

**WHITE PAPER**

---

**IMPORTING MODELS WITH  
METAEDIT+ API**

**MetaCase**

E-mail: [info@metacase.com](mailto:info@metacase.com)

WWW: [www.metacase.com](http://www.metacase.com)

**USA:**

MetaCase

5605 North MacArthur Blvd.

11th Floor, Irving, Texas 75038

Phone (972) 819 2039

Fax (480) 247 5501

**International:**

MetaCase

Ylistönmäentie 31

FI-40500 Jyväskylä, Finland

Phone +358 14 641 000

Fax +358 420 648 606

# IMPORTING MODELS WITH METAEDIT+ API

## Abstract

This paper describes how Python function definitions are read in from external files and then automatically imported to the design models using MetaEdit+ API and Visual Basic's Web Services toolkit.

## 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 mobile devices, where recent Series60 phones have a Python programming environment available as one of the phone applications. With the help of a domain-specific language tailored for Python application development, developing new applications for phone devices becomes much easier and faster.

## 2 METAEDIT+ API

MetaEdit+ API provides the interface to access the MetaEdit+ repository programmatically in real-time. The main function is to enable exchange of conceptual data between MetaEdit+ and other programs. The API supports reading, creating, and updating model elements, as well as controlling MetaEdit+ for scripting and simulation support. Here we describe an example of how external file based function descriptions can be imported into the MetaEdit+ repository using MetaEdit+ API and Web Service References toolkit for Microsoft Excel 2002/Visual Basic (Figure 1).

The min procedure is as follows:

1. Reading the functions in
2. Creating right kind of data structure in Visual Basic
3. Importing new objects to MetaEdit+ through API.

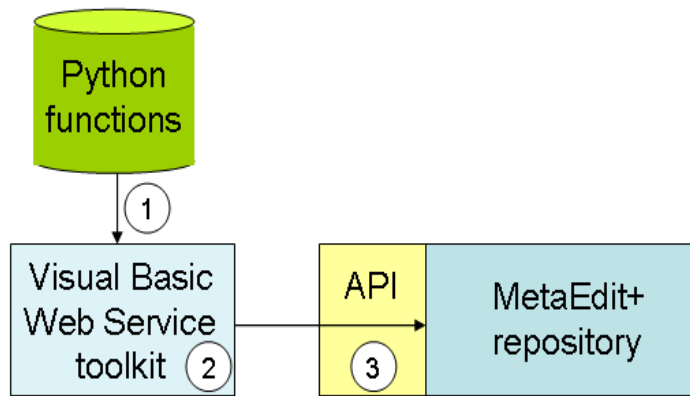


Figure 1. Importing structure

## 2.1 Example

Every domain is different, and therefore every DSM example is different. Here we use a domain that is well known and already in our pocket, mobile phone applications. In this example, we will use Series 60 phones and the Python framework (Nokia, 2004), which enables deployment of enterprise applications in mobile phones. The DSM language applies these directly as modeling concepts as illustrated in Figure 2.

