

**Rule-Writer's Tool Kit**

**PowerEditor 5.10**

Revised: February 11, 2008

Re-Released: September 7, 2018

The PowerEditor documentation suite includes the following materials:

PowerEditor Business Analysts Guide

PowerEditor Custom Reports Guide

PowerEditor Installation and Customization Guide

PowerEditor Reference Manual

PowerEditor Release Notes

PowerEditor Rule Writers Toolkit

*Release Notes* for earlier versions back to and including 4.0 are also provided in the release package. For technical assistance with upgrading or any other PowerEditor-related issue please contact CoreLogic Technical Support at 1-855.369.2410, or [ADESupport@corelogic.com](mailto:ADESupport@corelogic.com)

Table of Contents

[1. Introduction 3](#_Toc190513739)

[1.1 Purpose 3](#_Toc190513740)

[2. Guideline Templates 4](#_Toc190513741)

[2.1 Floats and Precision 4](#_Toc190513742)

[2.2 Exists With Operator 5](#_Toc190513743)

[2.3 Test Function Call from LHS 7](#_Toc190513744)

[3. Messages 8](#_Toc190513745)

[3.1 Column Message Examples 8](#_Toc190513746)

[3.2 Message Tag 9](#_Toc190513747)

[3.3 Column Message Attributes 9](#_Toc190513748)

[3.4 Configuring Conditional Message Defaults 11](#_Toc190513749)

[4. Guideline Actions 12](#_Toc190513750)

[4.1 Action Special Tokens 12](#_Toc190513751)

[5. Parameter Templates 15](#_Toc190513752)

[5.1 Overview 15](#_Toc190513753)

[5.2 Developing Parameter Templates 16](#_Toc190513754)

[5.3 Figures 17](#_Toc190513755)

[6. PowerEditor Tools Reference 20](#_Toc190513756)

[6.1 Overview 20](#_Toc190513757)

[7. Advanced Topics 26](#_Toc190513758)

[7.1 Time Slice 26](#_Toc190513759)

List of Figures

[Figure 2‑1 Display Decimals Field of Template 4](#_Toc165440469)

[Figure 4‑1 RHS Tokens 12](#_Toc165440470)

[Figure 4‑2 Error: Using %cellValue% in Template Rule 13](#_Toc165440471)

[Figure 5‑1 Sample Parameter Template Definition 18](#_Toc165440472)

[Figure 5‑2 Sample Parameter Template Deployed Object Instances 19](#_Toc165440473)

[Figure 6‑1 PowerEditor Migration Tool (Migration: 4.x to 4.2 Tool shown) 24](#_Toc165440474)

[Figure 6‑2 Template Repair Tool 25](#_Toc165440475)

[Figure 6‑3 Domain XML Tool 25](#_Toc165440476)

# Introduction

## Purpose

This is a handy little guide containing things you didn't want to know about PowerEditor but went ahead and asked anyway. The document is a holding place for very technical knowledge that is:

* too technical to be included in a user's guide
* not part of installation or customization guide

Some of this information might someday be rolled into a real rule-writer's guide that is soon to be part of the documentation suite. For now, the information is collected here.

This guide contains information that was previously scattered among the following reference documents:

PowerEditor Tools ReferenceTemplate Message XML Specification

PowerEditor Parameter Developers Guide

Exists/With Developer's Guide

# Guideline Templates

This section describes miscellaneous technical details for template creation.

## Floats and Precision

Precision can be specified in two places in the PowerEditor: the Template’s *Display Decimals* field, and the Domain file’s *Precision* attribute. This section provides the skinny on how each of these gets used. This information is up to date as of PE v5.0.

### Template Display Decimals Field

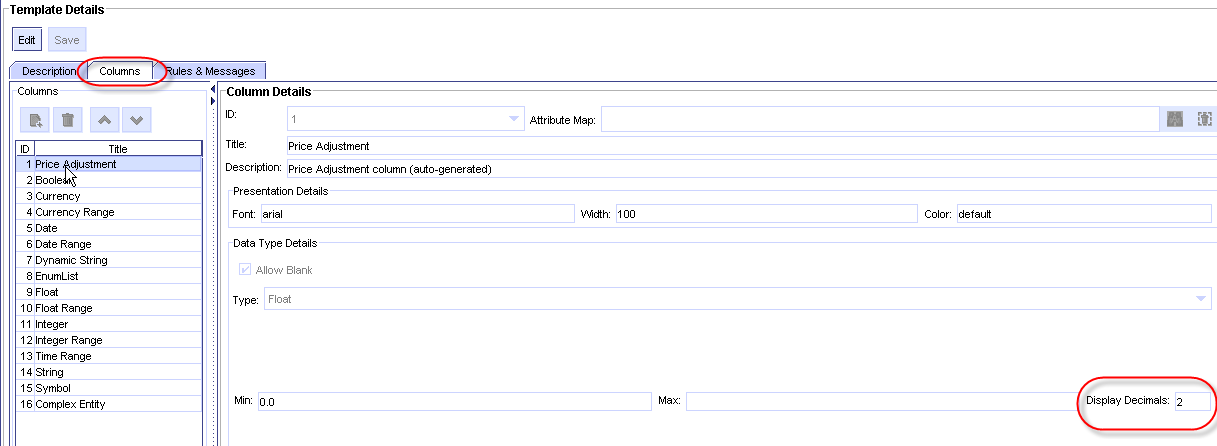


Figure ‑ Display Decimals Field of Template

This value impacts the behavior in two ways. Firstly, cell values in the guideline grid will be displayed with this precision.

