of these constraints from an Optimization Modeler view. To create a fixed constraint editor in VDL, we can use the objects `CstrDescr`, `CstrOperator` and `CstrRHSVal` that are defined by the *sdkconstraint* package in a `vd1-table`. The below model extract demonstrates the use of the generic constraint editor framework. It uses the template function `ctrcreatesum` to create linear sum constraints.

```
uses "sdkconstraint"

declarations
  Coeff: array(set of mpvar) of real      ! Constraint coefficients
  x: array(R:range) of mpvar              ! Decision variables
end-declarations
...
CstrDescr::([1,2])["text1","text2"]      ! Constraint descriptions
forall(i in R) Coeff(x(i)):= 1            ! Some coefficient values
...
forall(c in Cstrs)
  ! Create a linear constraint with coefficients in Coeff (array indexed by mpvar)
  ctrcreatesum(c, Coeff, CstrRHSVal(c))
```

This is the corresponding snippet of VDL code.

```
<vd1-table>
  <vd1-table-column entity="CstrDescr"></vd1-table-column>
  <vd1-table-column entity="CstrOperator" editable="true"></vd1-table-column>
  <vd1-table-column entity="CstrRHSVal" editable="true"></vd1-table-column>
</vd1-table>
```

Here is now a complete example which has the Mosel model, companion file and VDL code that illustrate the use of the *fixed constraint editor*. In this example, we have two types of chess sets that we can make and they are subject to certain resource constraints. A user should be able to use the constraint editor to change the available resource amounts. The objective is to decide how many units of each type of chess set to make in order to maximize the total profit.

The `CstrDescr`, `CstrOperator` and `CstrRHSVal` arrays are indexed by the set `Cstrs` of the *sdkconstraint* package. To create the constraints, the user has to initialize all of these objects with default values. In the below example (file `chess2_genctr.mos` in the example *chess2_gen* of the SDK distribution) we will only allow the user to modify the right-hand side value of the resource constraints. We are using the function `ctrcreate` to create the constraints. This function will use the corresponding `CstrOperator` for creating each constraint.

```
model Chess2                                ! Start a new model
  uses "mminsight"                          ! Use Optimization Modeler
  uses "mmxprs"                             ! Load the Optimizer library
  uses "sdkconstraint"                      ! Use the sdkconstraint package

  !@om.manage input
  public declarations
    !@om.alias Product types
    !@doc.descr Types of chess sets to make
    UnitTypes: set of string
    !@om.alias Profit per unit
    !@doc.descr Profit per type of unit made
    ProfitPerUnit: array(UnitTypes) of real
    !@om.alias Unit resource requirement
    !@doc.descr Resource amount needed for making a unit
    UnitResourceRequirements: array(UnitTypes,Cstrs) of real
  end-declarations

  !@om.manage result
```

```

public declarations
    unitstobuild: array(UnitTypes) of mpvar      !@doc.descr Number of units to build
    MaxProfit: lincvtr                          !@doc.descr Maximize Profit
end-declarations

declarations
    !@doc.descr Coefficient array for Limit Constraints
    CoeffLimit: array(set of mpvar) of real
end-declarations

!@om.resultdata.delete=on-queue

! Initialize the problem instance data
if (insightgetmode = INSIGHT_MODE_RUN) then
    ! Inject scenario data and continue
    insightpopulate
else
    CstrEnable::([1,2])[true,true]
    CstrDescr:: ([1,2])["Available hours", "Available wood in Kg"]
    CstrOperator::([1,2])[CstrOperatorId("<="),CstrOperatorId("<=")]
    CstrRHSVal::([1,2])[200,400]
    ProfitPerUnit:: ([ "small","large"])[5,20]
    UnitResourceRequirements:: ([ "small"],[1,2]) [1,3]
    UnitResourceRequirements:: ([ "large"],[1,2]) [3,2]
    finalize(UnitTypes)
    finalize(Cstrs)

    ! Stop after data initialization
    if (insightgetmode = INSIGHT_MODE_LOAD) then
        exit(0)
    end-if
end-if

! Production amounts are discrete quantities
forall(u in UnitTypes) unitstobuild(u) is_integer

! Objective function
MaxProfit:= sum(u in UnitTypes) ProfitPerUnit(u)*unitstobuild(u)

! Write the resource constraints
forall(c in Cstrs) do
    ! Initialize the coefficients
    forall(u in UnitTypes)
        CoeffLimit(unitstobuild(u)):= UnitResourceRequirements(u,c)
    ctrcount+=ctrcreate(c,CoeffLimit,CstrRHSVal(c))
end-do

! Solve the problem
insightmaximize(MaxProfit)

! Display results
if getprobststat=XPRS_OPT then
    logging(0, "Results:")
    logging(0, "+Total profit: " + MaxProfit.sol)
    logging(0, "+-> Number of small sets produced: "+unitstobuild("small").sol)
    logging(0, "+-> Number of large sets produced: "+unitstobuild("large").sol)
end-if

end-model

```

The package *sdkconstraint* configures the appearance and management of all entities for the OM UI, for example it defines *CstrDescr* as labels for the constraint index *Cstrs*. This results in displaying a description of the constraints in the editor instead of the index values.

The companion file for this project (file *chess2_gen.xml* of the example *chess2_gen*) only has a single entry, specifying the name of the file that implements the VDL view.

```
<?xml version="1.0" encoding="UTF-8"?>
```



```
<model-companion xmlns="http://www.fico.com/xpress/optimization-modeler/model-companion"
  version="3.0">
  <client>
    <view-group title="Main">
      <vdl-view title="Chess VDL view" path="chess.vdl"/>
    </view-group>
  </client>
</model-companion>
```

Since we are only allowing the user to change the right-hand side values of the constraints, the only additional lines we will need in the VDL file (see file `chess.vdl` of the example `chess2_gen`) is the below.

```
<vdl-table>
  <vdl-table-column entity="CstrRHSVal" editable="true"></vdl-table-column>
</vdl-table>
```

Figure 1.1 shows what the resulting constraint editor will look like. The ‘Limits’ column entries can be edited to change the right hand side values.

Chess 2 VDL View

Scenario 'Scenario 1' created on Wed Jun 24 08:04:39 UTC+0200 2015

Resources

Constraints ▲	Limits
Available hours	200.0
Available wood in Kg	400.0

RUN

Figure 1.1: Constraint Editor

If we also wish to give the user the flexibility to specify the operator for each constraint, we need to add the `CstrOperator` entity to the `vdl-table` defining the constraint editor. We will include a drop down select menu to choose the applicable operator. Note that the `sdkconstraint` package configures labels (`CstrOperatorSymbols`) for `CstrOperators` to display the string representation of the operators instead of the operator index in the drop down selection menu.

```
<vdl-table>
  <vdl-table-column entity="CstrOperator" editable="true" editor-type="select"
    editor-options-set="CstrOperators"></vdl-table-column>
  <vdl-table-column entity="CstrRHSVal" editable="true"></vdl-table-column>
</vdl-table>
```

The resulting constraint editor is shown in Figure 1.2. Notice that the ‘Operators’ column entries are a drop down menu.

Chess 2 VDL View

Scenario 'Scenario 1' created on Wed Jun 24 08:54:40 UTC+0200 2015

Resources

Constraints ▲	Operator	Limits
Available hours	<=	200.0
Available wood in Kg	<= ▼	400.0



Figure 1.2: Constraint Editor with operator selection

If you want to restrict the operators to only a subset of the available operators, you can create a separate set of valid operators and use this set in the `editor-options-set` attribute of the VDL tag `vd1-column`.

You can also choose not to use the `ctrcreate` function and manually create the constraints as shown in this Mosel code snippet.

```
! Define the resource constraints
forall(c in Cstrs) do
  rhs:= CstrRHSVal(c)      ! Retrieve rhs value
  op:= CstrOperator(c)     ! Retrieve logical operator
  case CstrOperatorSymbols(op) of
    "<=":
      sum(u in UnitTypes) UnitResourceRequirements(u,c)*unitstobuild(u) <=
        CstrRHSVal(c)
    ">=":
      sum(u in UnitTypes) UnitResourceRequirements(u,c)*unitstobuild(u) >=
        CstrRHSVal(c)
    "=":
      sum(u in UnitTypes) UnitResourceRequirements(u,c)*unitstobuild(u) =
        CstrRHSVal(c)
  end-case
end-do
```

1.2.2.2 User-Defined Constraint Editor

A *user-defined constraint editor* extends the functionality of the fixed constraint editor with the ability to create new constraints from a VDL view. Such *user-defined constraints* are created by combining *aggregation operators* and *decisions* (= a selector for a set of decision variables) that are defined in the Mosel model using the conventions of the *sdkconstraint* package.

In order to be able to add the *user-defined constraint* to an Optimization Modeler application, it is necessary to state which are the decisions that will be controlled by the end-user. This is done by populating the set `CstrDecisions` with the relevant decision indices, and the array `CstrDecisionLabels` with the corresponding labels to be displayed in the user interface.

Decisions are used as an interface between the model and the end-user, and are converted during the model execution into mathematical constraints. The creation of the mathematical constraints is done in the same way as for the *fixed constraints*.

The Mosel code snippet below is taken from the *traintimes_usrctr* example application (file *traintimes_om_usrctr.mos*). It shows the definition of the underlying data structures and constraint templates for the generation of new constraints from the user interface:

```

declarations
  tempCoeff: dynamic array(mpvvar) of real
  ! Variable indicating which train was chosen
  choose: array(TRAINS) of mpvar
end-declarations

forall(c in Cstrs) do
  if(CstrEnable(c)) then
    ! Reset the coefficient array
    delcell(tempCoeff)
    ! Define coefficients as an array indexed by 'choose' variables
    forall (t in TRAINS)
      tempCoeff(choose(t)):= CoeffDecCtrs(CstrDecision(c),t)
    ! Define coefficients for slack variables
    if(CstrOperator(c)=CstrOperatorId("<=")) then
      tempCoeff(dev(c)):= -1
    else
      tempCoeff(dev(c)):= 1
    end-if

    ! Add the constraints to the mathematical model
    ctrcreatesum(c, tempCoeff, CstrRHSVal(c))
  end-if
end-do

```

From the Optimization Modeler UI (VDL views), the user can then create new constraints by specifying the relevant entities in a special vdl-table (attribute `modifier="=sdk.modifiers.userConstraintEditor"`, whereas the alternative setting that creates a fixed constraint editor is `modifier="=sdk.modifiers.constraintEditor"`). Here is an example of VDL code (file *traintimes_usrctr.vdl*) implementing a user-defined constraint editor:

```

<h5>Constraint Editor</h5>
<vdl-table id="constraint" modifier="=sdk.modifiers.userConstraintEditor">
  <vdl-table-column set="Cstrs"></vdl-table-column>
  <vdl-table-column entity="CstrDescr" editable="true" width="200"></vdl-table-column>
  <vdl-table-column entity="CstrDecision" editable="true" class="text-left"
    editor-type="select" editor-options-set="CstrDecisions"></vdl-table-column>
  <vdl-table-column entity="CstrEnable" editable="true"></vdl-table-column>
  <vdl-table-column entity="CstrOperator" editable="true"
    editor-type="select" editor-options-set="CstrOperators"></vdl-table-column>
  <vdl-table-column entity="CstrRHSVal" editable="true"></vdl-table-column>
</vdl-table>

```

The *constraint editor* graphical component is created by adding the attribute `modifier="=sdk.modifiers.userConstraintEditor"` to a vdl-table. This VDL component modifier will add in the relevant Javascript code necessary to enable the addition and removal of rows and also to autoincrement the index set *Cstrs*.

The VDL code printed above results in a web view like the one in Figure 1.3.

Constraint Editor

Constraints	Description	Variable	Apply	Operator	Limits
1	Price Less Than \$50	Price	true	<=	50.0
2	At most one hop	Changes	true	<=	10
3		Arrival			

+ -

Arrival Time
Changes
Departure Time
Price
Reservation

Figure 1.3: User-Defined Constraint

Readers who are interested in more details and further examples of the *user-defined constraint editor* are referred to the section 2.2.5 about data segmentation and how to create constraints based on user-defined segments.

1.2.3 Automatic Suggestions

The *sdkconstraint* package provides a framework for incorporating automatic suggestions for repairing your model if it is infeasible or on improving your model for better results. To incorporate this framework we need to make suitable additions to the Mosel code and the VDL view definition. In this subsection we shall describe both suggestion mechanisms and also demonstrate their use through an example.

1.2.3.1 Repairing infeasibility

The Suggestion framework will provide automatic suggestions on how to repair your model if it is infeasible. The constraints that are causing the infeasibility are identified and suggestions are given on how to change these constraints to achieve feasibility. The suggestions are then presented to the user who can choose to apply the suggested repairs.

To use the framework, in addition to the `Cstr*` arrays that we have employed in the definition of constraint editors, we need to initialize the `CstrSlack` array. If the `CstrSlack` array is initialized, when a constraint is created using the `ctcreate*` family of functions, it automatically creates a *slack variable* for this constraint. However, by default, this variable will be hidden and will not be applied to the model. The steps that you can follow for introducing repair suggestions into the Mosel model are listed below.

1. Apply suggestions from previous run and clear the suggestions.
2. Include `CstrSlackObj` defined in the *sdkconstraint* package in the objective.
3. Run the optimizer.
4. If the problem status is 'infeasible', unhide slacks using `unhideAllSlacks` and run the optimizer again.
5. Call the `cstrsuggestrepair` procedure.

Notice that at the point of defining the constraint, we must check if the user chose to apply a suggestion from the previous run. If the user did choose a suggestion, then we must modify the right-hand side of the relevant constraint. The use of `cstrsuggestrepair` is demonstrated in the example at the end of this section.

1.2.3.2 Improving the results

The Suggestion framework can provide automatic suggestions on how to improve the result of a model. The model looks for constraints that are binding and suggests modifications that can possibly lead to a better result. The suggestions are then presented to the user who can choose to apply the suggested improvements. The steps that you can follow for introducing improvement suggestions into the Mosel model are listed below.

1. Apply suggestions from previous run and clear the suggestions.
2. Run the optimizer.
3. If problem status is 'optimal', call the `cstrsuggestimprovement` procedure.

Notice that at the point of defining the constraints, we must check if the user chose to apply a suggestion from the previous run. If the user did choose a suggestion, then we must modify the right-hand side of the relevant constraint. The use of `cstrsuggestimprovement` is demonstrated with an example in the next section.