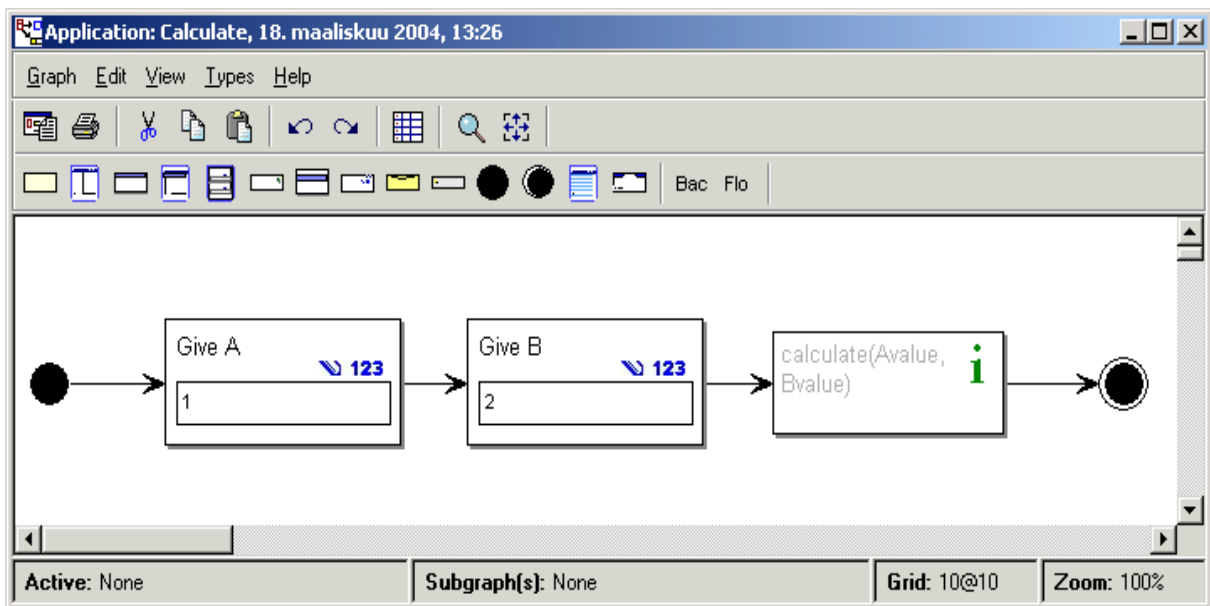


Figure 2. Calculate example

In the example, the application logic starts by asking two numbers in two separate query dialogs. After receiving the values from user, the sum of these two numbers is calculated and shown in the Note element and then the application ends. Arrows between the elements present the application's execution order. The design model is directly based on domain concepts, such as Note, Pop-up, SMS, Form, and Query. All other key concepts of the language can be seen in the model's toolbar. One of the key concepts is Function, which is used both inside the objects and as a part of the flow arrows in the picture (e.g. checkings for input data validness etc.). There are many

functions on the file level and while all the functions have similar internal structure, we like to import them automatically to the model without typing or copying & pasting them manually. One useful idea is to create an own model, where all reusable functions would be stored (e.g. ‘MyFavoriteFunctions’). Whenever function definition is needed, it would be easy to first start searching for it in the library model of reusable function definitions.

## 2.2 Functions

Function describes a reusable unit (definition) that is characterized by its name, parameters, body and documentation. An example of ‘calculate function’ is shown in Figure 3. A ‘calculate function’ has two parameters (a,b), it’s implementation code is placed in the Function body field while the Documentation field contains such information as the developer’s comments.

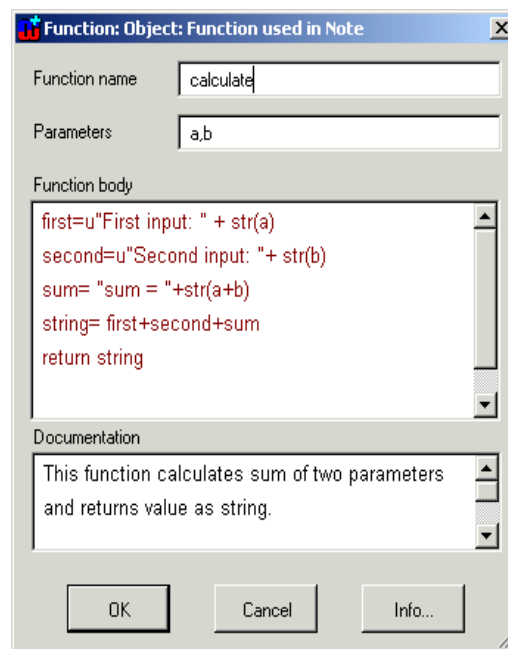


Figure 3. MetaEdit+’s function dialog

## 2.3 Building the environment

First we downloaded the Web Service References Toolkit 2.0 for Windows 2000 SP4 from the web (<http://www.msfn.org/comments.php?catid=2&id=7573>) and installed it to the computer. In the next phase, we generated the WSDL file from MetaEdit+ and after that, we opened Excel’s Visual Basic Editor and launched the MetaEdit+ WSDL file as ‘Web Service URL’. Visual Basic Web Service References Toolkit could not utilize the original WSDL file’s ‘MENull’ empty definition, we had to slightly modify it, see Figure 4. After that the toolkit generates automatically the necessary Visual Basic codes for all WSDL file’s API commands.