Secondly, if the column is referenced in a message, the literal value in the message will be generated with precision. For example, a message of

Adjustment of %column 1%

will be generated as

:message (sprintf “Adjustment of .75”)

### Domain Precision Attribute

DomainAttributes of the domain file can have a *Precision*. For example:

<DomainAttribute DeployType="Percent” Name="Adjustment" **Precision="3"** …/>

The Precision value is ignored if the DeployType is not one of the following: Percent, Currency, or Float. The Precision value is used when generating messages that reference the domain attribute. For example, a message of

Adjustment of |Price.Adjustment|

will be generated as

:message (sprintf “Adjustment of .3f” ?adjustment)

Specifically, if a message references an attribute whose deploy type is Percent, Currency or Float, then %.*<p>*f will be generated. *<p>* is a precision specification as described below. *Important Note:* If your message contains a reference to an attribute that is *:UNSPECIFIED*, this will generate a runtime error. In general, you should write your rules so that :*UNSPECIFIED* values will not appear in messages.

The precision value *<p>* that appears in the %.*<p>*f is determined as follows.

* + - If the domain attribute has a precision value, this value is used. An example of a domain attribute precision value is as follows:

<DomainAttribute DeployType="Percent” Name="Adjustment" **Precision="3"** …/>

In this example, %.3f will be generated in the message.

* + - If the domain attribute does not have a precision value, the value defaults to 2, and %.2f will be generated. The value 2 is hard-coded and immutable.

## Exists With Operator

This section was last updated for PE 4.0. Even though the information is presented in old XML style, it is a useful reference.

### Overview

The exists-with operator is used

1. As a shorthand notation to refer to multiple attributes of the same class (instance, really)
2. To enforce lineage among parent-child classes
3. To enforce comparison among two different instances of the same class

The syntax[[1]](#footnote-1) of an exists-with expression is

“exists” ClassName [ InstanceName ] [“excluding” InstanceName] “with” ( ( AttributeName | ClassName.AttributeName ) Operator Value and ... )

#### As Shorthand Notation

Exists-with operator is used in the simplest form when used as a shorthand notation to reference multiple attributes of the same class. For example,

if exists Class1 with (Attribute1 == %column 1% and ... and AttributeN == %column N%) then action()

This illustrates the first point. You can reference multiple attributes of the same class with less typing. Expressions following the "with" operator can be any compound expressions. The above rule is equivalent to the following rule:

if Class1.Attribute1 == %column 1% and ... and Class1.AttributeN == %column N% then action()

#### Enforcing Lineage Among Parent-Child Classes

In the expresesion following the “with “ operator, you may also make references to Class.Attribute, not just attributes, as follows:

if exists Class1 with (Class2.Attribute1 == %column 1% and Class3.Attribute3 in %column 2%) then action()

This idiom specifies that Class1 is a parent class of Class2 and Class3. This requires that, in the Domain Definition XML, the parent class (Class1) must contain <DomainClassLink> tags for each of the classes references in the expression following the "with" operator, in this case Class2 and Class3. For each referenced child class, PE generates an extra pattern that "links" the parent and the child classes. This link enfores the parent and the child classes have the same lineage. For example, the following ARTScript rule would be generated from the above example.

(object ?group

(instance-of LDO:Class1)

(LDO:Class2-LINK ?class2)

(LDO:Class3-LINK ?class3)

)

(object ?class2

(instance-of LDO:Class2)

(LDO:Attribute1 ?attribute1 &:(eq ?documentationtype "SISA"))

)

(object ?class3

(instance-of LDO:Class3)

(LDO:Attribute2 ?attribute2 & "Permanent Resident Alien”)

)

Note that you cannot mix references to attributes and class.attributes in the expression following the “with” operator. That is, the following is illegal:

if exists C1 with (C22.Attrib = %column 2% and Attr3 in %column 3%) then ...

#### Comparison between two different instances

The exists-with expression is also used to reason over attributes of a two different instance of the same class, as follows:

if

exists Class1 instance1 with (Attr1 == %column 1%) and

exists Class1 instance2 excluding instance1 with (Attr2 == %column 2%)

then action()

The above idiom tests if there is an instance of Class1 with Attr1 equal to the value of Column one, and another instance of Class1 with Attr2 equal to the value of Column Two. The use of excluding, along with specific instance name allows comparison among two or more instances of the same class.

## Test Function Call from LHS

This section was last updated for PE 4.0. Even though the information is presented in old XML style, it is a useful reference.