1.2.3.3 Example

We will use the same Chess example as before to demonstrate the use of the Suggestion framework. We have introduced two new constraints limiting the number of large and small chess sets that can be made (see file `chess2_sugg.mos` of the *chess2_sug* example).

```

model Chess2                                ! Start a new model
uses "mminsight"                            ! Use Optimization Modeler
uses "mmxprs"                               ! Load the Optimizer library
uses "sdkconstraint"                        ! Use the sdkconstraint package

!@om.manage input
public declarations
  !@om.alias Product types
  !@doc.descr Types of chess sets to make
  UnitTypes: set of string
  !@om.alias Profit per unit
  !@doc.descr Profit per type of unit made
  ProfitPerUnit: array(UnitTypes) of real
  !@om.alias Unit resource requirement
  !@doc.descr Resource amount needed for making a unit
  UnitResourceRequirements: array(UnitTypes,Cstrs) of real
end-declarations

!@om.manage result
public declarations
  unitstobuild: array(UnitTypes) of mpvar    !@doc.descr Number of units to build
  MaxProfit: lincpr                          !@doc.descr Maximize Profit
  !@om.alias Solution Status:
  !@doc.descr Is solution optimal
  Optimal: string
  !@doc.descr Result Profit
  Profit: real
end-declarations

declarations
  CoeffLimit: array(set of mpvar) of real    ! Coefficient array for Limit Constraints
  CoeffProdSmall: array(set of mpvar) of real ! Coefficient array for Prod Constraints
  CoeffProdLarge: array(set of mpvar) of real ! Coefficient array for Prod Constraints
end-declarations

!@om.resultdata.delete=on-queue

```

```

! Initialize the problem instance data
if (insightgetmode = INSIGHT_MODE_RUN) then
    ! Inject scenario data and continue
    insightpopulate
else
    CstrEnable::([1,2,3,4])[true,true,true,true]
    CstrDescr:: ([1,2,3,4])["Available hours", "Available wood in Kg",
        "Number of small sets to build","Number of large sets to build"]
    CstrRHSVal::([1,2,3,4])[200,400,2,66]
    CstrOperator::([1,2,3,4])[CstrOperatorId("<="),CstrOperatorId("<="),
        CstrOperatorId(">="),CstrOperatorId(">=")]
    CstrDecision::([1,2,3,4])["Resource","Resource","ProdSmall","ProdLarge"]
    CstrSlack::([1,2,3,4])["ResourceSlack","ResourceSlack","ProdSmallSlack","ProdLargeSlack"]
    ProfitPerUnit:: ([ "small","large"])[5,20]
    UnitResourceRequirements:: ([ "small"],[1,2]) [1,3]
    UnitResourceRequirements:: ([ "large"],[1,2]) [3,2]
    finalize(UnitTypes)
    finalize(Cstrs)

    ! Stop after data initialization
    if (insightgetmode = INSIGHT_MODE_LOAD) then
        exit(0)
    end-if
end-if

! Production amounts are discrete quantities
forall(u in UnitTypes) unitstobuild(u) is_integer

forall(c in Cstrs) do
    ! Apply repairs and suggestions
    forall(s in SuggestTypes) do
        if(exists(SuggestSelect(string(c),s)) and SuggestSelect(string(c),s)) then
            writeln("RHS before suggest: ",CstrRHSVal(c))
            CstrRHSVal(c):= real(SuggestValue(string(c),s)) ! Modify RHS based on suggestions
            writeln("RHS after suggest: ",CstrRHSVal(c))
        end-if
    end-do

    ! Write constraints
    case CstrDecision(c) of
        "Resource": do
            forall(u in UnitTypes)
                CoeffLimit(unitstobuild(u)):= UnitResourceRequirements(u,c)
            ctrcount+=ctrcreate(c,CoeffLimit,CstrRHSVal(c))
        end-do
        "ProdSmall": do
            CoeffProdSmall(unitstobuild("small")):= 1
            ctrcount+=ctrcreate(c,CoeffProdSmall,CstrRHSVal(c))
        end-do
        "ProdLarge": do
            CoeffProdLarge(unitstobuild("large")):= 1
            ctrcount+=ctrcreate(c,CoeffProdLarge,CstrRHSVal(c))
        end-do
    end-case
end-do

! Reset all suggestions
ctrresetsuggest

! Objective function
MaxProfit:= sum(u in UnitTypes) ProfitPerUnit(u)*unitstobuild(u) - CstrSlackObj

! Solve the problem
insightmaximize(MaxProfit)

! If infeasible then unhide slacks to find infeasibility
if(getprobstat=XPRS_INF) then
    logging(0, "*** Problem is infeasible **")
    logging(0, "Resolving with slack variables...")

```

```

    Optimal:= "Infeasible"
    unhideAllSlacks
    insightmaximize(MaxProfit)
elif(getprobstat=XPRS_OPT) then
    logging(0, "Optimal solution found")
    Optimal:= "Optimal"
    Profit:= sum(u in UnitTypes) ProfitPerUnit(u)*unitstobuild(u).sol
end-if

if (getprobstat=XPRS_INF) then
    exit(1, "Problem is infeasible and cannot be repaired")
end-if

! If slack was used employ the repair model else look for improvement
if (CstrSlackObj.sol>0) then
    cstrsuggestrepair
else
    cstrsuggestimprovement
end-if

! Display results
if getprobstat=XPRS_OPT then
    logging(0, "Results:")
    logging(0, "+Total profit: " + MaxProfit.sol)
    logging(0, "+-> Number of small sets produced: "+unitstobuild("small").sol)
    logging(0, "+-> Number of large sets produced: "+unitstobuild("large").sol)
end-if
end-model

```

To display the suggestions to the user we will use the `SuggestSelect`, `SuggestDescr` and `SuggestValue` entities that are defined and configured via `om` annotations in the `sdkconstraint` package. These objects will be populated with the suggestions that the model generates. To display them, we will need to create a `vd1`-table with these objects.

The following `VDL` code (file `chess2_sugg.vdl` of the `chess2_sug` example) creates the suggestion table shown in Figure 1.4.

```

<vd1-section heading="Improvement/Repair Options" heading-level="2" vdl-if-results="0">
  <vd1-row>
    <vd1-table>
      <vd1-table-column entity="SuggestDescr"></vd1-table-column>
      <vd1-table-column entity="SuggestSelect" editable="true"></vd1-table-column>
    </vd1-table>
  </vd1-row>
  <vd1-row>
    <vd1-execute-button mode="RUN" caption="Run"></vd1-execute-button>
  </vd1-row>
</vd1-section>

```

Improvement/Repair Options

Suggestions▲	SuggestTypes⬇	Description⬇	Apply⬇
1	improvement	Improve 'Available hours' by increasing value to 202	false



Figure 1.4: Suggestions

In the table in Figure 1.4, the user can choose to apply the suggestion by changing the value in the 'Apply' column to 'true' and clicking Run. This will cause the right-hand side of the constraint in the constraint editor to change.

CHAPTER 2

Segmentation

2.1 Overview

The SDK Segmentation module can be used to extend an application with a user defined segmentation of a population of elements (e.g. customer accounts). This functional module comprises the Mosel package *sdksegment* that defines new functionality for the Mosel language, and a JavaScript library for use with an Optimization Modeler user interface.

The framework supports segmentation information residing in an external database or in memory. It enables the use of the segmentation in constraints and as filters on result data. The module provides a ready-to-go segmentation editor which allows the user to create new segments by specifying one or more filter conditions on the attributes of the population.

The module also provides functionality to create logical groups of segments. Group properties can be used to make sure that the group content follows a particular structure, like ensuring that the segments of the group covers all records in the database.

The following sections will cover in detail the various use cases of the module.

2.2 Usage examples

This section documents several standard usage examples of the User Segmentation module.

- Section 2.2.1 shows how to integrate the Solutions SDK into your application to add the data segmentation capability to VDL web application views.
- Section 2.2.2 describes the integration of the *sdksegment* package with a relational database. It presents how to connect and retrieve information from a database, and then how to create filters and segments on the database.
- Section 2.2.3 presents the general concept of attributes and how they are used to create filters. The section also describes how to copy data from the database to the Mosel memory for offline data processing.
- Section 2.2.4 shows how to use the *sdksegment* package to retrieve the segments that contain a particular element.
- Section 2.2.5 discusses the integration of the *sdksegment* package functionality with the Optimization Modeler user interface via specific VDL components.
- Section 2.2.6 presents advanced concepts that are further detailed in the various test models shipped with the *sdksegment* package.

2.2.1 Integrating the SDK with VDL views

Firstly, you need to include the SDK in your VDL page.

```
<vdl-include src="sdk.vdl"></vdl-include>
```

This will make the `sdk-segment-editor` and `sdk-segment-group-editor` VDL tags available in the view and will add in the code that is required for working with segment filters and segment group editors.

2.2.2 Database table segmentation

The `sdksegment` package provides the routine `segloadcfgdb` to automatically determine the eligible attributes for data segmentation. The routine will treat each field of a database table as a potential element attribute that can be used in filtering.

In the following example, the database table **CUST_TABLE** represented in the table 2.1 is scanned to retrieve the name and type of each column.

Table 2.1: Example of a Customer attribute table

customerid	status	birthdate	age	gender
1	Current	1964-10-24	36	male
2	Prospect	1957-05-15	43	female
3	Former	1956-05-02	46	male
4	Former	1932-02-24	68	female
5	Former	1963-05-03	37	male
6	Former	1983-09-30	17	male
7	Former	1923-02-21	73	male
8	Current	1995-02-07	5	female
9	Former	1986-10-06	14	male
10	Former	1994-12-29	6	female

In the Mosel code snippet printed below that is an extract from the file `testtrdbms.mos` in the folder `src/test/mosel` of the SDK distribution, the call to `segloadcfgdb` is used to retrieve the column names and types. This call will also create an associated *attribute* for each column. The mapping between the *attribute*, the table and the column will be stored in a specific data structure for future reference.

```
segloadcfgdb('CUST_TABLE')                                ! Scan table CUST_TABLE
```

Once attributes are loaded, it is possible to create *filters* and associate them with segments by using the routine `segaddcondition`. Each segment is associated to a domain *Customers*. Domains are used to classify the segments that are filtering different sets (for examples segments over *Customers* and segments over *Offers*). A segment without any filters will contain all records.

```
segment(SEG_ALL,'Customers','All customers')              ! All customers
segment(SEG_GND_M,'Customers','All male')                 ! Create the segment object
segaddcondition(SEG_GND_M,'gender','=','male')            ! Add condition
segment(SEG_GND_F,'Customers','All female')
segaddcondition(SEG_GND_F,'gender','=','female')
segment(SEG_GND_U,'Customers','Unknown gender')
segaddcondition(SEG_GND_U,'gender','not in ','male,female') ! Unknown gender
```

Once the filters are defined and the database connection is open it is possible to seamlessly query the database. For example, in the Mosel code snippet below the call to `seggetsize` triggers the

creation of the filter and if needed of the SQL statement that will be used to count the number of elements in the set. If the filter is valid, the query is executed directly in the database.

```
writeln("Number of customers in database:", seggetsize(SEG_ALL))
writeln("Number of invalid customer records:", seggetsize(SEG_GND_U))
```

Calling `segget` also triggers the creation of the filter and of the SQL statement that will be executed to retrieve the elements of the resulting subset. But in this case, the evaluation of the SQL query will fetch the element identifiers instead of counting them.

```
writeln("The set of women: ", segget(SEG_GND_F))
```

Note about in-memory and database operations:

The *sdksegment* package is designed to support in database operations by creating and executing SQL statements as well as in-memory operations by looking up Mosel arrays. The user can directly call the associated routines *segdb** for database operations, and *segmem** for in-memory operations. It is also possible to use the routines *seg** that let the package decides which family of routines to employ based on the availability of a database connection that has been opened with the *segdbopen* routine.

2.2.3 Attributes of elements

In the context of customer segmentation, the attributes are single dimensional information associated with a customer. For example, the unique customer identifier, her gender, her age or her postal code are single dimensional attributes that can be used when defining the data segmentation filter.

When the element attributes are stored in a relational database, it is often more efficient to run all data segmentation queries directly inside the database. This allows for using the fast query execution engine of the database and reduces the amount of data that are stored in Mosel memory. Once the attributes are loaded, it is possible to retrieve the set of possible values for a particular *attribute* by calling the routine *segssample* (resp. *segmemsuggest* for in memory attributes and *segdbsuggest* for attributes stored in the database). This routine will retrieve sample values for the given attributes, that can be used in the application view or in the Mosel model.

In the following pseudo Mosel code the possible values for the attribute 'gender' are retrieved by a call to *segssample*:

```
segdbopen(DB_CONNECTION)           ! Open database connection
segloadcfgdb("CUST_DATA")         ! Load attribute definition
...
segssample('gender')              ! Fill SegAttrSuggest with sampled values
writeln("Attribute 'gender': ",)
forall(v in SegAttrValues | exists(SegAttrSuggest('gender',v)))
writeln(SegAttrSuggest(a,v))
...
segdbclose                        ! Close database connection
```

In some cases one may wish to load the values of all attributes in memory. This allows the model to run without requiring it to keep alive the database connection. It also allows the developer to directly read or write the element attribute values. To do so, the developer can either use the routine *segcopydbtomem* that copies all attribute values from the database to memory or perform the copy manually using the *seggetattr* and *segsetattr* routines.

```
segloadcfgdb("CUST_DATA")           ! Load attribute definition
! Create master segment
```

```

segment(SEG_ALL,Customers,"All customers in the database")

! Create 'adult' segment
segment(SEG_ADULT,Customers)
segaddcondition(SEG_ADULT,"age", ">=",18)

! Retrieve the elements from DB
Customers := segdbget(SEG_ALL)

! Copy attribute values to memory
segcopydbtomem(SEG_ALL,Customers,{"gender","age"},"CUST_DATA")

! Retrieve age of customer 123
age123 := seggetattrnum(123,"age")
writeln("Age of customer '123' = ", age123)
writeln("Number of adults: ", segmemgetsize(SEG_ADULT))

! Change customer's segment
segsetattr(123,'age',if(age123>=18,10,45))
writeln("Number of adults: ", segmemgetsize(SEG_ADULT))

```

The attributes and associated metadata are stored using certain conventions. The *default attribute name* is obtained by decorating the attribute identifier and is stored in `SegAttrName`. In order to make sure you are using the right convention for attribute identifier naming the package provides the routine `segattrid`.

```

writeln(segattrid("AgE"))           ! Displays: 'age'
writeln(segattrdecorate("CreditClass")) ! Displays: 'Credit Class'

```

It is also possible to prevent *sdksegment* from automatically decorating the attribute names by setting the *attrdecorate* module option. When this option is set to *'none'* decoration is not performed and the attribute identifier will be displayed in the UI. It is also possible to manually set the decorated name by setting the parameter to *user*. The user can then set the attribute name by filling the array `SegAttrName`.

```

writeln(segattrid("AgE"))           ! Displays: 'age'

segattrid("AgE of the captain")     ! age_du_capitaine
segsetparam("attrdecorate","none")  ! Do not decorate
segattrdecorate("age_of_the_captain") ! "age_of_the_captain"

segsetparam("attrdecorate","user")   ! User decoration
segattrdecorate(attrid)              ! Use default: "Age Of The Captain"
SegAttrName(attrid):= "L'age du capitaine" ! Set decorated name
segattrdecorate(attrid)              ! "L'age du capitaine"

segsetparam("attrdecorate","auto")

```

The following example assumes we have a spreadsheet that contains the two columns "custid" and "FICO". The FICO score is retrieved from this file and stored in a temporary array "FICO". The spreadsheet is then analyzed by a call to the routine `segloadcfgcsv`, which will create two attributes "custid" and "fico". It is then up to the developer to copy the relevant data into *sdksegment* data structures by calling `segsetattr`. Once the data are loaded into *sdksegment* the temporary array is no longer needed and should be deleted to reduce the memory footprint.