## Original

```
...  
</xsd:complexType>  
<xsd:complexType name="MENull" />  
  
<xsd:complexType name="MEOopArray">  
...
```

## Modified

```
...  
</xsd:complexType>  
  <xsd:complexType name="MENull">  
    <xsd:all>  
      <xsd:element name="notalot" type="xsd:string" />  
    </xsd:all>  
  </xsd:complexType>  
<xsd:complexType name="MEOopArray">  
...
```

Figure 4. WSDL file modification

## 2.4 Python file

Python functions to be imported contain the following structure. In Figure 5, 'calculate function' is used as an example; it sums two numbers together and returns the total.

```
1 def calculate(a,b):  
2 # This function calculates sum of two parameters and returns value  
  as string.  
3     first=u"First input: " + str(a)  
4     second=u" Second input: "+ str(b)  
5     sum= "sum = "+str(a+b)  
6     string= first+second+sum  
7     return string
```

Figure 5. Example of Python function

Each function starts with the 'def' keyword, followed by the name of the function, and then the list of parameters separated by a comma (,) inside the parenthesis. Each comment starts with the '#' mark. The main method body starts indented after the comment line and it ends on the line starting with the 'return' keyword. So in the example above row1 defines the Calculate function which has two parameters (a,b). Row2 represents the developer's comment about the function. Rows 3-7 make up the main body of the function

## 2.5 Visual Basic code structure

In Excel's Visual Basic we created the handling procedure where the user first selects the Python file to be imported into the MetaEdit+ database. The Visual Basic script then reads the opened file line by line. After finding the non-space character/word in beginning of the line input, it tests whether it is a reserved word ('def', '#', or tab). Based on that result, it defines the property field structure, which will be then filled in Excel. Additionally from the 'def' line, the function's parameters are parsed out. To visualize, debug and test that solution, we fill the imported property values of the Excel sheet, see Figure 6.

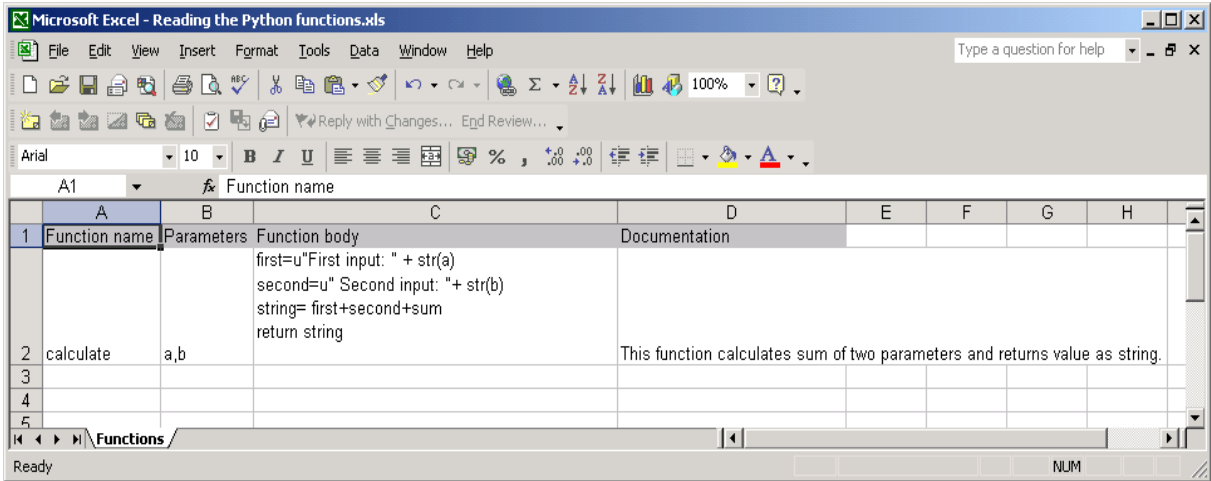


Figure 6. Imported values in Excel

In Excel, every function is put on an own row. The first two columns (Function name, Parameters) are defined as a ‘String’ type in the metamodel and the next two are defined as type ‘Text’.

After all functions are filled in Excel, the automated procedure will get the function line by line and fill the right property slots in Visual Basic to prepare for a new object creation in Visual Basic and in the MetaEdit+ database.

## 2.6 API commands in Visual Basic script

In Visual Basic code, we used the following Visual Basic data structures and API commands:

Data structures:

- object As New struct\_MEOop
- meAnyNull As New struct\_MEAny
  - meAnyNull.meType = "MENull"
  - meAnyNull.meValue = ""

```
Array(meAnyNull, meAnyNull, meAnyNull, meAnyNull)
```

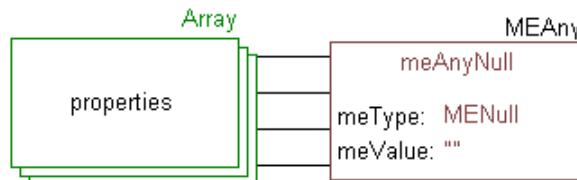


Figure 7. Properties array

In Figure 7 the Properties object is an array of four MEAny objects (each of these four MEAny object is named as “meAnyNull”, and it’s meType is “MENull”).