To facilitate OR expressions, it is possible to make a call to a Boolean function from LHS of a deployment rule. The syntax for a test function call is

aetest.<Function-Name> in [ arg1, … , argN]

where argN can be any of the following:

* “|ClassName|”
* “|ClassName.AttributeName|”
* %colum N%
* String literal
* Integer literal
* Float literal

For example,

aetest.FunctionName in [“|ClassName|”,”|Class.AttributeName|”, %column 2%, “some-value”]

results in the following code in the generated ARTScript,

(test (FunctionName ?classname ?attributename 1495.75 "some-value"))

# Messages

This section is contains excerpts from an older Message XML specification, PE 4.0. This needs to be rewritten to include message-authoring screens, but is included here as a reference for otherwise hard-to-find information.

## Column Message Examples

Column messages are fragments of template messages that are generated conditionally based on the presence of a cell value.

The following is sample template message.

"Prepayment Penalty waived for |Lien.LienType| Lien because **%columnMessages(2,4,3,1,5,6)%"**

The following are sample column messages.

Column 1:

type="any"

"Citizenship must be %cellValue%

Column 2: Message fragment with object/attribute reference

"Doc Type |Transaction.DocumentationType| not allowed."

Column 3: Multiple Column Messages

type="enum"

cellSelection="default"

enumDelimiter=", "

enumFinalDelimiter=" or ">

"occupancy must be %cellValue%"

type="enum"

cellSelection="enumExcludeSingle"

"occupancy cannot be %cellValue%"

type="enum"

cellSelection="enumIncludeSingle"

occupancy must be one of: %cellValue%

type="enum"

cellSelection="enumExcludeMultiple"

enumDelimiter="; ">

occupancy cannot be one of: %cellValue%

Column 4: Range Style

type="range"

rangeStyle="bracketed">

%cellValue% CLTV

Column 4: Column message with no content - cell Value is used

type="enum"

rangeStyle="verbose"

## Message Tag

Message attributes are as follows:

conditionalDelimiter Text that will be placed between conditionally generated columns text. If spaces are desired, the spaces must be included in this text; the message generator will not surround the conditional delimiter with spaces. Optional: default specified in configuration XML

conditionalFinalDelimiter Text that will be placed between conditionally generated text for the next-to-last column text the last column text. If spaces are desired, the spaces must be included in this text; the message generator will not surround the conditional delimiter with spaces. Optional: default specified in configuration XML.

Message Fragments can be referenced as follows:

%columnMessages(<values>)% <values> can have one of two formats. A comma delimited set of column numbers indicates the columns which should be included in the conditional portion of the message. For example, %columnMessages(3,2,1,4,5)%. Alternatively, the word “all” indicates using all columns in increasing order. For example, %columnMessages(all)%.

## Column Message Attributes

Attributes of the MessageFragment tag are as follows. *Note that unless otherwise specified all of these are optional, and default to values specified in the configuration XML*.

type One of the following:

range: applies to any of the Range DataSpecs. The attribute rangeStyle is required.

enum: applies to EnumList DataSpec. The attribute cellSelection is required, the attributes enumDelimiter, finalEnumDelimiter, enumPrefix are optional.

any: applies to DataSpecs of any type. The only attribute allowed is prefix, which is optional.

rangeStyle For type="range", this specifies the style of the message that is generated. This must be one of “verbose”, “symbolic” or “bracketed”. Examples of these three styles are as follows:

verbose: “greater than or equal to zero and less than 100”

symbolic: “>=0, <100”

bracketed: “[0, 100)”

cellSelection For type="enum", this specifies phrases that are specific to options selected by the PowerEditor user for a given cell. **This must be used with multipleSelect EnumList columns only; otherwise, the message will not be output**. For example, the column message might be different if the cell contains single vs. multiple enumerated items, or if the “Exclude Selection” option is checked from an enumerated list. Possible values for this attribute are:

default: Specifies default phrase and/or options, overridden by those below.

enumExcludeSingle: Single enumerated value selected, “Exclude Selection” selected.

enumIncludeSingle: Single enumerated value selected, “Exclude Selection” not selected

enumExcludeMultiple: Multiple enumerated values selected, “Exclude Selection” selected

enumIncludeMultiple: Multiple enumerated values selected, “Exclude Selection” not selected

enumDelimiter For type="enum", with multipleSelect enumerated values, this specifies the phrase that is used between multiple selected items.

enumFinalDelimiter For type="enum", with multipleSelect enumerated values, this specifies the phrase that is used between next-to-last and the last of multiple selected items.

enumPrefix For type="enum", the text will be prepended to the cell value.

The value of the MessageFragment tag is a string that specifies the message fragment to be used for this column if the column does not contain a null value. The syntax for this is the same as message text. It can contain references to attributes, columns, and the cell value. If this value is blank, the cell value will be used, as in the form specified in the MessageFragment attributes and in the configuration XML.

## Configuring Conditional Message Defaults

The following is sample XML from a config.xml. The message section can be in the RuleGenerationDefault section or RuleGenerationUsageType section of the config file.