Remark: Treating the element ID as an attribute is only required if you want to create filters with conditions on the element ID. Otherwise this is useless because the element ID is already known since it is the primary key for the element. You should also notice the call to `segattrid` which ensures that we are using an attribute name that satisfies the attribute naming convention (letters in lower case).

```

declarations

```

```
Customers: set of integer
FICO: dynamic array(Customers) of real
end-declarations
initializations from "customers.csv" ! Load customer data
FICO
end-initializations

segloadcfcsv("customers.csv")      ! Determine attributes from a CSV file
forall(i in Customers)
  segsetattr(segattrid("id"),i)      ! Set customer IDs
  segsetattr(segattrid("FICO"),FICO) ! Copy attribute values to sdksegment
delcell(FICO)                       ! Free temporary array
```

2.2.4 Segment lookup

The package *sdksegment* provides the routine `segmemfindsegments` to retrieve the segments an element belongs to. It also defines the routine `seglookupsegment` to find a segment that is defined by bounds on a single attribute.

In the example below, three segments are created. Each segment is defined over a single attribute 'gender', and they describe a partition of the data because each element belongs to a single segment. The routine `seglookupsegment` is used to retrieve the segment that contains all women, while the routine `segmemfindsegments` is used to find all the segments containing a particular customer.

```
declarations
Customers: set of string
end-declarations
segexample(Customers,1000) ! Create 1000 customers

! Create 3 tiers based on annual revenue
forall(s in 1..3) do
  segment(s,Customers)
  segaddcondition(s,'annualrevenue','>=',real(floor(20*(s-1)/3)))
  segaddcondition(s,'annualrevenue','<',real(floor(20*s/3)))
end-do

! Create a gender based segment filter
segment(4,Customers)
segaddcondition(4,'gender','=','M')

! Retrieve the segments containing a particular element
i:=123
writeln(" Customer ", i, " is part of segments: ")
forall(s in segmemfindsegments(i)) writeln(" - ",s)

! Retrieve segments based on gender
seg:=seglookupsegment('gender','M')
writeln(" Segment gender='M': ", seg)
seg:=seglookupsegment('gender','F')
if (seg=SegmentT(SEG_INVALID)) then
  writeln(" Segment gender='F' does not exists.")
end-if
```

2.2.5 Graphical user interface

The *sdksegment* package offers unique functionality to dynamically create rules for aggregating metrics or to create new types of constraints from an Optimization Modeler user interface. The graphical component *segment editor* allows the user to interactively create slices of data which can then be used in the Mosel model for the definition of *user-defined constraints* on subsets of decisions.

The package also provides tools that can be used in Optimization Modeler applications to aggregate metrics and create decision analysis dashboards. The data segments define dimensions

of the whole dataset that can be explored efficiently via Tableau views.

2.2.5.1 Optimization Modeler Integration

sdksegment is designed to be used with Optimization Modeler. In order to integrate the package with the application it is necessary to follow the steps in the section 2.2.1. Note that all model entities of the SDK package *sdksegment* that are required by the UI are readily configured via the corresponding `om` annotations at their declaration. You only need to edit these annotations (and recompile the package source) if you wish to modify their default settings, e.g. to display a different *alias* value.

Remark: In order to fully support all features of the *SDK*, the JavaScript and associated CSS files from the subdirectories *lib/js* and *lib/css* of the SDK distribution must be copied to the corresponding directories of the final application.

2.2.5.2 Segment Editor

The segment editor is a VDL extension used to manage the segments and the associated filters. It is presented in the screenshot in Figure 2.1 where the list of filters defines the available segments.

These segments can be automatically generated by the Mosel model (for example creating 10 tiers based on annual revenue). They can also be defined by the user by adding conjunctive conditions based on the element attributes. The *condition editor* is populated with the attribute list and value suggestions determined during the last execution of the Mosel model. All conditions must be satisfied for an element to belong to the corresponding segment.

Filters

▶ All customers in database	10000 matches
▶ Can be contacted by SMS	2965 matches
▶ Can be contacted by email	2976 matches
▶ Can be contacted by telephone	5022 matches
▶ Can be contacted by letter	2498 matches
▶ Customers aged at least 18 years	9834 matches
▶ High annual revenue customers	3952 matches
▼ Customers aged 18-40 living single	1217 matches

✖ birthdate ▾ <= ▾ 1996-11-21

✖ birthdate ▾ > ▾ 1974-11-21

✖ householdsize ▾ = ▾ 1

New Condition

Figure 2.1: Segment Filter Edition

In order to add the segment editor feature to a web application you need to use the *sdksegment* package in the underlying Mosel model and add the *sdk-segment-editor* extension to the corresponding VDL page. The example *folio* shows how data segmentation can be added for reporting based on the share characteristics.

The example in Figure 2.2 shows the code to be added to a web view to load the Javascript libraries and CSS of the SDK.

Figure 2.2: (file web/segment-editor/segment-test.vdl)

2.2.5.3 Constraint Editor

Segments can also be used to create constraints. A *constraint editor* can easily be added to the Optimization Modeler user interface by using the VDL component `autotable` and setting the attribute `sdk-constraint="user"`. This feature gives the end user the ability to add new constraints defined over interactively created sets of elements directly from the GUI.

The set of *decisions* that can be manipulated by the end user is defined in the Mosel model. The logic and meaning of the exposed decisions is also managed in the model. This level of indirection make it possible to expose more business information to the end user.

In the application example *folio*, only one type of decision variables occurs in the optimization model (*frac*). But we have chosen to expose the 'Share Percentage' and 'Share Revenue' decisions to the end-user, by setting the `CstrDecisions` and `CstrDecisionLabels` accordingly. The *user-defined constraints* defined on these decisions are converted using the actual decision variables from the mathematical model. In this example, the end user is also able to create segments (for example 'The Shares in the French Market') using the *segment editor* and apply constraints on these segments by setting the `CstrSegment` array through the *user-defined constraint editor*. To generate the constraints, the Mosel model will have to lookup the coefficients of the constraints for the elements that belong to the constrained segment.

In the following example (file *folio.mos* in the example *folio*) we will show how to configure the user-defined constraint editor to allow adding new constraints to the portfolio optimization model. The first step is to describe what decisions will be exposed to the end-user. This needs to be done along with the data initializations.

```
! Configure the constraint editor
CstrDecisions := {'x_per', 'x_rev'}           ! Expose two decisions
CstrDecisionDescr('x_per') := 'Share Percentage'
CstrDecisionDescr('x_rev') := 'Share Revenue'
```

It is then necessary to implement the logic how to convert the exposed decision to the decision variables used in the mathematical model:

```
declarations
  Wgt: dynamic array(xset: set of mpvar) of real  ! Constraint coefficients
  allfracs: set of mpvar                          ! Set of decision vars.
end-declarations

! All variables are used when no segment is defined
allfracs := union(share in SHARES) {frac(share)} ! Use all variables by default

forall(c in Cstrs) do                               ! Iterate over each constraints
  delcell(Wgt) ! Clean up coefficients

  if (CstrDecision(c)='x_per') then                  ! The share percentage (e.g. frac)
    if (exists(CstrSegment(c))) then
      forall(share in segget(CstrSegment(c))) Wgt(frac(share)) := 1
```

```

else
  forall(share in SHARES) Wgt(frac(share)) := 1
end-if
elif (CstrDecision(c)='x_rev') then          ! The share revenue (e.g. RET*frac)
  if (exists(CstrSegment(c))) then
    forall(share in segget(CstrSegment(c))) Wgt(frac(share)) := RET(share)
  else
    forall(share in SHARES) Wgt(frac(share)) := RET(share)
  end-if
end-if
end-if

if (Wgt.size>0) then                          ! Create the constraints
  dummy := ctrcreate(c,Wgt,CstrRHSVal(c))
end-if
end-do

```

The level of indirection provided through the concept of *decisions* adds business value to the *constraint editor*, and at the same time it improves the ease of maintenance of the application by allowing the Mosel developer to change the optimization variables, without any need for modifications to the user interface.

Examples of *user-defined constraint editors* are shown in Figures 2.3 and 2.4. In the first example, the user is adding a limit to the discounted price that can be applied in the TORONTO region. The second example presents more complex constraints, combining different types of aggregators (sum, forall, average). These aggregators are stored in the array `CstrAggregator` and are automatically taken into account during the call to the family of routines `ctrcreate*`.

Constraints

The following table allows you to view and edit the solution constraints:

25 records per page Search:

#	Description	Aggregator	Decision	Segment	Product	Operator	Attribute	RHS Value
2	TORONTO discounted price limit	Average	Discounted Price	TORONTO	F4YC	<=		1.5000

⊕ ⊖ ← Previous 1 Next →

Figure 2.3: Constraint Editor for Segments (example 1)

Constraint Editor

Cstrs	CstrDescr	CstrEnable	CstrAggregator	CstrDecision	CstrSegment	CstrOperator	CstrRHSVal
1	Maximum investment per share is 30% of total budget	true	For all	Share Percentage	All shares in database	<=	0.3
2	Limit the ratio of high-risk shares to 33%	true	Sum	Share Percentage	High-Risk Shares	<=	0.33
3	Force investment in French market to 10% of total budget	true	Sum	Share Percentage	French	>=	0.1
4	Force average revenue per share in United States market to 5 units	true	Average	Share Revenue	United States	>=	5.0

⊕ ⊖

Figure 2.4: Constraint Editor for Segments (example 2)

The two examples can be created by adding a suitably configured VDL autotable to an Optimization Modeler application. The file `constraint-test.vdl` in the folder `src/test/web/segment-editor` shows how to create such a constraint editor.

2.2.5.4 Segment Group Editor

Segment groups are used to logically organize segments. The segment group property definition (*SegGrpProp*) constrains how segments are grouped together. The current implementation supports three different constraints:

- *exclusive-segment*: prevents segments from being members of several different groups,
- *forbid-overlap*: segments forming a group must not contain the same element twice,
- *must-cover-all*: a group must be composed of segments that cover all elements of the domain).

Segment groups can be created and edited via the segment group editor components of SDK. A typical segment group editor view is shown in the screenshot in graphic 2.5.

Figure 2.5: Segment Group Editor

The upper part ('Segment Group Definition') of this view is formed by an autotable that is customized for editing segment groups (note that all Mosel arrays and sets used in this table are defined by *sdksegment*):

```
<vdl-section heading="Segment Group Definition" heading-level="3">
  <vdl-row>
    <vdl-table add-remove-row="addrow-autoinc" column-filter="true"
      modifier="=sdk.modifiers.segmentGroupDefinition">
      <vdl-table-column set="SegGroups" class="hide"></vdl-table-column>
      <vdl-table-column entity="SegGrpDescr" class="index" editable="true"
        heading="Description"></vdl-table-column>
      <vdl-table-column entity="SegGrpProp" editable="true" editor-type="checkbox"
        vdl-repeat="a in scenario.entities.SegGrpProperties" heading="a.label">
        <vdl-index-filter set="SegGrpProperties" value="a.value"></vdl-index-filter>
      </vdl-table-column>
      <vdl-table-column entity="SegGrpSize" headin="Size"></vdl-table-column>
      <vdl-table-column entity="SegGrpEnable" editable="true" editor-type="checkbox"
        heading="Enable"></vdl-table-column>
    </vdl-table>
  </vdl-row>
</vdl-section>
```

Once a segment group is enabled for editing by checking the box 'Enable', it will appear in the dropdown menu under 'Segment Group Mapping' that lets the user select which group to edit in the lower portion of the view. After selecting a segment group to be edited, the available segment definitions appear on the left, and the selected segments appear on the right of the *segment group editor* section that is defined by the following VDL code. The buttons in the center serve for selecting and deselecting segments for defining this group.

```
<sdk-segment-group-editor>
  <vdl-section heading="Segment Group Mapping" heading-level="3">
    <vdl-row>
      <vdl-column size="2">
        <label class="groupSelectedLabel">Select a group</label>
      </vdl-column>
      <vdl-column size="2">
        <select id="groupSelected" class="group-selected"></select>
      </vdl-column>
    </vdl-row>
  </vdl-section>

  <vdl-section class="segment-group" heading="Segments" heading-level="3">
    <vdl-row>
      <vdl-column size="5">
        <h5>Segments</h5>
        <table id="segmentsTable" column-filter="true"></table>
      </vdl-column>

      <vdl-column class="buttonPanel" size="1" style="padding-top:200px;">
        <button id="addToGroup" class="addToGroup btn btn-primary">&gt;</button><p/>
        <button id="removeFromGroup" class="btn btn-primary">&lt;</button>
      </vdl-column>

      <vdl-column size="5">
        <h5 class="group-title">Segments in Group: </h5>
        <table id="segmentsInGroupTable" column-filter="true"></table>
      </vdl-column>
    </vdl-row>
  </vdl-section>
</sdk-segment-group-editor>
```

The constraints on the segment group definition that result from the segment group properties are not enforced in the graphical interface. They require the execution of a Mosel routine, namely the procedure `seggrpcheck` of `sdksegment`. During the execution of `seggrpcheck`, the properties of each segment group are assessed. If any property constraints are not satisfied suitable error messages are reported to the user. A call to `seggrpcheck` also populates the count of unique group elements in the array `SegGrpSize`. Furthermore, it concatenates the segment filters into entries of the array `SegGrpFilter` which can then be used to consistently check whether the segment group state is aligned with the current segment files.

The execution of the routine `segupdate` also triggers a call to `seggrpcheck`.

The following example shows how to define segment groups ('Group 1' and 'Group 2') directly in a Mosel model and follows up with a check of the segment group properties and display of the filter definition for each group.

```
declarations
  SEG_TIER1=10
  SEG_TIER2=11
  SEG_TIER3=12
  SEG_TIER3OVER=13
  SEG_TIER0=14
  msg: text
end-declarations

! Segment definition
segment(SEG_TIER1,"Customers","Tier1")
```

```

segaddcondition(SEG_TIER1,"age","<=",18)

segment(SEG_TIER2,"Customers","Tier2")
segaddcondition(SEG_TIER2,"age",">",18)
segaddcondition(SEG_TIER2,"age","<=",45)

segment(SEG_TIER3,"Customers","Tier3")
segaddcondition(SEG_TIER3,"age",">",45)

segment(SEG_TIER3OVER,"Customers","Tier3err")
segaddcondition(SEG_TIER3OVER,"age",">",40)

segment(SEG_TIER0,"Customers","Tier0")
segaddcondition(SEG_TIER0,"age",">=",0)

! Group definition
SegGrpDescr(1) := "Group 1"
seggrpset(1,{SEG_GRP_PROP_FORBIDOVERLAP})
seggrpadd(1,{SEG_TIER1,SEG_TIER2,SEG_TIER3,SEG_TIER3OVER})

SegGrpDescr(2) := "Group 2"
seggrpset(2,{SEG_GRP_PROP_MUSTCOVERALL})
seggrpadd(2,{SEG_TIER1,SEG_TIER2,SEG_TIER3})
SegGrpMap(2,SEG_TIER1) := true
SegGrpMap(2,SEG_TIER2) := true
SegGrpMap(2,SEG_TIER3) := true

! Check segment group constraints
seggrpcheck
forall(g in SegGroups, p in SegGrpProperties)
    writeln("Error in group '", seggrptext(g), "' : ", SegGrpPropInfo(g,p))

! Display filters of the segments contained in the groups
forall(g in SegGroups)
    writeln("Group '", seggrptext(g), "' filter: ", SegGrpFilter(g))

```

2.2.5.5 Tableau Integration

The package *sdksegment* provides tools to break down the results of a model run per segment. This methodology allows UI users to identify new segments of customers that provide a high revenue, or it can also be used to debug a model by dissecting the results into smaller parts.

