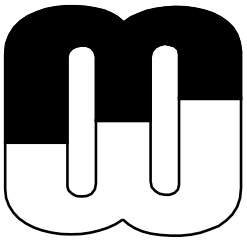


WHITE PAPER

IMPORTING MODELS WITH METAEDIT+ API

C# CONNECTION FROM EXCEL

MetaCase
Ylistönmäentie 31
FI–40500 Jyväskylä, Finland

Phone +358 400 648 606
E-mail: info@metacase.com
WWW: http://www.metacase.com

IMPORTING MODELS WITH METAEDIT+ API FROM EXCEL

Abstract

This paper describes how requirements specified in Excel are integrated with requirements in models of MetaEdit+ by using the MetaEdit+ API. The Excel connectivity is implemented with Visual Studio Web Service support.

1 DOMAIN-SPECIFIC MODELING

Domain-Specific Modeling (DSM) raises the level of abstraction beyond programming by specifying the solution using directly domain concepts. One suitable application area is architectural modeling languages, like EAST-ADL for automotive industry. EAST-ADL modeling language contains two kinds of requirement object types describing the requirements and how the architecture satisfies them. In this paper we describe how the requirements can be defined in Excel and integrated with requirements specified in models of MetaEdit+.

2 METAEDIT+ API

MetaEdit+ API provides the interface to access the MetaEdit+ repository programmatically in real-time. The API interface in MetaEdit+ is implemented as a SOAP Web Service server. The main function of API is to enable exchange of conceptual data between MetaEdit+ and other programs. The MetaEdit+ API supports reading, creating, and updating model elements. In addition to these data-centric functions, MetaEdit+ provides some additional features that can be accessed or controlled by the API, like highlighting or animating diagram elements for simulation or tracing purposes.

We will describe later in this paper at C# code level the integration with Excel and MetaEdit+. From Excel we export the requirements and import them to the MetaEdit+. The solution also updates the requirements changed in Excel with the ones already imported to MetaEdit+.

3 EXAMPLE

Every domain is different containing own concepts and rules for them, and therefore every DSM is also different. Here in this sample we use EAST-ADL language that is used for designing the architecture models for automotive industry. Two particularly relevant EAST-ADL concepts for our integration case the two different type of requirements; 'Requirement' and 'QualityRequirement'. Samples of these two different types opened in MetaEdit+ property dialogs are illustrated in Figure 1.

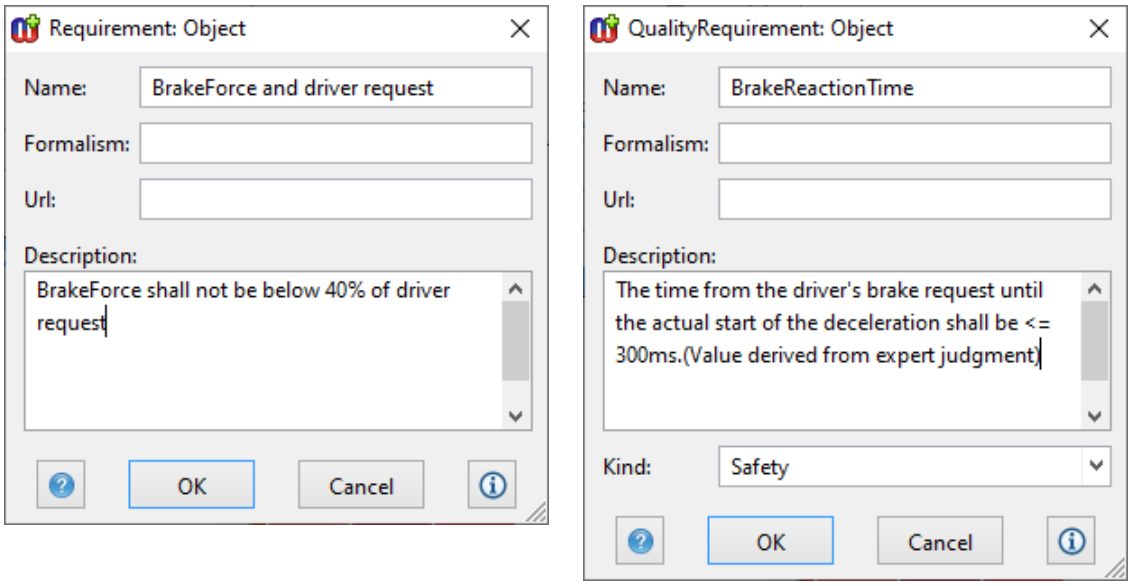


Figure 1. Requirement and QualityRequirement property dialogs

Requirement object consists of four property fields illustrated on the left of Figure 1: 'Name', 'Formalism' and 'Url' are specified as String fields and 'Description' is defined as multiline Text field. Difference between 'Requirement' and 'QualityRequirement' objects on the metamodel side, is the 'Kind' property type. 'Kind' is defined as pull-down property field illustrated at the bottom of the QualityRequirement on the right of Figure 1.

