Deltek Know more. Do more.™

Deltek Maconomy 2.3 GA

Maconomy Dialog Language (MDL)

December 2, 2016

While Deltek has attempted to verify that the information in this document is accurate and complete, some typographical or technical errors may exist. The recipient of this document is solely responsible for all decisions relating to or use of the information provided herein.

The information contained in this publication is effective as of the publication date below and is subject to change without notice.

This edition published December 2016.

Contents

Overview

The Maconomy system contains a large number of dialog windows and printouts. Maconomy supplies tools to edit the layout of these dialogs and printouts. This Maconomy Layout Languages manual describes Window Design using MDL.

A separate section on preprocessing commands is provided in “MDL and MPL Preprocessor.”

This version of the manual describes the functionality of the layout editing tools supported by TPU 2.2.x.

MDL and MPL Preprocessor

This document describes the preprocessor functionality in MDL and MPL. This was introduced in TPU 53.

The preprocessor functionality is also available from M-Script. For more information please see the M-Script Language Reference Manual.

Introduction

The MDL and MPL preprocessor functionality allows code sections of MDL and MPL (both in standard prints and universe reports) to be dependent on add-ons as well as system parameters and system information. This means that you can, for example, define a dialog which only shows a certain island if a certain system parameter has been marked, or a printout which only shows certain information if a certain add-on has been installed.

The main reason for using this is to leave out parts of layouts, which are irrelevant in certain setups. This functionality may be used mostly by the Deltek R&D department, but can be used by any layout and MRL report developer.

Versions

The preprocessor is available in MDL and MPL as of TPU 53.

To be able to import MDL layouts using preprocessor directives, a Maconomy Windows client version 4.3.0 is necessary. The preprocessor will, however, work correctly on older clients, and MPL with preprocessor directives can be imported with older clients as well.

Application version 8.0SP11 is necessary for automatic recompilation of MPL after changes to system parameters and system information. This application version also renames all system parameters such that pseudo localization tags (@) are removed.

Preprocessor Options

The preprocessor is applied to MDL and MPL before the relevant compiler is invoked. This means that the preprocessor directives can occur anywhere in the MDL or MPL syntax.

Syntax

The syntax for preprocessor options is as follows:

```
#if <expression>
   ...
#endif
```

and

```
#if <expression>
   #else
#endif
```

The # directives must occur as the first token on a line.

`<expression>`

is one of

```
addon(<number>)
systemparameter.<systemparametername>
systeminformation.<systeminformationfield>
```

For systems running with Danish kernel language, this last option would be:

`systemoplysning.<systeminformationfield>`.

Note that a number of system parameters have an “@” as the first character. If this is the case, you must enclose the entire system parameter in a pair of backslashes, as illustrated in the following examples.

```
## WRONG:
#if systemparameter.@AllowChangeofVATOnInvoiceLines
   ...
## CORRECT:
#if systemparameter.\@AllowChangeofVATOnInvoiceLines\
   ...
```

The system parameter or the system information field must be of the type Boolean. Otherwise, an error message is produced (can be viewed in the `LayError.txt` resp. `PrintLayoutErrors.txt` file, which is placed in the Maconomy client folder). If an add-on, a system parameter or a system information field does not exist, it is treated as `false`.

Examples

Write a Text if an Add-On is Set

```
<island "Add-on">
   #if addon(65)
      "Add-on 65 is set"
   #endif
```

```
<end island>
```

Write a Text if an Add-On is Set, Otherwise Another

```
<island "Add-on">
#if addon(65)
   "Add-on 65 is set"
#else
   "Add-on 65 is not set"
#endif
<end island>
```

Write in Italics if an Add-On is Set

The preprocessor directives can be used everywhere in the layout.

```
"Hi"
#if addon(65)
   :italic+
#endif
```

Write a Text if a System Parameter is Set

```
#if systemparameter.UseDailyTimeSheets
   "We use daily time sheets"
#endif
```