User-defined segments can be made available to the Tableau software by two different means. The easiest and most memory intensive method consists in creating an array that associates segments and elements. This array can be created with a call to *seggetmap*. However, if the number of elements that are manipulated by *sdksegment* is huge, then the model developer will have to do the metrics aggregation per segment.

The graph in Figure 2.6 is taken from a Marketing Campaign optimization application. It shows the ratio between the number of contacts and the yield per customer segment and per channel. This simple graph visually demonstrates how effective is the channel 'Telephone' for this marketing campaign. It also clearly identifies the 'New Tier' segment as an important source of yield.

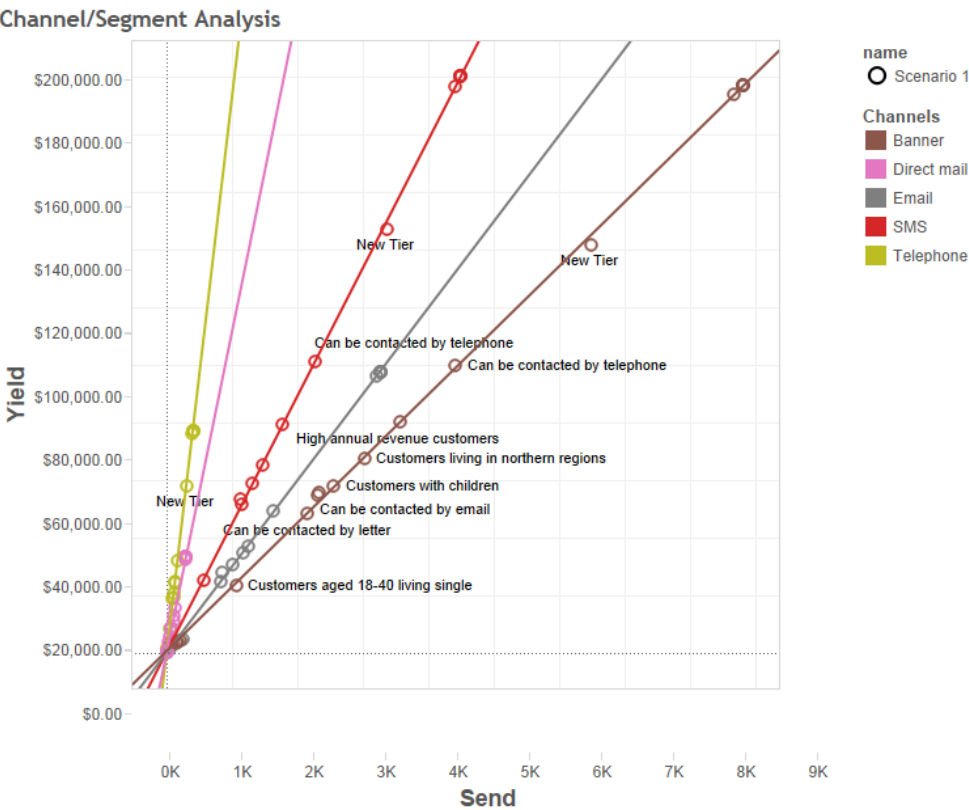


Figure 2.6: Data Segmentation in Tableau

The graph in Figure 2.7 was created by using the segment name as a geographical position information. In this example each segment represents a group of customers living in one of the three states. The segment name is simply set to the state, and this information is used in Tableau for drawing the map.

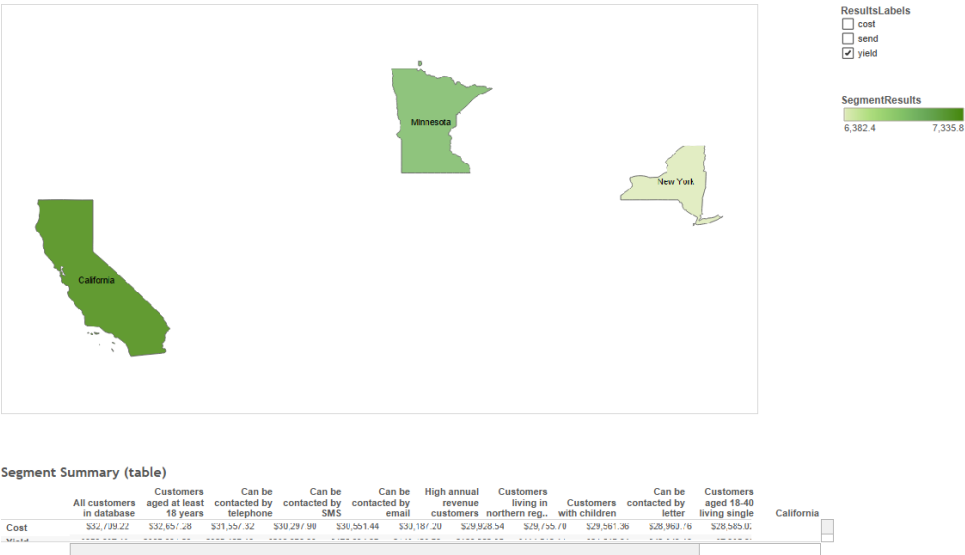


Figure 2.7: Geographical Segmentation

2.2.6 Advanced

Readers who are interested in more detailed and advanced examples are referred to the various test models that accompany the SDK package. These examples can be found in the development version of the SDK distribution in the folder *src/test/mosel*. Examples of web views can also be found in *src/test/web*. These files can be used as quickstart examples to familiarize with the functionality provided by *sdksegment*.

The following list summarizes the features used in the various tests:

- `testgap.mos`: Using data segmentation when solving a Generalized Assignment Problem.
- `testgenericcstr.mos`: Using data segmentation and creating generic constraints over the segments.
- `testlookup.mos`: Searching for segments based on a particular attribute and retrieving segments containing a particular element.
- `testmultiset.mos`: Managing attributes that are defined for different sets of elements.
- `testrdbms.mos`: Using database operations.
- `testattr.mos`: Attributes, segment groups management and segment definition archiving.

2.3 Advanced concepts

This section provides implementation details and an advanced description of the design of the *sdksegment* package. Readers who are not interested in contributing to the development of the package can skip this section.

2.3.1 Configuration

2.3.1.1 Memory and database

The package has been designed to support in memory as well as in database operations. When contributing new functionality it is important to make sure that it provides consistent results for the two modes.

For example, when the connection to a database is alive, then by default all segment filtering is done directly in the database. However, when the database connection has been dropped, the default operation mode switches to in-memory lookup.

2.3.1.2 Cache management

In order to improve lookup performance, all the `segget` routine results are cached in memory. The whole cache is invalidated if the attribute value of an element is changed through a call to `segsetattr`. The database operations never use the cache to render results, but always fill it with the results of the SQL queries. Database operations cannot make use of the cache because it is impossible to control when the attribute value changes.

For this reason, the attribute values cannot be accessed directly by the developer. The various routines `seggetattr` can be used to read the values, and `segsetattr` to set the values. The current implementation does not allow the use of `segsetattr` in database mode.

2.3.1.3 Types

The `sdksegment` package can convert certain external types (database field types) to Mosel types. This conversion is required for the various checks that are carried out when creating the segment filters. Attributes with unknown external types will not be taken into account by the package.

The current implementation supports the following list of attribute types which should be sufficient for working with most databases.

```
SegAttrTypes = { 'boolean',  
  'int', 'integer',  
  'real', 'double', 'float',  
  'varchar', 'char', 'string',  
  'date', 'datetime' }
```

Each attribute type is mapped to a Mosel type. This Mosel type is then used to check the validity of the filters. The following list are the currently defined mappings between attribute types and the Mosel types:

```
MoselType('int') := 'integer'  
MoselType('integer') := 'integer'  
MoselType('real') := 'real'  
MoselType('double') := 'real'  
MoselType('float') := 'real'  
MoselType('date') := 'datetime'  
MoselType('datetime') := 'datetime'  
MoselType('boolean') := 'boolean'  
MoselType('string') := 'text'  
MoselType('char') := 'text'  
MoselType('varchar') := 'text'
```

Developers can add further type conversions by extending the set `SegAttrTypes`, and adding the mapping to the corresponding Mosel type.

2.3.1.4 Operators

The *sdksegment* package supports a finite number of test operations on the attribute values. The compatibility between these operations and the corresponding Mosel type should also be defined.

The current implementation supports the following operators:

```
SegSegmentFilterOpSymbols:: (1..8) ["<", " ≤ ", ">", " ≥ ",
    "=", "<>", " in ", " not in "]
```

The operators '*in*' and '*not in*' only apply to text elements. They can be used to check if a particular text element is part of a list of text that is given as a comma separated list of text.

The unique ID of an operator can be retrieved using the array `SegSegmentFilterOpId`.

Developers who want to extend the list of operators will need to:

- add the new operator to the `SegSegmentFilterOpSymbols`, and update `SegSegmentFilterOpId`
- define compatibility with Mosel types through `SegSegmentFilterOpTypes`
- update `segmenttest` and `segfilter` accordingly.

2.3.1.5 Hiding attributes

The *sdksegment* package supports the hiding of attributes from the graphical user interface.

The *SegAttrHide* array is used to determine which attributes should be hidden (*i.e.* excluded) from the dropdown list of the segment editor. An attribute marked as hidden will not be visible in this dropdown list. If the attribute is already part of a condition, then it becomes disabled and the only action that can be performed by the user is to remove it from the condition list.

In order to prevent accidental changes by a non-authorized user, the array *SegAttrHide* is annotated as 'always hidden' in the package source. It may be useful to create a dedicated view that presents *SegAttrHide* in an *autotable* in order to allow a user with an administrator role to select which attributes can be shown.

The screenshot in Figure 2.8 shows a segment editor for which the 'country' attribute is hidden.

The screenshot shows a graphical user interface for a segment filter. At the top, there is a header bar with a dropdown menu set to 'United Kingdom' and a 'not evaluated' status indicator. Below the header, there is a list of conditions. The first condition is 'Country = UK' and the second is 'Risk ≤ 25'. Each condition has a red 'x' icon to its left and a question mark to its right. At the bottom of the conditions list, there is a button labeled 'NEW CONDITION'.

Figure 2.8: Segment Filter - Hidding Attributes

2.3.1.6 Automated sampling of attribute values

The *sdksegment* package allows to sample the value of existing attributes. These values can then be presented in the graphical user interface to help the user fill the conditions.

The screenshot in Figure 2.9 is taken from the 'folio' example, it shows a list of suggestions for the 'category' attribute.

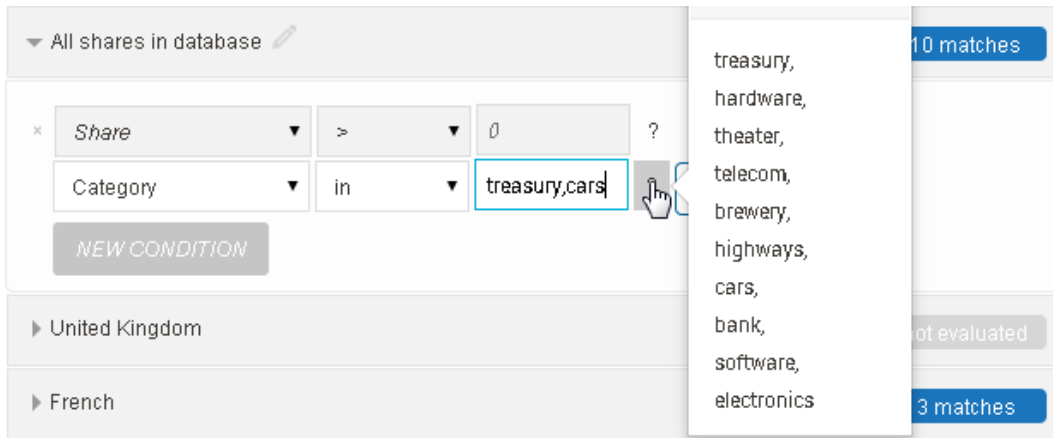


Figure 2.9: Segment Filter - Attribute Value Suggestions

This list can be obtained by calling the sampling function directly for an attribute (using *segsample*) or by setting the 'autosample' parameter and executing any *segload** function.

2.3.1.7 Multiple Domains

The *sdksegment* package can be used to manage multiple sets of attributes.

For example a set of attributes related to customers and second set related to offers.

In order to configure the segment editor component to work with multiple attribute domains, the attribute 'domain' of the component must be set to the name of the attribute domain used in the model.

In the following extract of VDL views, we create two segment editors: one for the domain "Customers" and one for the domain "Activities".

```
<vdl version="3.2">
  <page>
    ...
    <vdl-row>
      <vdl-column heading="Share Segments">
        <sdk-segment-editor global-filter-entity="SelectedCountry" domain="shares">
          </sdk-segment-editor>
        </vdl-column>
      </vdl-row>

      <vdl-row>
        <vdl-column heading="Stock Exchange Segments">
          <sdk-segment-editor global-filter-entity="SelectedCountry" domain="stockexchanges">
            </sdk-segment-editor>
          </vdl-column>
        </vdl-row>
        ...
      </page>
```


</vdl>

Figure 2.10 shows two segment editors on the same page view: one for the shares and another one for the regions. These two domains serve for aggregating results over regions or share types. This example is taken from the 'folio' application that comes as a part of the SDK distribution. **Note:** Although this is not shown here, the 'folio' example could quite easily be extended to offer the user the possibility to constrain his portfolio to a particular region and share type.