<MessageTypes>

<Message type="range" rangeStyle="verbose"/>

<Message type="enum"

cellSelection="enumIncludeMultiple"

enumDelimiter=", "

enumFinalDelimiter=" or "

enumPrefix=" any of "/>

<Message type="enum"

cellSelection= "enumExcludeSingle"

enumPrefix=" not "/>

<Message type="enum"

cellSelection="enumExcludeMultiple"

enumDelimiter=", "

enumFinalDelimiter=" or "

enumPrefix=" any of "/>

<Message type="conditional"

conditionalDelimiter=", "

conditionalFinalDelimiter=" and "/>

</MessageTypes>

# Guideline Actions

## Action Special Tokens

Action supports special tokens. As LHS, arguments of an action can be a reference to an attribute or column value, among other things. The following table summarizes them. This section is up to date as of PE 5.0.

|  |  |
| --- | --- |
| Token | Description |
|  |  |
| “%activationDate%” | Activation date for the rule (a string) |
| "%cellValue%" | Reference to the current cell value. *This token can only be used in column rules.* See notes below. |
| "%context%" | Generates a specification for the context associated with this rule. See example below. |
| "%column *<n>*%" | Column reference. For example, %column 1%. Note the space before the column number. |
| “%expirationDate%” | Expiration date for the rule (a string) |
| “%lineageID%” | Reference to the variable in the lineage-id pattern which is automatically generated for certain types of objects. This lineage pattern is specified in the PowerEditorConfig file. |
| “%rowNumber%” | Reference to the row number of the activation for this rule. Deployed as an integer. |
| “%ruleName%” | Reference to the name of the AE rule. Deployed as a symbol. |
| "%ruleID%" | Reference to the first column in the template of type ruleID |
| "%templateID%" | The internal template ID. Deployed as an integer. |
| "%templateName%" | The display name of the template. Deployed as a string. |
| "%*<entity-type>*ID%" | The ID for the entity that matches on the LHS. See example below. |
| $create | Mark the beginning of list creation call. See example below |
| create$ | Mark the end of the list creation call. See example below. |
| “|ClassName|” | Reference to an instance of a class. |
| “|ClassName.AttributeName|” | Attribute reference. |

Figure ‑ RHS Tokens

### CellValue

"%cellValue%" is intended for use with column rules, since column rules will be generated based on cell values. This token is equivalent to the column reference for the column to which the rule is attached. If the rule is attached to column 2, for example, then %cellValue% is equivalent to %column 2%.

Template rules have no notion of the current cell value. Currently, if you use this in a template rule, you'll see the following error on deploy:

.... Failed to generate rule: -2 (java.lang.ArrayIndexOutOfBoundsException)

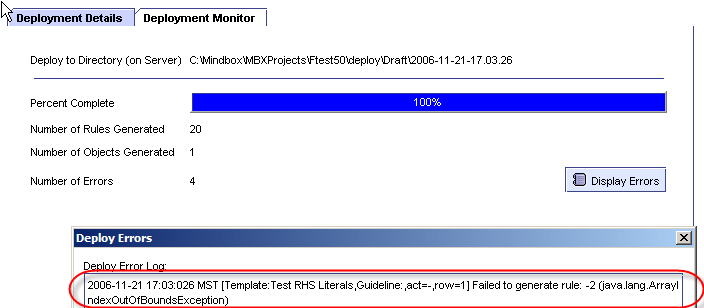


Figure ‑ Error: Using %cellValue% in Template Rule

### Context

If you specify "%context%" in your RHS action, the following code will be generated:

(create$ (build$ *<entityType1> <category/entity> <entityID1>* )

(build$ *<entityType2> <category/entity> <entityID2>* ) ...)

The generated code is a list of sub-lists. A sublist is generated for each entity or category that exists in the context. For example, if there are three categories in the context, three sub-lists will be generated.

* *entityType* is the deployName for the EntityType that's in the context
* *category/entity* will be one of the literal values *category* or *entity*, depending on which element is in your context.
* *entityID* is the internal ID of the entity that's in the context.

In the following examples, assume your Guideline Action is defined as follows.

pe\_action(context\_ids, "%context%" ...)

If your context is two products and a channel category, your generated rule will look like this:

(pe-action

:context (create$ (build$ Product entity 113)

(build$ Product entity 114)

(build$ Channel category 513)

)

...

)

### Create Lists

An argument to an action can be a call to create a list of arguments. For example,

pe\_action(list\_example, $create, arg1, arg2, arg3, create$)

This will generate:

(pe-action

:list-example

(create$

arg1

arg2

arg3

)

### Entity IDs

For any entity-type defined in your config file, you can specify **%***<entity-name>***ID%** on the RHS. For example, if your config file has:

<EntityType name="clientOffice" displayName="Client Office" typeID="0"…>

Then you could specify the following on the RHS

pe-action(clientIds, "%clientOfficeID%", …)

# Parameter Templates

This chapter provides overview of Parameter Editor feature of PowerEditor and guide for developers (i.e., MindBox consultants). It is not a user’s guide of Parameter Editor User Interface, and shall focus on what developers need to know. This document assumes that readers understands how PowerEditor works in general and have developed or worked with templates.