MetaEdit+ users can apply the connectivity solution as follows:

- Start MetaEdit+ and login to EAST-ADL project
- Start MetaEdit+ API
- Enter requirements in Excel and export them into MetaEdit+
- Check and use the imported requirement in MetaEdit+

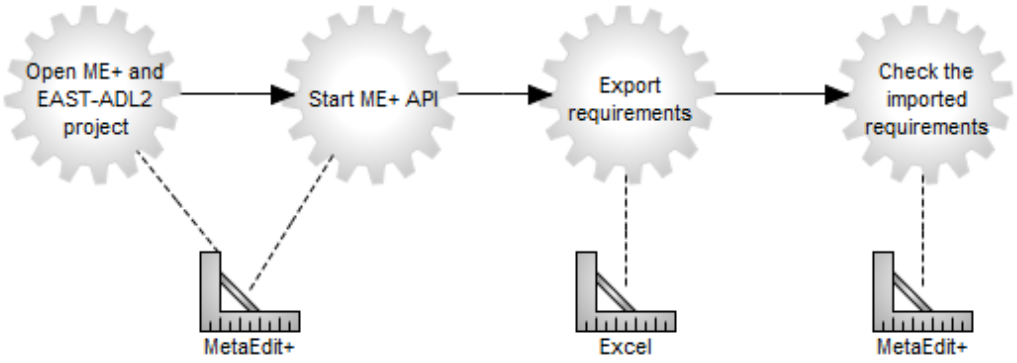
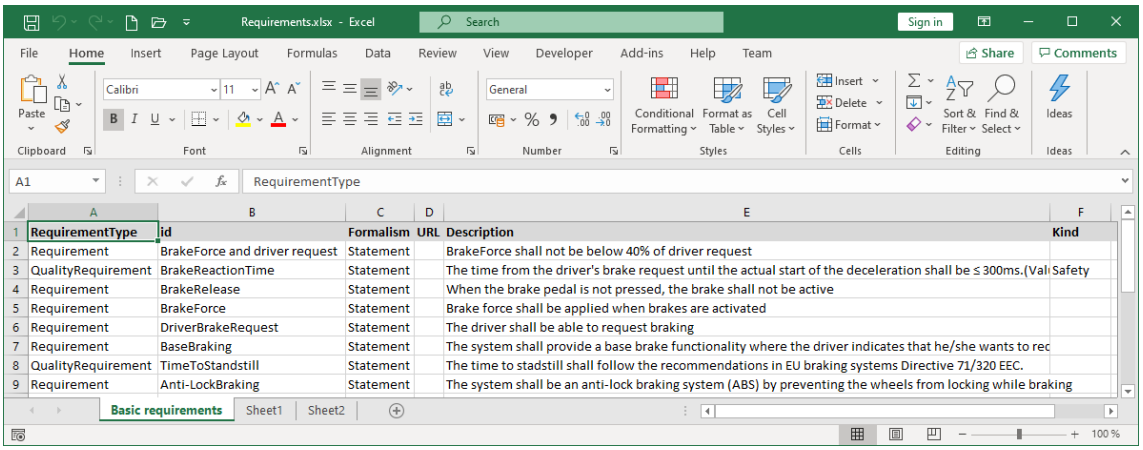


Figure 2. Importing process

3.1 Requirements in Excel

Figure 3 presents an example of Requirements and QualityRequirements described in Excel worksheet. Excel worksheet contains both of these requirements in the same table. Each individual requirement is listed on its own row and RequirementType column (A) specifies the type of the requirement. Kind column (F) specifies the value specific only for the QualityRequirement.