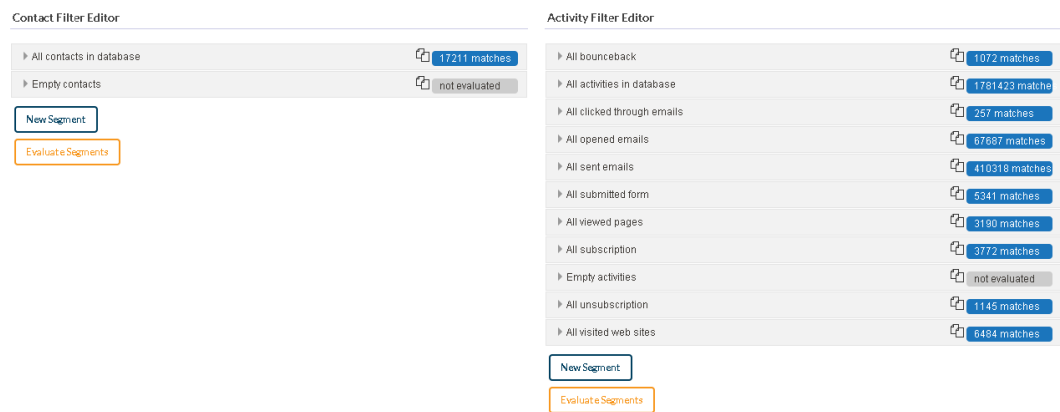


Figure 2.10: Segment Filter - Multiple Domain

CHAPTER 3

Utilities

3.1 Overview

The SDK Utilities module provides a collection of toolbox-style functionality for the solution developer. In their majority these are Mosel routines encapsulated in the *sdkutil* package.

The message and error handling system also has a user interface component for displaying messages originating from a model in Optimization Modeler views and there is an accompanying JavaScript library.

The history system is equally implemented by the Mosel package *sdkutil* but comes with its own interface component that is documented in a separate chapter (see [Chapter 4](#)).

The module functionality falls into different categories:

- Message and error handling
- Notes logging
- Timers
- String and text handling utilities
- File access functionality
- Rounding functions
- Bit handling routines
- Package configuration options

Purpose and usage of text handling and maths utilities are straightforward and the reader is referred to the short examples provided with the subroutine documentation. The functionality for handling messages, errors, and exit codes is explained by means of examples in the following section. Please see [Chapter 4](#) about how to work with the logging of notes in the history system.

3.2 Usage examples

This section documents several standard usage examples of the Utilities module.

- [Section 3.2.1](#) shows how to integrate the Solutions SDK into your application to add the messaging system to the web application views.
- [Section 3.2.2](#) demonstrates how to use the message handling facilities and standard application logging routines.

- Section 3.2.3 presents the message dictionary functionality that can be used for documentation purposes or as a localization framework to support alternative languages in the messaging system.
- Section 3.2.4 describes the use of routines that are used to measure the computation time of portions of Mosel model code.

3.2.1 Integrating the SDK with VDL views

To include the SDK into a VDL page, you must include the CSS file for the SDK (`sdk.css`) and the SDK Javascript library (`sdk.min.js`) in the client-resources `css` and `js` folders respectively. You then include them into a VDL page by including the `sdk.vdl` page as follows:

```
<vdl-include src="sdk.vdl"></vdl-include>
```

The presence of any `sdk-messages` tag component will add in the code that is required to display error, warning, and info messages generated by the model.

3.2.2 Message handling

3.2.2.1 Logging routines

The message logging routines `logging`, `infolog`, `warninglog`, and `errorlog` print formatted messages to the standard output and also collect message texts for reporting purposes, e.g. to be displayed in Optimization Modeler VDL or HTML views. Stored message texts are truncated after a maximum size has been reached (the size limit is specified by the parameters `UTIL_MAXLOGSIZE`, `UTIL_MAXWARNLOGSIZE`, and `UTIL_MAXERRLOGSIZE` respectively).

The appearance of the logging format can be changed with the parameter `UTIL_LOGFORMAT` (value 0: message only, value 1: message preceded by time measurement—default, 2: logging format also includes date and node name—recommended for distributed applications).

Please note that the procedure `errorlog` does not interrupt the model execution. In the case of errors, the model needs to be terminated with a call to `exit`, several overloaded versions of `exit` that ensure a standard exit behavior including an update of the logging messages are defined by `sdksutil`. The package also defines several exit codes (constants `UTIL_RET_*`).

The amount of debug output printed by `logging` depends on the setting of the value `UtilVerbose`—this scalar is initialized with the value of the package constant `UTIL_VERBOSE` when first loading the package, its value can be modified during model execution by calling the `utilsetparam` routine with the 'verbose' setting (note that a separate parameter, 'sdkverbose', initialized via the package constant `UTIL_INTVERBOSE` serves to control the level of message output generated internally by the SDK.)

The Mosel code shown below is an extract of the example file `testmessages.mos` that is included with the test examples of SDK development builds.

```
utilclearlogs                                ! Clean up logs from previous runs

utilsetparam("verbose", 4)                   ! Set application verbosity level

setparam("ioctrl", true)                     ! I/O error handling by the model
initializations from DATAFILE
MYVAL
end-initializations

if getparam("iostatus")>0 then                 ! Exit with error message
  exit(UTIL_RET_NODATA, _("Error reading input data file."))
else
  infolog(_("Reading input data succeeded.))    ! Display status message
```

```

        logging(9, formattext(_("Data read: %d"), MYVAL))
                                ! Debug output (if verbose>=9)
    end-if
    setparam("ioctl", false)

    if not (or(g in GOALS) ifGoal(g)) then      ! Check some input data values
        warninglog(_("No goals selected for optimization."))      ! Print a warning
    end-if
    if not check_non_negativity(MYVAL) then
        errorlog(2, formattext(_("Value %d is negative."), MYVAL)) ! Log an error
    end-if
    ...
    if utilerror then
        exit(UTIL_RET_ERROR, ErrorMessage)      ! Display all logged errors + exit
    end-if
    ...
    infolog(_("Optimization terminated."))      ! Display another status message

    exit                                          ! Clean exit with error management

```

Mosel models using the message handling system of *sdkutil* should be terminated by a call to one of the overloaded versions of `exit` defined by *sdkutil* or alternatively, call `logcleanup` in order for the message stack to remain consistent.

3.2.2.2 Messaging notification in the Optimization Modeler UI

The SDK includes a basic *messaging notification system*, allowing Optimization Modeler VDL views to display different types of notifications to the user to show the status of a model execution.

The relevant Mosel entities (`MsgLog`, `MsgLogLevel`, `MsgLogStatus`) that serve for capturing error, warning and info messages and communicating them to Optimization Modeler are readily configured with suitable `om` annotations in the SDK package *sdkutil*. The '`msgverbose`' setting of the `utilsetparam` routine selects which message types are to be stored for display in OM.

To include the messaging module into a VDL page, you must include the VDL tag `sdk-messages` (see file `sdk-messaging.vdl` of the example project *sdktest*).

```

<vdl version="3.2">
  <vdl-include src="sdk.vdl"></vdl-include>
  <vdl-page>
    <sdk-toolbar><sdk-messages></sdk-messages></sdk-toolbar>
    ...
  </vdl-page>
</vdl>

```

Optionally, the property `timeout` of the tag `sdk-messages` can be defined with the message display duration in seconds (default: 25 seconds), and the setting `timeout="-1"` indicates that messages need to be clicked on to terminate their display. Using the attributes `error-enabled`, `warning-enabled`, and `info-enabled` the developer can configure which type(s) of messages are to be displayed (by default all message types are being displayed). Several attributes of `sdk-messages` can be defined jointly, for example to show only error messages using a timeout of 20 seconds:

```

<sdk-messages warning-enabled="false" info-enabled="false" timeout="20"></sdk-messages>

```

The example *sdktest* that is provided with the SDK distribution includes a simple view that lets the user select a message type to see its appearance in the notification system. The following VDL view definition (file `sdk-messaging.vdl`) produces the view shown in Figure 3.1.

```

<vdl version="3.2">

```

```

<vdl-include src="sdk.vdl"></vdl-include>
<vdl-page>
  <sdk-toolbar>
    <sdk-messages error-enabled="true" info-enabled="true" warning-enabled="true">
    </sdk-messages>
  </sdk-toolbar>
  <vdl-row>
    <vdl-column size="5">
      <vdl-form>
        <vdl-field label-size="4" size="5" entity="SelMsgType"
          label="Select message type:" options-set="MsgTypes" type="radio"></vdl-field>
        <vdl-execute-button caption="Generate messages"></vdl-execute-button>
      </vdl-form>
    </vdl-column>
  </vdl-row>
</vdl-page>
</vdl>

```

The Mosel entities used by this view are configured with the following `om` annotations in the model file `sdktest.mos`:

```

!@om.manage input
public declarations
  !@om.alias MessageTypes
  !@om.transform.labels.entity MsgTypeLabel
  MsgTypes: set of integer           ! Message codes
  !@om.hidden true
  MsgTypeLabel: array(MsgTypes) of string ! Display text for message codes
  !@om.update.afterexecution
  SelMsgType: integer                ! Selector for UI
end-declarations

```

The XML companion file `sdktest.xml` of this example has the corresponding entry to include the view:

```

<client>
  <view-group title="Main">
    <vdl-view title="Messaging" default="true" path="sdk-messaging.vdl"
      requires-scenarios-loaded="true"/>
  </view-group>
</client>

```

Remark: Please note that logging messages are only reported at the *Run Log* tab, not in the VDL view itself.

The messages are displayed in the graphical user interface using boxes. These boxes will disappear after a configurable time.

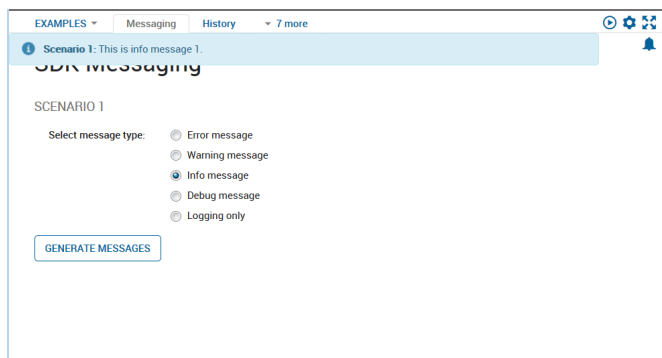


Figure 3.1: VDL view of the 'sdktest' messaging example

The list of messages can be retrieved by clicking on the top right bell icon.

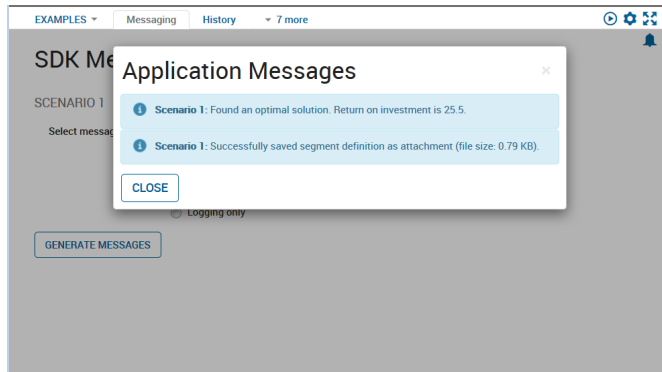


Figure 3.2: List of messages from the 'sdktest' example

Remark: Please note that if the model configures the results data to be deleted on-change, *i.e.* if it defines the annotation `!@om.resultdata.delete=on-change`, then when the message is dismissed either by clicking it or if it disappears after the configured time, the results will be deleted.

3.2.3 Error message dictionary

Within an application, it is possible to build up a lookup table that associates error codes with message texts. When systematically registering application (error) messages via such a table, a list of application errors (also referred to as *message dictionary*) for documentation purposes is immediately available (`utilwritedict`).

Predefined error codes of *sdkutil* include `UTIL_ERR_OR` (no error) and `UTIL_ERR_NOIMPL` (functionality not implemented), other codes (integer values different from 0) can be defined within an application using `utilsetdictentry` (user selects code value) or `utiladddictentry` (function returns an automatically selected code value). Predefined and application messages can be retrieved via their code number using `utilgetdictentry`. We recommend to prefix error or message code constants defined by a model with some application-specific characters. Furthermore, it is good practice to mark up such message texts with Mosel translation markers for a possible later use with the XPRNLS functionality.

The following Mosel code is an extract of the example file `testmessages.mos` (located under the `test` subdirectory of SDK development builds).

```

declarations
  MY_MSG_DATAERR, MY_MSG_OPTEND, MY_MSG_NOGOAL,
  MY_MSG_SHOWDATA, MY_MSG_DATASUCC: integer ! Automatic code value choice
  MY_MSG_NEG = 2 ! User-specified message code
end-declarations

MY_MSG_DATAERR := utiladddictentry(_("Error reading input data file.))
MY_MSG_DATASUCC := utiladddictentry(_("Reading input data succeeded.))
MY_MSG_SHOWDATA := utiladddictentry(_("Data read: "))
MY_MSG_NOGOAL := utiladddictentry(_("No goals selected for optimization.))
MY_MSG_OPTEND := utiladddictentry(_("Optimization terminated.))
utilsetdictentry(UTIL_MSG_ERR, MY_MSG_NEG, _("Negative value for: "))

utilwritedict("msgcodes.txt") ! Write all messages to a file
utilwritedict("errcodes.txt", UTIL_MSG_ERR) ! Write error messages to a file
...
if getparam("iostatus") <> 0 then ! Exit with error message
  exit(UTIL_RET_NODATA, utilgetdictentry(MY_MSG_DATAERR))

```

```
else
  infolog(utilgetdictentry(MY_MSG_DATASUCC))          ! Display status message
  logging(9, utilgetdictentry(MY_MSG_SHOWDATA)+ MYVAL) ! Debug output (if verbose>=9)
end-if
setparam("ioctl", false)

if not (or(g in GOALS) ifGoal(g)) then                ! Check some input data values
  warninglog(utilgetdictentry(MY_MSG_NOGOAL))         ! Print a warning
end-if
if not check_non_negativity(MYVAL) then
  errorlog(MY_MSG_NEG, utilgetdictentry(MY_MSG_NEG) + MYVAL) ! Log an error
end-if
...
if utilerror then
  exit(UTIL_RET_ERROR, ErrorMessage)                 ! Exit and display logged errors
end-if
...
infolog(utilgetdictentry(MY_MSG_OPTEND))              ! Display status message
```

3.2.4 Timers

The 'timer' functionality defined by *sdkutil* makes it possible to work with multiple time measures in a model. Each timer is identified by its name (a string indicated when starting the timer). The following example shows a typical calling sequence of timer functions in a Mosel model.

```
utiltimerstart("ReadData")          ! Start the 'ReadData' timer
utiltimerstart("Total")              ! Start the 'Total' timer

initializations from DATAFILE
...
end-initializations

utiltimerstop("ReadData")            ! Stop the 'ReadData' timer

! ... do something else

utiltimercont("ReadData")            ! Continue measurement with 'ReadData' timer

initializations from DATAFILE2
...
end-initializations

utiltimerdisplay                     ! Display measurements of all defined timers

writeln_("Time for reading data: ", utilgetelapsed("ReadData"))
writeln_("Total elapsed time: ", utilgetelapsed("Total"))
```

For a complete version of this model, please see the example file `testtimers.mos` that is included with the test examples of SDK development builds.