## Overview

Parameter Editor allows business users to maintain various piece of data that affect the behavior of a rule engine. We shall call such data a parameter. Parameters are primarily different from guidelines in the way they are deployed. Guidelines are deployed to the engine as rules. Parameters are deployed as object instances.

In general, parameters have much broader context and their values remains the same regardless input to the engine. Parameters are useful for setting global variables or numeric limits that needs to be visible to or maintained by business users.

### Parameter Templates

Like guidelines, a developer must first define templates for like parameters. We shall call these template parameter templates. To distinguish them from parameter templates, we shall call the templates that PE currently supports guideline templates.

Unlike guideline templates, parameter templates have no deployment rules. It contains column definitions, each of which is linked to an attribute of a class defined in Domain Definition file. This link is specified using the “attributeMap” attribute of <ColumnDefinition> tags. This is the same attribute used for mapping enumeration values for a column.

The class referenced in the “attributeMap” attribute must contain an attribute for each context element: channel, investor, and product. For example, if an attribute of the ClassA is referenced then ClassA must have an attribute for Channel, another one for Investor, and another for Product. The names of these context link attributes shall be “ChannelContext,” “InvestorContext,” and “ProductContext.” As with other names, the deploy label for these attributes will be used in generated rules. The class must also have an attribute for activation date and expiration date, with name “ActivationDate” and “ExpirationDate”, respectively.

### Context for Parameters

Parameters are associated with the same context as a guideline. That is, context for parameter templates consist of any combination of channels, investors, and products or categories.

Values for a parameter template are maintained as activations with varying activation and option expiration dates, just like guideline templates. An activation of a parameter template has the following attributes:

* Activation date
* Expiration date (optional)
* Status

### Deployment of Parameters

Parameters are deployed as either instances of objects (ARTScript file) or as executing a script. Note object instance deployment is the default behavior. Refer to Section 0 on how to specify deployment method.

#### Object Instance Generation

Values for parameter templates are deployed as instances of appropriate classes with the parameter values set as attributes. Each context element, as well as activation date range, are also set as an attribute of the instances.

Note that if a category that has no product in it is set as a context, the attribute for product will have “nil”.

#### Script Execution

If directed, PE will execute a script once per each parameter template, rather than deploying parameters for the template as object instances.

## Developing Parameter Templates

Parameter templates consist of one of more columns, each of which is associated with an attribute of a domain class defined in the Domain Definition file. Parameter templates and guideline templates have the same structure, other than the limitation described below. Parameter values business users specify for each column is set as the value for the reference attribute.

A collection of values for each column is deployed as a single object instance, each of which is mapped to its own attribute. Typically, each object instance corresponds to a single parameter grid row. This is not true when a category is selected as a parameter context and when the selected category contains two or more products. Such a grid row would be deployed in multiple object instances, the number of them equal to the number of products in the category.

### Parameter Template Limitations

When writing parameter templates, node the following limitations:

1. Parameter templates do not use deployment rules at neither template level or at a column level.
2. The following data types are NOT allowed for columns:

* CurrencyRange
* FloatRange
* IntegerRange
* Multi-selection EnumList (single-selection EnumList is supported)

1. Each column must specify “attributeMap.”
2. All attributeMap reference of a template must refer to the same class.

Limitation 2 enforces a single value to be mapped to an attribute when deployed. This eliminates unnecessary complexity to deployment logic. If muti-value attributes are desired, one can write a function to collect and consolidate multiple instances.

### AttributeMap Reference Limitations

In addition to these limitations, attributeMap of a column can only reference an attribute of a domain class that provides the following attributes:

* *one attribute for each entity type that can be in a parameter context.*
* ActivationDate
* ExpirationDate

Entity types are specified per the name of the field that is in the control pattern.

The last two for declaring activation date range. These attributes will be set for each deployed object instance, if and only if a parameter grid has a corresponding context. That is, if no channel is specified as context, no attribute for channel will be set.

Note no attribute is required for categories. Parameters that have a category in the context will be deployed with ids of products in the specified category. If no product is found, “nil” is output.

Like guidelines, ids used for context elements are generated based on the configuration settings in the PowerEditorConfiguration.xml file.

### Parameter Template XML Structure

The XML structure of parameter templates is identical to that of guideline templates, except that the root tag is <ParameterTemplateDefinition>, not <GridTemplateDefinition>.

See below for a sample parameter template XML. Figure 2 shows a sample object instances generated from the sample template.

### Deploy Method

Data of a parameter template can be deployed in two different ways: object instance and script execution. The deployment method for a particular parameter template is specified with a <DeployMethod> tag in a <ParameterTemplateDefinition> tag. If no DeployMethod tag is present, parameters will be deployed as object instances. So, if you want to execute a script per template, rather than deploying parameters for the template as object instances, you must provide the DeployMethod tag, as shown in code samples below.