Adding property values to the empty property slots is done by using the `instProps` -command:

```
meAPI.wsm_instProps(new_object_type, properties, property_values, meAnyNull, meAnyNull)
```

- `new_oject_type` is defined as `METype`, which `METype.name="Function"`
- `properties` is defined as an array of four `MEAnys` (see Figure 7 above)
- `property_values` is an array of four `MEAny` objects (named as "new\_property"). See Figure 8 example of Calculate function's `property_values`.
  - `property_values(0)` `meType="String"`, `meValue="Calculate"`
  - `property_values(1)` `meType="String"`, `meValue="(a,b)"`
  - `property_values(2)` `meType="Text"`, `meValue="first=u"First input:..."`
  - `property_values(3)` `meType="Text"`, `meValue="This function calculates..."`

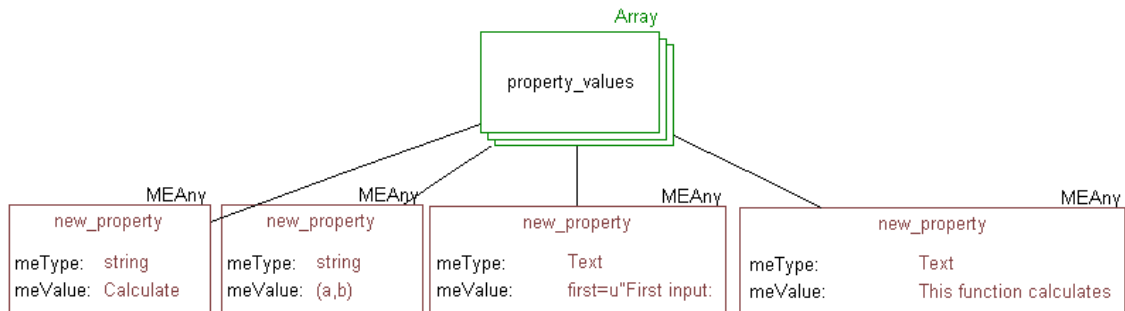


Figure 8. Example Calculate `property_values`

`instProps` -command returns a new object identifier as `MEAny` type. In the next step that object new identifier will be divided to object's `graphID` and `objectID` (both `MEOop` type).

```
meAPI.wsm_addToGraph(new_object, graph)
```

- `new_object` is the new object (type `MEOop`) and `graph` (type `MEOop`) is the model where the new object will be added.

## Full Visual Basic code related to instProps command:

```
Dim    meAnyNull As New struct_MEAny
        meAnyNull.meType = "MENull"
        meAnyNull_own.meValue = ""

properties=Array(meAnyNull, meAnyNull, meAnyNull, meAnyNull)
Set new_object_type = New struct_METype
new_object_type.name = "Function"

Set object_and_properties = New struct_MEAny
For property_number = 0 To 3
    Set new_property = New struct_MEAny

    If property_number = 0 Or property_number = 1 Then
        new_property.meType = "String"
    Else
        new_property.meType = "Text"
    End If

    new_property.meValue = "'" & Worksheets("Functions").Cells(value_row,
value_column).Value & "'"
    Set property_values(i) = new_property
    value_column = value_column + 1
Next
Set object_and_properties = meAPI.wsm_instProps(new_object_type, properties,
property_values, meAnyNull, meAnyNull)

Set added_object = meAPI.wsm_addToGraph(new_object, graph)
```

## 2.7 Usage

When using the solution, we first need to start the server in MetaEdit+ API. Next, we launch the Visual Basic macro, which opens the Python function file-selection dialog. An automated procedure then imports the functions from the selected file to the MetaEdit+ repository.

## 2.8 Special issues and notes

Each property type (String, Text, Collection, Object etc) requires its own block/structure in Visual Basic, which correctly maps the property value to the property field. Particularly in cases where there is a deep structure to be imported (property field, which includes an object, which has property, etc.), the Visual Basic structures do require more processing and hierarchy to enable automated importing.

Special characters (e.g. tabs, line feeds, etc.) do usually require detailed processing to ensure that they will remain in original format also after importing.

## REFERENCES

Nokia, (2004) *Python for Series60: API reference*, version 1.0, <http://www.forum.nokia.com/>