CHAPTER 4

History

4.1 Overview

The SDK History module provides a *Notes history* feature, through which users can add notes to scenarios as they are working on it. Using this module, the solution developer can add a *Notes Viewer* to a VDL page where users can view and add notes to the scenario. The Notes Viewer can be embedded in any VDL view by using a specific VDL tag. The solution developer can also use this module to add notes directly from the model during execution. In addition to the title and the content of the note, the module also records the username of the user adding the note and the time when the note was added. The corresponding Mosel routines are part of the *sdkutil* package.

Unlike *messages* that are handled by the message notification system (see Section 3.2.2.2) and most often are meant for temporary display only, *notes* are persisted by the application and the UI user cannot delete any entries from the notes history.

4.2 Usage examples

To use the History module in a VDL page, there are three steps that you need to follow.

1. Include the `sdk.min.js` and `sdk.css` files in the `js` and `css` subfolders of `client-resources` respectively
2. Include the `sdk.vdl` page in the `client-resources` folder
3. Using a `vd1-include` tag, pull in the `sdk.vdl` file to your VDL page
4. Add the `sdk-history` VDL tag to the VDL page

First, you need to include the SDK in your VDL page.

```
<vd1-include src="sdk.vdl"></vd1-include>
```

Next, you need to add the `sdk-history` tag to the VDL page. This tag will create an icon in the view to open the *Note Viewer* through which the user can view and add notes to the scenario. The following VDL code extract shows how to add a Notes History to a VDL page, and Figure 4.1 is a screenshot of the resulting Note Viewer.

```
<vd1 version="3.2">
  <vd1-page class="compact">
    <sdk-history></sdk-history>
  <vd1-section heading="My Form">
```



```

    ...
  </vdl-section>
</vdl-page>
</vdl>

```

The history icon will be placed on the page in a position corresponding to where the tag has been placed in the markup. To replicate the previous behaviour of the icon appearing in the top right hand corner of the page, add it to the new `sdk-toolbar` tag.

```

<vdl version="3.2">
  <vdl-page class="compact">
    <sdk-toolbar><sdk-history></sdk-history></sdk-toolbar>
    <vdl-section heading="My Form">
      ...
    </vdl-section>
  </vdl-page>
</vdl>

```

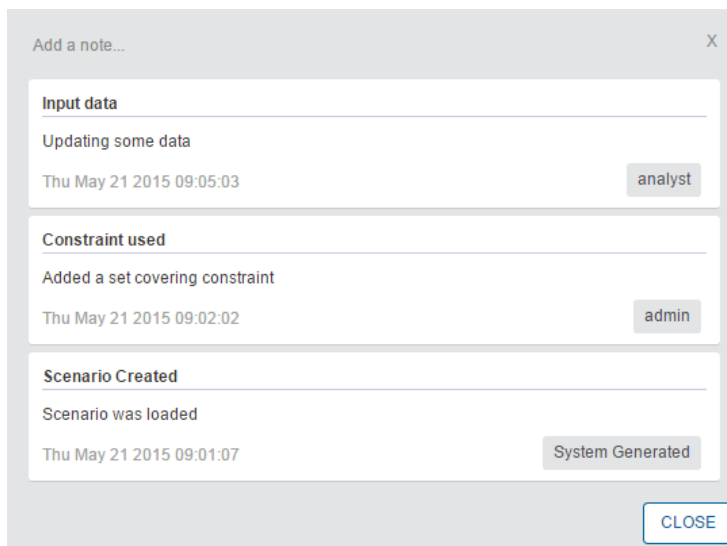


Figure 4.1: Note Viewer

The maximum character limit of a note is 2048 characters. For a note that is long, only two lines will be displayed in the Note Viewer as shown in Figure 4.2. To view the whole note, the user will have to click the `click to read more` link—the result of this action can be seen in Figure 4.3. Also worth mentioning, a note will always be formatted in plain text formatting, any HTML tags included in the note will be displayed as text.

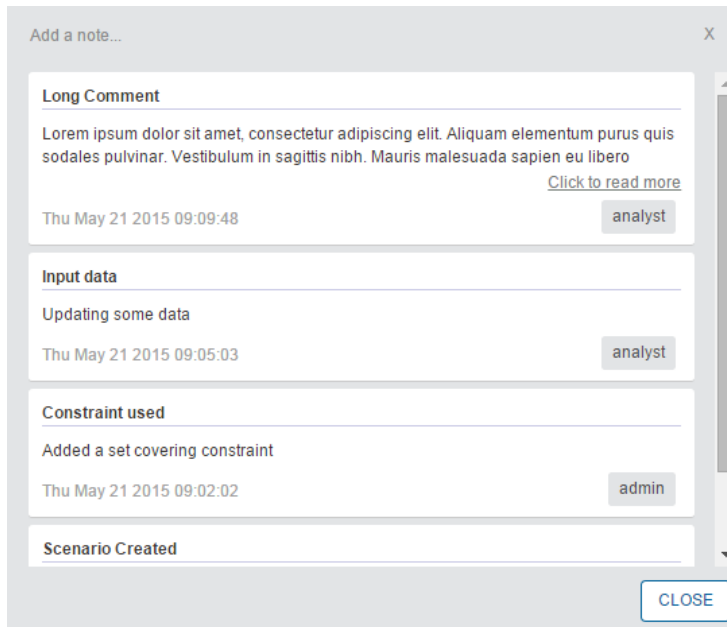


Figure 4.2: Truncated Note

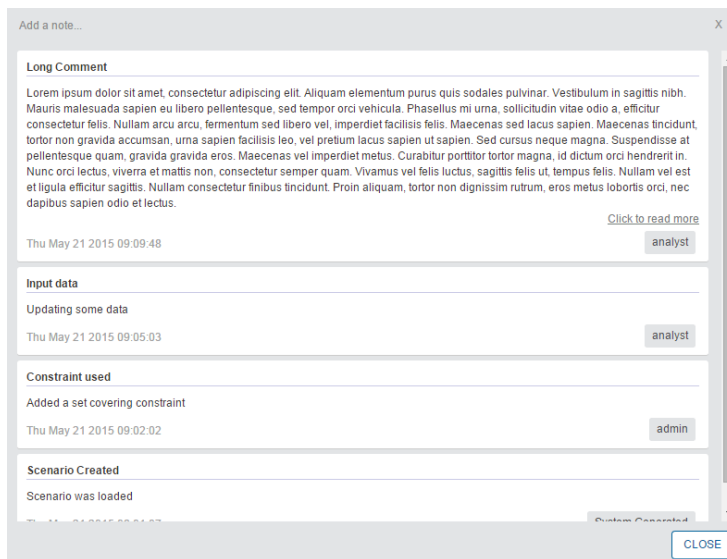


Figure 4.3: Expanded Note

The solution developer may choose to add notes during the execution of the model. The username for all model created notes will be `System Generated`. The decision of which pieces of information generated by model runs are to be saved into the notes history needs to be made carefully, keeping in mind that this history is stored with the scenario and does not get emptied as long as the scenario exists.

```
model "conditionals"
  uses "sdkutil"
  ...
  ! Add a note
```

```

        notelog("Scenario Run", "The scenario was successfully run" )
        ...
    end-model

```

As is shown in the following VDL code, a vdl-table using the `Note` entity can be created to expose to the user a sortable and filterable view of all the notes (see screenshot of the resulting view in Figure 4.4).

```

<vdl version="3.2">
  <vdl-page>
    <vdl-section>
      <vdl-column heading="Notes" heading-level="5">
        <vdl-table page-mode="paged" page-size="50" column-filter="true">
          <vdl-table-column set="NoteIds" width="30">Notes</vdl-table-column>
          <vdl-table-column entity="Note"
            vdl-repeat="=m in scenario.entities.UtilNoteAttrs"
            heading="=scenario.entities.UtilAttrLabel(m.value).label">
            <vdl-index-filter set="UtilAttrs" value="=''+m.value"></vdl-index-filter>
          </vdl-table-column>
        </vdl-table>
      </vdl-column>
    </vdl-section>
  </vdl-page>
</vdl>

```

We can sort the notes such that the latest note appears at the top by adding a *VDL modifier* to this autotable as demonstrated below in the Javascript code added to the VDL page.

```

<vdl version="3.2">
  <script>
    function descSort(tableOptions) {
      tableOptions.overrides.aaSorting = [[0, 'desc']];
      return tableOptions;
    }
  </script>
  <vdl-page>
    <vdl-section>
      <vdl-column heading="Notes" heading-level="5">
        <vdl-table page-mode="paged" page-size="50"
          column-filter="true" modifier="=descSort">
          ...
        </vdl-table>
      </vdl-column>
    </vdl-section>
  </vdl-page>
</vdl>

```

Notes

Show entries Search:

Notes	Title	Comment	Username	Timestamp
Notes	Title	Comment	Username	Timestamp
2	Scenario Run	The scenario was successfully run	System Generated	2015-06-23T17:02:38
1	Scenario Loaded	The scenario was loaded	System Generated	2015-06-23T17:02:36

Previous **1** Next

Figure 4.4: VDL Notes Autotable

Remark: Please note that if the model configures the results data to be deleted on-change, *i.e.* if it defines the annotation `!@om.resultdata.delete=on-change` then any new note that is added will cause the results to be deleted.

CHAPTER 5

Data Validation

5.1 Overview

The SDK Data Validation Rules module provides functionality through which users can define constraints on the data managed by the *sdksegment* module. Using the graphical components of this module, the solution developer can add a Data Validation Editor and a Data Validation Report to Optimization Modeler views by using VDL tags. The solution developer can also use this module to add data quality checks directly from the model.

5.2 Usage examples

To use the Data Validation Rules module in a VDL page, there are three steps that you need to follow.

1. Include the `sdk.min.js` and `sdk.css` files in the `js` and `css` subfolders of `client-resources` respectively
2. Include the `sdk.vdl` page in the `client-resources` folder
3. Using a `vd1-include` tag, pull in the `sdk.vdl` file to your VDL page
4. Add either the `sdk.modifiers.dataRuleValidationEditor` or the `sdk.modifiers.dataRuleValidationReport` as a modifier attribute on a `vd1-table` in your VDL page

The code examples and screenshots presented in this section are taken from the *folio* example which is part of the SDK distribution.

5.2.1 Setting up the page

First, you need to include the SDK in your VDL page.

```
<vd1-include src="sdk.vdl"></vd1-include>
```

Next, you need to add an `vd1-table` tag to the VDL page and add a modifier attribute set to either `sdk.modifiers.dataRuleValidationEditor` or `sdk.modifiers.dataRuleValidationReport` as appropriate. The modifier will create a table with triggers to handle the specific data structure and styling of the Data Validation Rules module.

5.2.2 Adding a validation report

The following example shows how to create a *Data Validation Report*. We set the *modifier* attribute of a *vd1-table* to `sdk.modifiers.dataRuleValidationReport` in order to create a table that summarizes the status of each individual data validation rule. By default, all satisfied rules are associated with a green dot and any violated ones are highlighted by a red dot. When a rule makes use of a *soft limit*, an orange dot will inform the end user about data violating the soft limit, but satisfying the hard limit. The grey dot is used when the rule is disabled.

```
<vd1 version="3.2">
  <vd1-page class="compact">
    ...
    <vd1-table modifier="sdk.modifiers.dataRuleValidationReport">
      <vd1-table-column set="Cstrs"><vd1-table-column>
        <vd1-table-column entity="CstrDescr" heading="Description"></vd1-table-column>
        <vd1-table-column entity="CstrEnable" heading="Enable"></vd1-table-column>
        <vd1-table-column entity="CstrSegment" heading="Segment"></vd1-table-column>
        <vd1-table-column entity="CstrAggregator" heading="Aggregator"></vd1-table-column>
        <vd1-table-column entity="CstrStatus" heading="Status"></vd1-table-column>
      </vd1-table>
    ...
  </vd1-page>
</vd1>
```

The graphical component resulting from the VDL code extract above is shown in Figure 5.1.





Cstrs▲	Description	Enable	Segment	Aggregator	Status
0	Risk cannot be negative	true	All shares in database	For all	
9	Risk is normal (high)	true	All shares in database	For all	
10	Risk is normal (low)	true	All shares in database	For all	
11	Average return on French market	false	French	Average	

Figure 5.1: Data Validation Rules Report

5.2.3 Adding a data validation editor

The example printed below creates a *Data Validation Editor* by setting the *modifier* attribute of a *vd1-table* to `sdk.modifiers.dataRuleValidationEditor`. The resulting table (see Figure 5.2) can be used to add new data validation rules in the Optimization Modeler UI.

```
<vd1 version="3.2">
  <vd1-page class="compact">
    ...
    <vd1-table modifier="sdk.modifiers.dataRuleValidationEditor">
      <vd1-table-column set="Cstrs"></vd1-table-column>
      <vd1-table-column entity="CstrDescr" heading="Description">
        </vd1-table-column>
      <vd1-table-column entity="CstrEnable" heading="Enable">
        </vd1-table-column>
      <vd1-table-column entity="CstrSegment" editor-type="select"
        editor-options-set="Segments" heading="Segment">
        </vd1-table-column>
      <vd1-table-column entity="CstrAggregator" editor-type="select"
        editor-options-set="CstrAggregators" heading="Aggregator">
        </vd1-table-column>
      <vd1-table-column entity="CstrCoefAttr" editor-type="select"
        editor-options-set="SegAttrs" heading="Attribute">
        </vd1-table-column>
      <vd1-table-column entity="CstrOperator" editor-type="select">
        </vd1-table-column>
    </vd1-table>
  </vd1-page>
</vd1>
```

```

        editor-options-set="CstrOperators" heading="Operator">
    </vdl-table-column>
    <vdl-table-column entity="CstrRHSVal" heading="Limit (hard)">
    </vdl-table-column>
    <vdl-table-column entity="CstrRHSValSoft" heading="Limit (soft)">
    </vdl-table-column>
    </vdl-table>
    ...
</vdl-page>
</vdl>

```

Cstrs ▲	Description	Enable	Segment	Aggregator	Attribute	Operator	Limit (hard)	Limit (soft)
0	Risk cannot be negative	true	All shares in database	For all	risk	>=	0.0	
9	Risk is normal (high)	true	All shares in database	For all	risk	>=	0.0	10.0
10	Risk is normal (low)	true	All shares in database	For all	risk	<=	30.0	27.0
11	Average return on French market	false	French	Average	ret	>=	1.0	

Figure 5.2: Data Validation Rule Editor

5.2.4 Validation rules in the Mosel model

The solution developer can also choose to state data validation rules directly in the Mosel model. The following snippet of Mosel code illustrates how to create a rule to check whether the values of the 'risk' attribute for all shares are non-negative. After stating the rule with a call to `datarule`, the rule is applied by calling `drupdate` and finally, a rules validation log is output with `drdisplay`.

```

model "testdatarule"
    uses "sdkdatarule"
    ...
    ! Create a default segment
    segaddcondition(SEG_ALL,'share','>',0)

    ! Create the datarule
    dummy := datarule(SEG_ALL,'risk','>=',0)

    ! Update data validation rule status
    drupdate

    ! Display rule status
    drdisplay

    ...
end-model

```

The data rule stated above uses a *hard limit*. An alternative form of the `datarule` function lets the users state both, hard and soft limits. If a *soft limit* is violated but the hard limit holds then the constraint is considered to be in range (constraint status `CSTR_STATUS_INRANGE`).

```
dummy := datarule(SEG_ALL,'risk','>=',0,0.1, "Hard and soft limit")
```

CHAPTER 6

File Browser

6.1 Overview

The VDL component 'filebrowser' lets the UI user choose a file or directory from a list of files provided by the model in XML format. The data from the selected file can be treated either as managed (accessed during scenario loading) or unmanaged (accessed during scenario run) data. In this chapter we show how to use a file browser in order to select an input data file—notice however that a file browser component could be used for other purposes, such as selecting a whole (sub)directory, or selecting an output file location. A complete example is 'traintimesfb' that is included with the project examples provided with the SDK distribution.