## Figures

<ParameterTemplateDefinition id="200100" name="Fist Parameter Template"

status="Production" maxRows="10" >

<Description>This is the first parameter template</Description>  
 <!-- optional: uncomment and change the name of script   
 for scrip-execution deployment.

<DeployMethod>

<Script>c:\\temp\\sample-param-script.bat %templateID%</Script>

</DeployMethod>  
 -->

<ColumnDefinition colNum="1" description="Doc Type"

attributeMap="Transaction.DocumentationType">

<Presentation title="Doc Type">

<Font>arial</Font>

<Color>automatic</Color>

<ColWidth>180</ColWidth>

</Presentation>

<DataSpec type="EnumList" multipleSelect="No" allowBlank="No" >

</DataSpec>

</ColumnDefinition>

<ColumnDefinition colNum="2" description="Draw period"

attributeMap="Transaction.DrawPeriod">

<Presentation title="Draw period">

<Font>arial</Font>

<Color>automatic</Color>

<ColWidth>100</ColWidth>

</Presentation>

<DataSpec type="Integer" multipleSelect="No" allowBlank="Yes">

<MinValue>-500</MinValue>

<MaxValue>500</MaxValue>

</DataSpec>

</ColumnDefinition>

<ColumnDefinition colNum="3" description="Occupancy"

attributeMap="Transaction.SYMATTR">

<Presentation title="Occupancy">

<Font>arial</Font>

<Color>automatic</Color>

<ColWidth>100</ColWidth>

</Presentation>

<DataSpec type="EnumList" multipleSelect="No" allowBlank="Yes" >

</DataSpec>

</ColumnDefinition>

</ParameterTemplateDefinition>

Figure ‑ Sample Parameter Template Definition

(make-instance ldo:transaction

ldo:activation-date 2453008.5

ldo:channel-context 2

ldo:investor-context 1

ldo:documentation-type "NRFA"

ldo:draw-period 60

bfe:sym-attribute ThirdSymbol

)

(make-instance ldo:transaction

ldo:activation-date 2453005.5

ldo:product-context PCODE-1

ldo:documentation-type "NIFA"

ldo:draw-period 20

bfe:sym-attribute ValueTwo

)

(make-instance ldo:transaction

ldo:activation-date 2453037.5

ldo:channel-context 1

ldo:investor-context 1

ldo:product-context nil

ldo:documentation-type "SIFA"

ldo:draw-period 66

bfe:sym-attribute ThirdSymbol

)

(make-instance ldo:transaction

ldo:activation-date 2453005.5ldo:documentation-type "NIFA"

ldo:draw-period 1111

)

Figure ‑ Sample Parameter Template Deployed Object Instances

# PowerEditor Tools Reference

This section was updated for PowerEditor 4.5.0.

## Overview

This section provides overview of PowerEditor tools that are run outside of PowerEditor user interface. These are external tools for developers (i.e., MindBox consultants), not for business users. This section assumes that readers understands how PowerEditor works in general and have developed or worked with templates.

PowerEditor tool set consists of migration tools for various versions of PowerEditor and Date Repair Tool. You can access any of the tools by launching PowerEditor Tools application.

### Running PowerEditor Tools Application

1. Download PowerEditor distribution zip file.
2. Unzip the distribution file into a directory.
3. Open DOS-window and CD to <dist-dir>/powereditor/tools, where <dist-dir> is where you extracted the distribution file in Step 2. Make sure the following files in the same directory:

* powereditor-tools.bat
* powereditor-tools.jar
* log4j-1.2.8.jar

1. Make sure PowerEditor is not running, and no applications are using the PE database you’re about to modify.
2. If you want to connect to your database other than using ODBC, follow the instructions in this item: otherwise, skip to the next step.

* Download the JDBC driver zip file for your DB from [\\mbhqweb\softdist-public\editors\jdbc-driver](file:///\\mbhqweb\softdist-public\editors\jdbc-driver) directory.
* Then, place all .jar files in the zip file into the same directory where powereditor-tools.jar file is.

1. Enter the following at command to launch the tool.

> powereditor-tools.bat

If you are need to access non-ODBC database, execute the script specified in the following table:

|  |  |
| --- | --- |
| Database Connection | Script to execute |
| MySQL | java –classpath powereditor-tools.jar;mysql-connector-java-3.1.7-bin.jar;log4j-1.2.8.jar com.mindbox.pe.tools.PowerEditorTool |
| Oracle | java –classpath powereditor-tools.jar;ojdbc14.jar;log4j-1.2.8.jar com.mindbox.pe.tools. PowerEditorTool |
| SQLServer (w/o ODBC) | java –classpath powereditor-tools.jar;msbase.jar;mssqlserver.jar;msutil.jar;log4j-1.2.8.jar com.mindbox.pe.tools. PowerEditorTool |

Figure 1 shows the screen capture of PowerEditor Tools application.