	A	B	C	D	E	F
1	RequirementType	id	Formalism	URL	Description	Kind
2	Requirement	BrakeForce and driver request	Statement		BrakeForce shall not be below 40% of driver request	
3	QualityRequirement	BrakeReactionTime	Statement		The time from the driver's brake request until the actual start of the deceleration shall be ≤ 300ms.(Vali	Safety
4	Requirement	BrakeRelease	Statement		When the brake pedal is not pressed, the brake shall not be active	
5	Requirement	BrakeForce	Statement		Brake force shall be applied when brakes are activated	
6	Requirement	DriverBrakeRequest	Statement		The driver shall be able to request braking	
7	Requirement	BaseBraking	Statement		The system shall provide a base brake functionality where the driver indicates that he/she wants to rec	
8	QualityRequirement	TimeToStandstill	Statement		The time to stadstill shall follow the recommendations in EU braking systems Directive 71/320 EEC.	
9	Requirement	Anti-LockBraking	Statement		The system shall be an anti-lock braking system (ABS) by preventing the wheels from locking while braking	

Figure 3. Sample of Excel requirements to be imported

When user wants to export these requirements to MetaEdit+, he first opens MetaEdit+ repository with EAST-ADL language, selects the default project for the imported requirements and starts MetaEdit+ API. Next user selects ‘Export to’ button from Add-Ins ribbon in Excel, shown in figure 4. After the export procedure is done an information dialog is shown summarizing the exporting results.

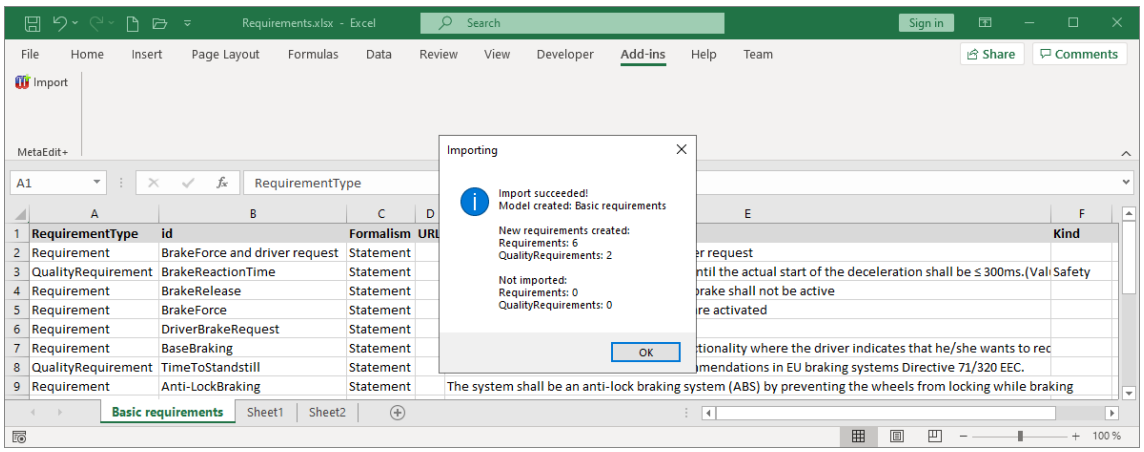


Figure 4. Sample of Excel requirements imported

4 IMPLEMENTING C# ADD-IN FOR EXCEL

4.1 Setting up the environment

To build the C# solution for exporting these requirements, first we initialize the Visual Studio's C# project and add Web Service support for it. Next we generate the MetaEdit+ WSDL description file for all API services available. Then we add the generated web service definition file (MetaEditAPI.wsdl) as a web service reference to our C# solution. After adding the WSDL reference, we are able to call and utilize API methods directly from C# code.

4.2 How to implement C# add-in in Excel

Main principle in exporting the requirements from Excel to MetaEdit+ is transform every requirement line in Excel first to the matching metamodel format in MetaEdit+. Here the format is that of requirements in EAST-ADL metamodel. This transformation is done in Visual Studio by using the C# programming language. After the transformation is done, C# will call API methods first to create the conceptual element into MetaEdit+ repository and later add this new object to the MetaEdit+ model.

Before importing any Requirement or QualityRequirement element, we first create a new RequirementsModel and give properties for it. New model is named as 'ImportedRequirements'. During the export and import steps we use the same model as a target for every single Requirement or QualityRequirement (row in Excel).

The C# code creating the new model structure is presented Listing 1. It shows the most important parts for the model creation logic. First we define new variable as METype and set its name value as "RequirementsModel". Every RequirementsModel graph has two property fields available: ModelName as a string and Description as a text. Next we

define a new variable for having both of these property values ‘graphPropertyValues’ which is defined as MEAny consisting two MEAnys.

‘graph_ModelName’ is defined as the first MEAny and its meType value is set as a String and meValue as “ImportedRequirements”. This ‘graph_ModelName’ is then set as the first item of graphPropertyValues. Within the similar manner the ‘graph_Description’ is defined as Text and value for it is set.