6.2 Usage examples

The use of a file browser in an Optimization Modeler application requires updates both on the UI side (see Section 6.2.1) and on the Mosel model side (see summary in Section 6.2.2). The latter involves considerably more work, since the model needs to prepare the input to the file browser component and implement the actions that should be triggered by the file selection.

6.2.1 VDL and Javascript

The UI component 'filebrowser' is added to VDL views with the tag `sdk-filebrowser` that always requires the arguments 'entity' and 'source' to be defined, where 'entity' is a model parameter or string scalar that receives the file selection and 'source' is the XML file listing in string format that is generated by the Mosel model. The text enclosed between the `sdk-filebrowser` markers will be used as the text for the button that opens the file browser window—if no text is provided it defaults to 'Browse'. An optional argument of the filebrowser tag is 'folder-only': if set to true, a folder name can be selected from the directory listing as the file selection to be returned.

```
<!-- Minimal form of the filebrowser tag -->
<sdk-filebrowser entity="FILESEL" source="FILELIST"></sdk-filebrowser>
```

The `sdk.min.js` and `sdk.css` files need to be included in the `client-resources` js and `css` sub-folders respectively. The `sdk.vdl` file should also be included.

The `sdk` is added to your VDL page via a `vd1-include` tag as follows:

```
<vd1-include src="sdk.vdl"></vd1-include>
```

Typically, the file selection component will be associated with a text field displaying the selection and a button that applies the file selection by reloading the scenario (if the selected data is

handled as managed input data) or running it (if the selected data file serves as unmanaged input data or for result output).

```
<vdl-row><p>Select an input data file before (re)running the scenario:<p></vdl-row>
<vdl-row>
  <vdl-form>
    <vdl-field entity="FILESEL" inline="true"
      style="width: 400px; float: left; margin-right: 2px;"></vdl-field>
    <sdk-filebrowser entity="FILESEL" source="FILELIST">Select data file</sdk-filebrowser>
  </vdl-form>
</vdl-row>
<vdl-row>&nbsp;</vdl-row>
<vdl-row>
  <vdl-execute-button id="run" mode="RUN" caption="Run Scenario"></vdl-execute-button>
</vdl-row>
```

The view elements resulting from this VDL code are shown in Figure 6.1.

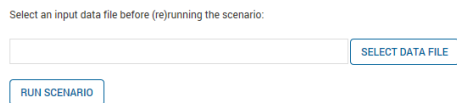


Figure 6.1: File selection component in VDL view

Clicking on the *Select data file* button in Figure 6.1 opens a file browser window on the directory listing specified in the 'source' argument, like the example shown in Figure 6.2. Subdirectories in the file listing are opened by clicking on their folder icon. Select a file name in the browser display to make it the current selection and confirm with *OK* to close the window. Note that selecting *Refresh* in the file browser window will trigger the OM execution mode 'sdkfb_refreshlist'.



Figure 6.2: File browser window

6.2.2 Mosel model updates for using a file browser

The Mosel model provides as input to the UI component 'filebrowser' a directory listing in a specific XML format (FILELIST in the following examples) and receives back a string that indicates the selection made among these files by the user (we shall be using the entity name FILESEL). The *XML directory listing* can be generated with the help of the *sdksutil* subroutines *fbfilelistxml* (generation of an XML document) or *fbupdatefilelist* (returns a string generated from the XML as required by the UI component 'fileupload'). The directory listing functionality of *sdksutil* supports three different types of directory listings:

- standard file system,

- browsing project and scenario attachments of an OM application,
- listing the contents of a bucket in an Amazon S3 repository.

Directory listings recursively include subdirectories up to a specified depth (determined by the parameter `FB_FILELIST_MAXDEPTH`) and subject to a limit on the number of nodes listed (parameter `FB_FILELIST_MAXNODES`). The directory type is specified as an argument to the two routines `fbfilelistxml` and `fbupdatefilelist`, other configuration options are the specification of a filter on the file name and whether to list all files or directory names only—see the corresponding subroutine documentation in the SDK Mosel Packages Reference Manual for details.

The result of the file selection returned from the UI into the model will have to be treated according to the selected directory type, the file type, and the purpose of this particular file access. The following sections describe different cases of file selection used to upload an input file with managed or unmanaged input data. Since attachments are the only directory type that does not require any specific setup we start by discussing this option—the corresponding project files are provided with the SDK distribution ('traintimesfb').

6.2.3 Selecting an attachment as input file

The general model structure we are working with is the following:

```
inputdata           ! Input data from Optimization Modeler
calculatederiveddata ! Calculate derived data
defineproblem       ! Define constraints and objective function
runoptimizer        ! Setup the optimizer and run it
reportsol           ! Report the solutions
infolog("Model run terminated successfully.")
exit                ! Clean exit with error management
```

The data input in `inputdata` takes two different forms, depending on whether the input file selected via the UI is treated as *managed* (read during scenario loading, selection information passed via model parameters) or *unmanaged data* (read during scenario run, selection information passed via scalar entities).

- Data initialization for managed data:

```
! Input data from Optimization Modeler
case insightgetmode of
  INSIGHT_MODE_LOAD: do          ! *** Stop after data initialization
    initdata                     ! Fixed input data
    FILESEL:=DATAFILE            ! Parameter DATAFILE contains selection
    if FB_FILELIST_REFRESH then
      res:=updatefilelist        ! Generate updated directory listing
      exit                      ! Model is run to refresh directory listing, stop here
    elif selectfile>1 then       ! Retrieve selected file
      readdata(string(pathsplit(SYS_FNAME,FILESEL))) ! Read selected input
    else
      exit(ERR_DATA, "Error reading input data (load)")
    end-if
    exit(0)
  end-do
  INSIGHT_MODE_RUN: do           ! *** Inject scenario data and continue
    insightpopulate
  end-do
  INSIGHT_MODE_NONE: do         ! *** Run outside of Optimization Modeler
    initdata                     ! Fixed input data
    FILESEL:=DATAFILE            ! Use default selection as input
    readdata(string(pathsplit(SYS_FNAME,FILESEL))) ! Read selected input
  end-do
```

```

else
  exit(ERR_DATA, "Unknown execution mode")
end-case

```

■ Data initialization for unmanaged data:

```

! Input data from Optimization Modeler
case insightgetmode of
  INSIGHT_MODE_LOAD: do      ! *** Stop after data initialization
    initdata                  ! Fixed input data
    if updatefilelist<1 then  ! Generate updated directory listing
      exit(ERR_DATA, "Error reading input data (load)")
    end-if
    if FILESEL="" then
      warninglog("No input data file selected")
    end-if
    exit(0)
  end-do
  "sdkfb_refreshlist": do    ! *** Refresh file listing
    insightpopulate
    res:=updatefilelist      ! Generate updated directory listing
    exit                    ! Model is run to refresh directory listing, stop here
  end-do
  INSIGHT_MODE_RUN: do      ! *** Inject scenario data and continue
    insightpopulate
    if selectfile>1 then     ! Retrieve selected file
      readdata(string(pathsplit(SYS_FNAME,DATAFILE))) ! Read selected input
    else
      exit(ERR_DATA, "Error reading input data")
    end-if
  end-do
  INSIGHT_MODE_NONE: do     ! *** Run outside of Optimization Modeler
    initdata                ! Fixed input data
    FILESEL:=DATAFILE       ! Use default selection as input
    readdata(string(pathsplit(SYS_FNAME,FILESEL)))
  end-do
else
  exit(ERR_DATA, "Unknown execution mode")
end-case

```

The following model entities are used for the implementation of the file selection:

```

parameters
  DATAFILE=""              ! Default file selection
end-parameters

!@om.manage input
!@om.update.afterexecution
public declarations
  FILELIST: string          ! XML representation of attachment listing
  FILESEL: string           ! Selected file
end-declarations

```

Notice that for the 'managed data' case the entity FILELIST could simply be managed as 'result' and FILESEL does not have to be public.

Let us now take a look at the two routines `selectfile` and `updatefilelist` that provide the actual implementation of how to select a file or obtain the directory listing (their definition is exactly the same for both cases).

```

! Update directory listing
function updatefilelist: integer
  FILELIST:=fbupdatefilelist(FB_DIR_ATTACH, "*.xls|*.xlsx")
  returned:=(FILELIST<>"", 1, 0)
end-function

```

```

! If a file is selected, download it from attachments. Update directory listing
function selectfile: integer
  returned:= updatefilelist
  if FILESEL="" then
    warninglog("No input data file selected")
  else
    ! Read an attachment (project or scenario attachment)
    case string(pathsplit(SYS_DIR,FILESEL)) of
      "Project": insightgetprojattach(string(pathsplit(SYS_FNAME,FILESEL)))
      "Scenario": insightgetscenattach(string(pathsplit(SYS_FNAME,FILESEL)))
      else writeln("Unknown attachment type.")
    end-case
    if insightattachstatus=INSIGHT_ATTACH_OK then
      logging(3, "Attachment retrieved.")
      returned:=2
    else
      logging(1, "Attachment not found: ", FILESEL)
    end-if
  end-if
end-function

```

The directory listing for attachments generated by *sdkutil* contains two subdirectories, *Project* and *Scenario*. In our implementation of *selectfile* the path component of the file name serves for selecting the suitable version of *insightget*attach*, and we remove it from the file name that is passed into these routines.

In *updatefilelist* we call the SDK routine *fbupdatefilelist* with a filter on the file names to be returned: our model works with Excel spreadsheets as input files. In order to list all attachments, the filter argument would have been left empty.

The *readdata* procedure that is called from the different cases of the initialization logic could look as follows, where the *IODRV* model parameter needs to be set in accordance with the expected file type (*IODRV*=*"mmsheet.xlsx"* in our case):

```

! Read the input data from the specified file
procedure readdata(dfile:string)
  if not(utilfexists(dfile)) then
    exit(ERR_DATA, "Could not locate file: "+dfile)
  end-if

  setparam("ioctl", true)      ! Switch to I/O error handling by the model
  initializations from IODRV+dfile
  ...
end-initializations

  if getparam("iostatus")<>0 then
    exit(ERR_DATA, "Error reading input data")
  else
    infolog("Successfully read input data")
  end-if
  setparam("ioctl", false)     ! Switch back to default I/O error handling
end-procedure

```

6.2.4 Selecting an input file from a standard file system

If Optimization Modeler and the Mosel instance used for executing the model are installed on the same machine, you may choose to enable local directories for read or read/write access by including them in the configuration file of the Mosel server *xprmsrv* (this is the file *xprmsrv.cfg* in the *bin* subdirectory of your Xpress installation—remember to restart the server service in order to apply any changes made to this file), for example:

```

[insight]
MOSEL_RESTR=NoExecWOnly
MOSEL_ROPATH=C:/Temp

```

```
MOSEL_RWPATH=C:/Temp2/Test
```

NB: You can check whether the correct settings are applied to a model by displaying their values from the model:

```
writeln("ROPATH: ", getenv("MOSEL_ROPATH"))
writeln("RWPATH: ", getenv("MOSEL_RWPATH"))
writeln("Restr: ", getparam("RESTRICT"))

! Check the access rights on the directory DATADIR="C:/Temp"
writeln("Input dir: ", DATADIR,
        " exists: ", bittest(getfstat(DATADIR),SYS_DIR)=SYS_DIR,
        " is readable: ", bittest(getfstat(DATADIR),SYS_READ)=SYS_READ)
```

Taking the case of unmanaged input data, we use the same data input case handling as with attachments:

```
case insightgetmode of
  INSIGHT_MODE_LOAD: do          !*** Stop after data initialization
    initdata                      ! Fixed input data
    if updatefilelist<1 then      ! Generate updated directory listing
      exit(ERR_DATA, "Error reading input data")
    end-if
    if FILESEL="" then
      warninglog("No input data file selected")
    end-if
    exit(0)
  end-do
  "sdkfb_refreshlist": do        ! *** Refresh file listing
    insightpopulate
    res:=updatefilelist          ! Generate updated directory listing
    exit                        ! Model is run to refresh directory listing, stop here
  end-do
  INSIGHT_MODE_RUN: do          ! *** Inject scenario data and continue
    insightpopulate
    if selectfile>1 then
      readdata(FILESEL)          ! Read selected input data
    else
      exit(ERR_DATA, "Error reading input data")
    end-if
  end-do
  INSIGHT_MODE_NONE: do         ! *** Run outside of Optimization Modeler
    initdata                      ! Fixed input data
    res:= updatefilelist          ! Generate updated directory listing
    FILESEL:= DATAFILE           ! Use default selection as input
    readdata(FILESEL)
  end-do
else
  exit(ERR_DATA, "Unknown execution mode")
end-case
```

And the two routines `updatefilelist` and `selectfile` now are defined as follows (no changes to any other parts of the model are required):

```
! Update directory listing
function updatefilelist: integer
  FILELIST:=fbupdatefilelist(FB_DIR_DEFAULT,DATADIR,"*.xls|*.xlsx")
  returned:=(FILELIST<>"", 1, 0)
end-function

! If a file is selected, check for its existence. Update directory listing
function selectfile: integer
  returned:=updatefilelist
  if FILESEL="" then
    exit(0, "No input data file selected")
  else
```

```

        if not(utilfexists(FILESEL)) then
            logging(1, "Could not locate file: "+FILESEL)
        else
            logging(1, "Selected file was found.")
            returned:=2
        end-if
    end-if
end-function

```

6.2.5 Selecting and downloading an input file from an HTTP server

In a distributed setting, the directory containing the data files may be made accessible via HTTP. The following two implementations of the `updatefilelist` and `selectfile` routines work with an HTTP server that provides the file listing in the form of an XML file in reply to the GET request 'listdir' and returns a file of the specified name in reply to the request 'file':

```

! Update the directory listing saved in FILELIST
function updatefilelist: integer
    status:=httpget("http://" + SERVERNAME + ":" + PORTNUM +
        "/listdir?filter=*.xls|*.xlsx", "text:FILELISTT")
    if status/100=2 then
        FILELIST:=string(FILELISTT)
        returned:=(FILELIST<>"", 1, 0)
    else
        logging(1, "HTTP Request failed with code: " + status + " (" +
            httpreason(status) + ")")
    end-if
end-function

! Update directory listing. Download the selected file into tmp: directory
function selectfile: integer
    returned:=updatefilelist
    if FILESEL="" then
        exit(0, "No input data file selected")
    else
        if httpget("http://" + SERVERNAME + ":" + PORTNUM +
            "/file?name="+urlencode(FILESEL), string(expandpath("tmp:"))+"/"+
            urlencode(pathsplit(SYS_FNAME,FILESEL)))/100<>2 then
            logging(1, "GET operation failed")
        else
            LOCALFILE:=expandpath("tmp:")+"/"+pathsplit(SYS_FNAME,FILESEL)
            returned:=2
        end-if
    end-if
end-function