### Specifying PowerEditor Database Settings

The top portion of the application is where you specify connection settings for PowerEditor database you want the application to access and modify. Before running any tool, you must first provide data connection settings. To do so,

1. Select JDBC driver. If you are using a JDBC driver not supplied with PowerEditor, please consult PE Dev group before continuing.
2. Enter Connection, user name, and password. Refer to “(Example)” for an example of a connection string for the selected JDBC driver.
3. Database configurations from previous sessions are available in the Saved Sessions dropdown. You may choose one from the dropdown.
4. Click “Test Connection” to verify that the information you provided is correct, if desired.

### 4.1.x to 4.2 Migration Tool

This tools migrates data in PE 4.1.x database into the format compatible with PowerEditor 4.2.

Before running this tool, you MUST update schema of your 4.1.x database for 4.2. Refer to release notes for version 4.3.0 for more details on how to update your database.

To run this tool,

1. Select the Migration: 4.x to 4.2 tab
2. Choose the entities you would like to migrate by clicking on any of the checkboxes named “Templates”, “Guideline Dates”, “Parameters”, “CBR”, and “Guideline Action Parameters”. If you are running this for the first time, it’s recommended that you select all check boxes.
3. Click the “Preprocess” button. The application will access DB and retrieve information necessary to complete the migration.
4. If you have chosen “Templates”, choose the “Templates” tab. If not, skip to Step 6.
5. The Template Version Mapping table may contain data. If so, be sure to provide version and parent ID of each template before continuing. If a template has no versions, leave parent ID empty.  
     
   If desired, press the “Set All To Default Version” to set the version of all templates in the table to the value specified in the Default Version combo field. You may want to change this value before clicking the button.   
     
   Parent ID must be a valid id. See column 1 for id of each template. The application attempts to set the parent ID when it fills the table. Please verify the parent id’s are set properly.
6. If you have chosen “Guideline Dates”, choose the “Guideline Dates” tab. If not, skip to Step 8.
7. The Guideline Activation Date Mapping table may contain data. If so, be sure to provide name and description for each date.
8. If you have chosen “Parameters”, choose the “Parameter Dates” tab. If not, skip to Step 10.
9. The Parameter Activation Date Mapping table may contain data. If so, be sure to provide name and description for each date.
10. If you have chosen “CBR”, choose the “CBR Dates” tab. If not, skip to Step 12.
11. The CBR Date Mapping table may contain data. If so, be sure to provide name and description for each date.
12. Click the “Migrate” button just below Migration 4.x to 4.2 tab.
13. According to the check boxes you have chosen in Step 2, you will be notified success of failure of migration of each data.

### Template Repair Tool

Template repair tool repairs template/column rules when a name of a domain attribute has changed. This tool repairs references to the modified domain attribute in template and column rules, in both LHS and RHS. This tool does not, however, modify references to the attribute in messages or in guideline action deploy rules.

To use,

1. Choose Template Repair tab below. See **Error! Reference source not found.**.
2. Enter the name of the attribute whose name has changed (old attribute name). The format is <domain-class-name>.<attribute-name>.
3. Enter the new name of the attribute. The format is <domain-class-name>.<attribute-name>.
4. Click on the Preprocess button. The table below is populated with id, column no (-1 for template level rule), name, version, and usage of templates that has a rule with a reference to the old attribute.
5. Click on the Migration button. A confirmation dialog, with number of templates modified upon success, is displayed.

### Grid Repair Tool

Grid repair tool corrects grid data in PowerEditor database when a structure of template changes. When a column is added or removed from a template, the grid data in DB becomes incompatible with the template. This tool inserts empty cell values to for added columns and removes cell values in deleted columns. This tool is for PowerEditor versions 3.3.x or earlier.

Note that Data Repair Tool is not necessary for PowerEditor version 4.x. Do NOT use this tool with PowerEditor version 4.0.0 or higher.

### Repairing Grid Data

1. Choose Grid Repair tab below.
2. Enter changes in template. For each template changed,
   1. Specify template ID
   2. Enter positions of added columns. Use 0 if a column is added before the existing columns (i.e., the new column becomes Column one). Use N if a column is added after column N (enter 2 if the new column becomes Column three). Enter column indeces separated by a comma. If more than one column is added to the same position repeat the same positions. For example, if you added two column before Column 1, enter 0,0. If you added three columns are Column 4, enter 4,4,4.
   3. Enter positions of deleted columns. Use 1 for Column 1, 2 for Column 2, and so forth.
   4. NOTE: if you are entering both added and deleted columns, specify added positions and deleted column number without considering added columns or remove columns. That is, if you added a column after Column 3 and deleted Column 1, enter 3 in the added column field and 1 in the deleted column field.
   5. Click Add.
   6. You may choose and edit previoulsy specified changes in a template, by selecting the appropriate row and clicking the Update button.
3. If you are satisified with the settings, click the Repair Data button. You will be prompted whether or not grid data were repaired successfully.

### Domain XML Tool

Domain XML Tool updates PE version 4.1 or earlier domain file into a format suitable for PE 4.2 or later.