See more details about the API, methods and types from the documentation at: https://www.metacase.com/support/55/manuals/mwb/Mw-9.html.

```
METype graphType = new METype();
graphType.name = "RequirementsModel";
// Set Default values for the graph
MEAny[] graphPropertyValues = new MEAny[2];

MEAny graph_ModelName = new MEAny();
graph_ModelName.meType = "String";
graph_ModelName.meValue = "'ImportedRequirements'";
graphPropertyValues[0] = graph_ModelName;

MEAny graph_Description = new MEAny();
graph_Description.meType = "Text";
graph_Description.meValue = "' RequirementsLibrary imported from Excel: " +
                            DateTime.Now.ToString() + "'";
graphPropertyValues[1] = graph_Description;
```

Listing 1. Create a new RequirementsModel properties

Listing 2 presents the actual creation of the new graph information into the MetaEdit+ repository. After both of the properties have been defined, we initialize the connection to the MetEdit+ API server and name it as meServer.

After initializing the required variables are: graph_properties as an array of two MEAnys and nullAny as MEAny with MENull as a meType value. Next we can create a new graph information by using the ‘instProps’ method. instProps method has five parameters: graphType defined earlier in listing 1 as RequirementsModel, graph_properties (as an array of two MENulls), graphPropertyValues defined in listing 1 as an array having a string and text with corresponding values, and two variables: np and area both as MENulls to create the new model to the default project in MetaEdit+. The instProps method returns MEAny with the MEOop of the new graph instance created or MENull for an error. This result is set to the ‘newGraphAny’ variable.

```
MetaEditAPIPortTypeClient meServer = null;
meServer = new MetaEditAPIPortTypeClient();

MEAny[] graph_properties = new MEAny[2];
MEAny nullAny = new MEAny();

nullAny.meType = "MENull";
nullAny.meValue = "";
//Set Initializes the Graph properties
graph_properties[0] = graph_properties[1] = nullAny;

MEAny area, np;
np = area = nullAny;


//Create a new Graph
MEAny newGraphAny = new MEAny();
newGraphAny = meServer.instProps(graphType, graph_properties, graphPropertyValues,
np, area);
```

Listing 2. Creating the new graph

Next we will do the same for every Requirement and QualityRequirement row found in Excel. Some of the Requirements may have been exported and imported already earlier. In these cases it is preferable to be able to update the existing requirements rather than creating them as new ones again. To support the incremental updates, we utilize MetaEdit+ unique object identifier, oid, for every object stored in MetaEdit+ repository. After creating the new object information in the repository we add object's unique identifier back to the object's row in Excel. Next time if Requirement oid is found from Excel, we know that this requirement has been exported and imported earlier and we need to update the existing requirement rather than creating a new one.

In case of updating the existing requirement or qualityrequirements, the instProps' fourth parameter 'np' is specified as an existing oid (update is done to the same object) and it is 'nullAny' (if new object is created).

After the Requirement or QualityRequirement information has been added to the repository, we add the object information as part of the graph model by using the API's addToGraph method. Listing 3 shows method having two parameters, both defined as MEOop: First parameter contains the new object's oid and second contains the graph's oid.

```
MEOop newObjectOop = new MEOop();
MEOop newGraphOop = new MEOop();
newObjectOop = createNewMEOop(newObjectAny);
newGraphOop = createNewMEOop(newGraphAny);
meServer.addToGraph(newObjectOop, newGraphOop);
```

Listing 3. Adding the requirement to the graph

We also created own function for returning the MEOop from any given MEAny parameter. This createNewMEOop function is utilized before calling the addToGraph in listing 4.

```
public MetaEditAPI.MEOop createNewMEOop(MEAny newObjectAny)
{   //Creates a new MEOop from MEAny parameter, return tempMEOop
    MEOop tempMEOop = new MEOop();
    string[] tokens = newObjectAny.meValue.Split('_');
    tempMEOop.areaID = Convert.ToInt32(tokens[0]);
    tempMEOop.objectID = Convert.ToInt32(tokens[1]);
    return tempMEOop;
}
```

Listing 4. Creating MEOop from MEAny

After C# code is ready and built, we added a new Ribbon item to our solution in Visual Studio's Solution Explorer and associated a new icon with our C# code. Next we publish our new add-in solution.

After the solution is published it will create all necessary files for this type of Requirements export functionality in Excel. When you open the Excel it will ask whether you want to install the new add-in functionality. If you select the installation, the new add-in will appear in your Excel with the sample data in Excel worksheet.

5 CONCLUDING REMARKS

Models can be exported into MetaEdit+ from different sources. We described here the principle of using API to import models from another tool, Excel. If you plan to implement your own integration mechanism, please see the MetaEdit+ API manual at: http://www.metacase.com/support/55/manuals/

The C# code for this integration as based on EAST-ADL is available on request: info@metacase.com.