```

The HTTP server can be implemented with Mosel (but this is not a necessity), in which case we use the *sdutil* routine `fbfilelistxml` to obtain the directory listing in the form of an XML file that is returned with the reply to the HTTP request 'listdir', as shown in the following code extract:

```

declarations
    flistxml: xmldoc
end-declarations

httpstartsrv                                ! Start the server

! Handle events received by the server
repeat
    wait                                     ! Wait for an event
    ev:=getnextevent
    if ev.class=EVENT_HTTPNEW then           ! Request pending
        r:=integer(ev.value)                ! Get request ID

        if httpreqlabel(r)="listdir" then
            !**** Extract directory listing and return it to sender ****

```

```

        reset(flistxml)
        fbfilelistxml(FB_DIR_DEFAULT, tdir, flistxml, readlabel(r,"filter"))
        save(flistxml,getfirstchild(flistxml,0),httpreqfile(r))
        httpreply(r,httpreqfile(r))

    elif httpreqlabel(r)="file" then
        !**** Return requested file to sender ****
        fname:=readlabel(r,"name")
        if bittest(getfstat(fname), SYS_TYP) = SYS_REG then
            httpreply(r,fname)                ! If available: send it
        else
            httpreplycode(r,HTTP_NOT_FOUND) ! Otherwise: reply "not found"
        end-if

    elif httpreqlabel(r)="stop" then
        !**** Stop HTTP server ****
        httpreplycode(r,HTTP_OK)                ! Reply "success"
        break
    end-if
end-if
until false

```

The auxiliary routine `readlabel` reads the value of a label from an HTTP request received by the server:

```

function readlabel(r:integer, lab:string):string
    if httpreqstat(r)=3 then                ! 3: some label data is available
        setparam("ioctl",true)            ! Ignore errors (eg, label not found)
        initialisations from httpreqfile(r)
        returned as lab
    end-initialisations
    setparam("ioctl",false)
end-if
end-function

```

6.2.6 Selecting and downloading an input file from S3

Access to remote files in an Amazon S3 repository can be achieved via the `s3` functionality. We work once more with the case of unmanaged input data, with the following data initializations code that downloads the data file from S3 irrespective of whether the model is run from an OM application or standalone.

```

case insightgetmode of
    INSIGHT_MODE_LOAD: do                ! *** Stop after data initialization
        initdata                        ! Fixed input data
        res:=selectfile(true)            ! Generate updated directory listing
        if FILESEL="" then
            warninglog("No input data file selected")
        end-if
        exit(0)
    end-do
    "sdkfb_refreshlist": do                ! *** Refresh file listing
        insightpopulate
        res:=selectfile(true)            ! Generate updated directory listing
        exit                            ! Model is run to refresh directory listing, stop here
    end-do
    INSIGHT_MODE_RUN: do                ! *** Inject scenario data and continue
        insightpopulate
        if selectfile(false)>1 then        ! Retrieve selected file
            readdata("tmp:"+pathsplit(SYS_FNAME,FILESEL)) ! Read selected input
        else
            exit(ERR_DATA, "Error reading input data")
        end-if
    end-do
    INSIGHT_MODE_NONE: do                ! *** Run outside of Optimization Modeler
        initdata                        ! Fixed input data

```

```
FILESEL:=DATAFILE          ! Use default selection as input
res:=selectfile(false)     ! Retrieve selected file
if res>1 then
  readdata("tmp:"+pathsplit(SYS_FNAME,FILESEL)) ! Read selected input
else
  if res=1 then warninglog("No input data file selected"); end-if
  exit(ERR_DATA, "Error reading input data (load)")
end-if
end-do
else
  exit(ERR_DATA, "Unknown execution mode")
end-case
```

Given that the connection via *s3* requires a login with user authentication, a key and some further configuration settings, the routine `updatefilelist` has been integrated into `selectfile` (adding a Boolean argument `listonly` for switching between the two versions) in order to handle the connection in a single place. The authentication details are contained in an object of the type `s3bucket` that needs to get passed into the corresponding *sdkfilebrowser* routines.

```
function selectfile(listonly:boolean): integer
  declarations
    mybucket: s3bucket
  end-declarations
  !... Configure 'mybucket' with connection credentials

  ! Update directory listing
  logging(3,"Reading S3 directory " + mybucket.keyprefix + ROOT)
  FILELIST:=fbupdatefilelist(mybucket,ROOT,"/*.*.xls|*.xlsx")
  logging(3,FILELIST)
  returned:=if( FILELIST<>"", 1, 0)

  if DATAFILE<>" and not listonly then
    s3getobject(mybucket, ROOT+DATAFILE, "tmp:"+pathsplit(SYS_FNAME,DATAFILE))
    if s3status(mybucket)=S3_OK then
      logging(3, "Selected file loaded.")
      returned:=2
    else
      warninglog(3, "Could not load selected file.")
    end-if
  end-if
end-function
```

CHAPTER 7

Data Explorer

7.1 Overview

The VDL component 'data explorer' makes it possible to explore all public model entities that have not been marked as 'always hidden' in the schema configuration without having to implement VDL views for individual entities, similarly to the built-in table view of the Optimization Modeler Analyst Client. The data explorer takes two forms, namely

- general data explorer (*deprecated*)
- element-wise data explorer

7.2 Usage examples

The SDK 'data explorer' is purely UI (display) functionality, it does not have any impact on the model.

The code examples and screenshots presented in this section are taken from the *folio* example which is part of the SDK distribution.

7.2.1 General data explorer

From Optimization Modeler 4.5 onwards the general data explorer SDK component is deprecated as it has been incorporated into the Optimization Modeler product. For users with the appropriate privileges an 'Entity Explorer' view is added to an 'Advanced' view group providing an enhanced version of this view.

7.2.2 Element-wise data explorer

An element-wise data explorer is created by adding the tag `sdk-dataexplorer` to a VDL page, setting its argument `type` to 'element' as shown below. The resulting view (see Figure 7.1) lets the user select an element in a particular set and the display will show all arrays indexed by this set and that define entries for the specified set element. Typically, the data explorer will make up for the full page contents.

```
<vdl version="2.0">
  <page>
    <sdk-dataexplorer type="element"></sdk-dataexplorer>
  </page>
</vdl>
```


Please notice that the data explorer component is only supported for VDL 2.0.

Using the data explorer requires the SDK Javascript to be initialized from the HTML runner of the view, without any other specific setup:

```
insight.ready(function () {
  window.sdk.init();
  var view = insight.getView();
  view.applyVDL( '#main', window.sdk.onRedraw );
});
```

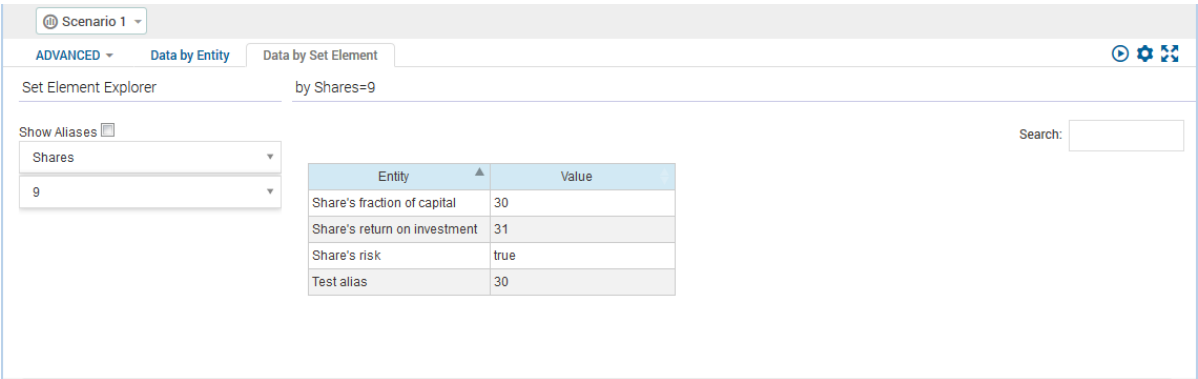


Figure 7.1: Element-wise data explorer

APPENDIX A

Contacting FICO

FICO provides clients with support and services for all our products. Refer to the following sections for more information.

Product support

FICO offers technical support and services ranging from self-help tools to direct assistance with a FICO technical support engineer. Support is available to all clients who have purchased a FICO product and have an active support or maintenance contract. You can find support contact information on the Product Support home page (www.fico.com/support).

On the Product Support home page, you can also register for credentials to log on to FICO Online Support, our web-based support tool to access Product Support 24x7 from anywhere in the world. Using FICO Online Support, you can enter cases online, track them through resolution, find articles in the FICO Knowledge Base, and query known issues.

Please include 'Xpress' in the subject line of your support queries.

Product education

FICO Product Education is the principal provider of product training for our clients and partners. Product Education offers instructor-led classroom courses, web-based training, seminars, and training tools for both new user enablement and ongoing performance support. For additional information, visit the Product Education homepage at www.fico.com/en/product-training or email producteducation@fico.com.

Product documentation

FICO continually looks for new ways to improve and enhance the value of the products and services we provide. If you have comments or suggestions regarding how we can improve this documentation, let us know by sending your suggestions to techpubs@fico.com.

Sales and maintenance

USA, CANADA AND ALL AMERICAS

Email: XpressSalesUS@fico.com

WORLDWIDE

Email: XpressSalesUK@fico.com

Tel: +44 207 940 8718

Fax: +44 870 420 3601

Xpress Optimization, FICO

FICO House

International Square

Starley Way

Birmingham B37 7GN

UK

Related services

Strategy Consulting: Included in your contract with FICO may be a specified amount of consulting time to assist you in using FICO Optimization Modeler to meet your business needs. Additional consulting time can be arranged by contract.

Conferences and Seminars: FICO offers conferences and seminars on our products and services. For announcements concerning these events, go to www.fico.com or contact your FICO account representative.

About FICO

FICO (NYSE:FICO) delivers superior predictive analytics solutions that drive smarter decisions. The company's groundbreaking use of mathematics to predict consumer behavior has transformed entire industries and revolutionized the way risk is managed and products are marketed. FICO's innovative solutions include the FICO® Score—the standard measure of consumer credit risk in the United States—along with industry-leading solutions for managing credit accounts, identifying and minimizing the impact of fraud, and customizing consumer offers with pinpoint accuracy. Most of the world's top banks, as well as leading insurers, retailers, pharmaceutical companies, and government agencies, rely on FICO solutions to accelerate growth, control risk, boost profits, and meet regulatory and competitive demands. FICO also helps millions of individuals manage their personal credit health through www.myfico.com. Learn more at www.fico.com. FICO: Make every decision count™.

Index

A

- adhoc view, 52
- aggregation operators, 6
- attribute, 14
- attribute name, 16
 - default, 16
- autotable, 19

C

- Chess problem, 3
- companion file, 4, 33
- condition editor, 18
- constraint
 - suggestions, 8
- constraint creation, 3
- constraint editor, 19
 - fixed, 3
 - user-defined, 6
- CstrAggregator, 20
- CstrDecisionLabels, 6, 19
- CstrDecisions, 6, 19
- CstrDescr, 3
- CstrOperator, 3
- CstrOperators, 5
- CstrOperatorSymbols, 5
- CstrRHSVal, 3
- Cstrs, 3, 7
- CstrSlack, 8
- CstrSlackObj, 8
- cstrsuggestimprovement, 9
- cstrsuggestrepair, 8
- ctrcreate, 3
- ctrcreate*, 20
- ctrcreatesum, 3

D

- data explorer, 52
- data rule, 40
- data validation, 40
- Data Validation Editor, 41
- Data Validation Report, 41
- database field types, 26
- database operations, 15
- datarule, 42
- decision, 6, 19
- directory listing, 44
- drdisplay, 42
- drupdate, 42

E

- editor-options-set, 6
- errorlog, 31

- exit, 31, 32
- exit behaviour, 31

F

- FB_FILELIST_MAXDEPTH, 45
- FB_FILELIST_MAXNODES, 45
- fbfilelistxml, 44, 45, 49
- fbupdatefilelist, 44, 45, 47
- file browser, 43
- file selection, 43
- file upload, 43
- filter, 14
- filter creation, 15

H

- history, 36

I

- in-memory operations, 15
- infolog, 31

L

- logging, 31, 33

M

- managed input data, 45
- message dictionary, 34
- message logging, 31
- messaging notification, 32
- model termination, 31
- modifier, 41

N

- note
 - sorting, 39
- note viewer, 36
- notes, 36
- notifications, 32

O

- operator selection, 5
- operator symbols, 5

S

- S3, 50
- sdk-history, 36
- sdk-messages, 32
- sdk-segment-editor, 14
- sdk-segment-group-editor, 14
- sdkconstraint, 2
- sdksegment, 13
- segaddcondition, 14
- segattrid, 16

- SegAttrName, 16
- SegAttrTypes, 27
- segcopydbtomem, 15
- segdbopen, 15
- segdbsuggest, 15
- segfilter, 27
- segget, 15, 26
- seggetattr, 15, 26
- seggetmap, 23
- seggetsize, 14
- seggrpcheck, 22
- segloadcfgcsv, 16
- segloadcfgdb, 14
- seglookupsegment, 17
- segmemfindsegments, 17
- segmemsuggest, 15
- segmenttest, 27
- segment editor, 17, 18
- segment group, 21
- segsample, 15
- SegSegmentFilterOpId, 27
- SegSegmentFilterOpSymbols, 27
- segsetattr, 15, 16, 26
- segupdate, 22
- slack variable, 8
- soft limit, 42
- SuggestDescr, 11
- suggestion table, 11
- suggestions
 - improve, 9
 - repair, 8
- SuggestSelect, 11
- SuggestValue, 11

T

- time measurement, 35
- type conversion, 26

U

- unhideAllSlacks, 8
- unmanaged input data, 45
- user-defined constraints, 6, 19
- utiladddictentry, 34
- UTIL_ERR_NOIMPL, 34
- UTIL_ERR_OR, 34
- utilgetdictentry, 34
- UTIL_INTVERBOSE, 31
- UTIL_LOGFORMAT, 31
- UTIL_MAXERRLOGSIZE, 31
- UTIL_MAXLOGSIZE, 31
- UTIL_MAXWARNLOGSIZE, 31
- utilsetdictentry, 34
- utilsetparam, 31, 32
- UTIL_VERBOSE, 31
- UtilVerbose, 31
- utilwritedict, 34

V

- VDL, 3, 18, 36
- vdl-column, 6

- vdl-table, 3

W

- warninglog, 31

X

- XML directory listing, 44