To run,

1. Choose Domain XML Tool tab on top. See **Error! Reference source not found.**.
2. Select ‘Add DeployIDs to Eum’ and/or ‘Add Deploy Value to DomainClassLink’ check boxes, depending upon what you wish to do.
3. Select an input domain file.
4. Select the target domain file (output).
5. Click the Migration button. A confirmation dialog will be displayed.

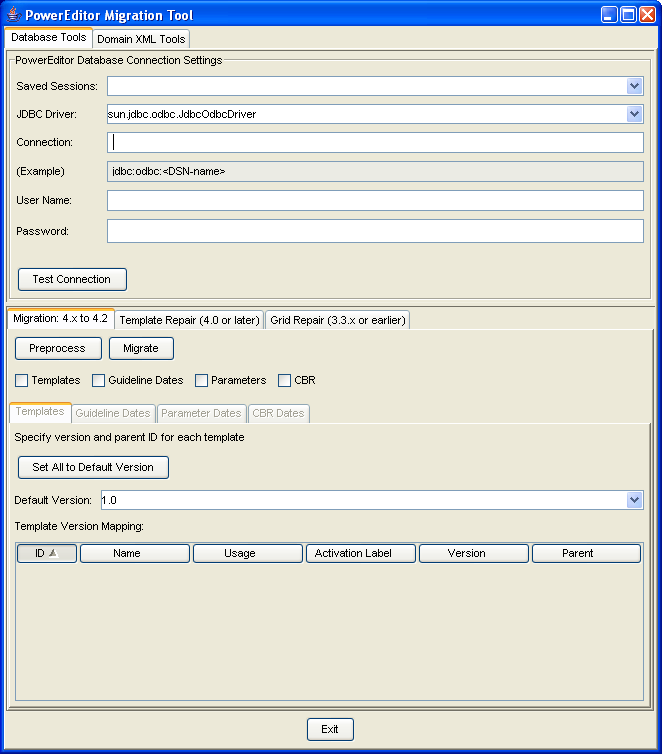


Figure ‑ PowerEditor Migration Tool (Migration: 4.x to 4.2 Tool shown)

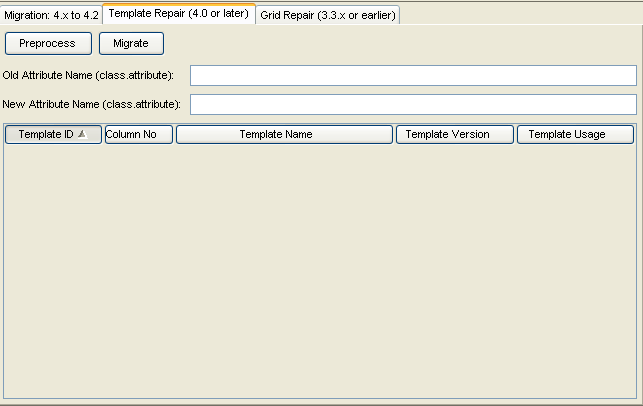


Figure ‑ Template Repair Tool

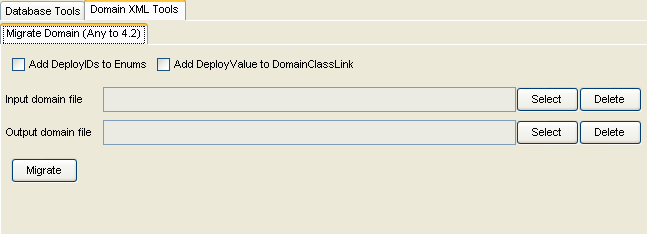


Figure ‑ Domain XML Tool

# Advanced Topics

## Time Slice

A time slice represent a period of time where no changes to policies/guidelines/entity or category relationships occur. That is, with a period of time that a time slice represents, policies in PE exhibit no changes. In PE changes occurs only on a date synonym. Hence, PE generates time slices using date synonyms. The number of time slices generated is equal to one plus the number of date synonyms in the system. For example, assume there are two date synonyms, DS1 for 12/01/2006 and D2 for 1/1/2007. Then, PE would generate three time slices one that goes from negative infinity to DS1 (TS1), another from DS1 to DS2 (TS2), and another from DS2 to infinity (TS3).   
  
Now, a given guideline activation may span across multiple time slices. Using above example, there might a guideline activation that goes from DS1 to infinity. This guideline applies to two time slices, TS2 and TS3.   
  
Normally, PE generates a single rule for a single guideline row, regardless of how many time slices a guideline activation applies to. But, if the guideline has a reference to a generic category, PE will generate more than one. The exact number of rules generated will depend on how many date synonyms are used to capture changes in category-to-category relationships over time. The exact number will be one plus the number of date synoymns used to capture category-to-category relationship that fall within the effective dates of a guideline. For example, if none of the date synonyms used to capture category-to-category relationship is outside of a guideline activation's effective dates, PE will generate one rule.

1. This is not a formal definition; for formal definition of exists-with expression, refer to Template Rule Specification Document [↑](#footnote-ref-1)