Write a Text if a System Information Field is Set

```
#if systeminformation.DifferentialVAT
   "We use differential tax"
#endif
```

Negate a Boolean Criterion

There is no syntax for negating a Boolean criterion. Instead you write:

```
#if systemparameter.UseDailyTimeSheets
#else
   "We don’t use daily time sheets"
#endif
```

Warning

You should be aware that using the preprocessor functionality increases the risk of introducing illegal MDL/MPL layouts in the system. Consider the following fragment:

```
#if systemparameter.UseDailyTimeSheets
   "Add-on 65 is set"
#else
   <island "title> .unknownfield <end stack>
#endif
```

The fourth line contains three errors: Missing `"` after `title`, reference to an unknown field, and an attempt to match `<island>` with `<end stack>`.

Nevertheless, if the system is set up to use daily time sheets, this MDL will be validated with no problems. This is due to the fact that the preprocessor is invoked prior to invoking the MDL compiler, which would otherwise detect the problems.

Now suppose that the **Use Daily Time Sheets** system parameter is changed to `false`. Now the layout will suddenly be invalid, and users will not be able to open the window for which the layout is defined.

It is therefore recommended that all MDL/MPL is tested thoroughly before any preprocessor directives are inserted.

Recompilation

This section describes what goes on behind the scenes—it does, however, contain important information and is recommended reading.

Automatic Recompilation

Because MDL is compiled every time that a window is opened, changes in system parameters, system information, or add-on information are immediately reflected in the MDL. Likewise, MPL for Universe Reports is compiled when the report is executed, and does not need recompilation.

Automatic recompilation requires application version 8.0 SP11.

Standard MPL for prints, however, does need recompilation. For system parameters and system information fields, this is achieved in the following manner:

1. When the application detects that a system parameter or a system information field has been changed, all compiled MPL layouts are deleted. They are not immediately recompiled, because this will present the server with too heavy a load.
2. When a user opens a print window, all layouts that are marked as validated for the print in question are recompiled. If a layout cannot be recompiled due to errors, it is marked as non-validated and will not appear in the layout selection pop-up.

Hence, problems in MPL and MDL layouts are only detected when the layout is used. This stresses the point made previously that special care should be taken when writing MDL/MPL with preprocessor directives.

If an add-on is changed using MConfig, all layouts are recompiled. Note that it should be checked that no errors arose from this.

Currently, the MBuilder has no connection to the database. Therefore, system parameters and system information fields are not evaluated in preprocessor directives at installation time. Instead all these are assumed to be `true`. Hence, the `#else` branch is never checked.

Manual Recompilation

Automatic recompilation is not performed if add-on information, system parameters, or system information is changed by other means than the Maconomy client or MConfig. This could, for example, be the case if system parameters are imported into the system. In this case, it is important to manually run `MaconomyServer` with the options `-UP` and `-UVP`. This will ensure that the prints reflect the current settings of the system.

The consultant performing these changes might find it useful to also run `-UVP` to check if all user modified window layouts are still valid. We assume that all standard layouts are still valid. There is currently no way of validating installed MPL for Universe Reports.

Introduction

Maconomy Dialog Language (MDL) is a language that is used to describe the Maconomy window layout. In MDL you can specify the fields and texts of a window, and arrange the fields in columns and in islands. Based on this, Maconomy performs the final formatting of the window. By means of MDL it is thus possible to focus on the content and the logical construction of windows. A window description that is written in MDL is called a *layout*.

An MDL layout contains the logical structure of the window data. The actual placing of window elements is handled by Maconomy. A typical Maconomy window is described by stating:

- Which panes the layout consists of.
- Which islands the card pane consists of.
- Which texts and database fields each island consists of.
- Which columns the table pane consists of.

To determine the position in the window you specify which elements are to be placed next to or above other elements.

Readers who are already familiar with languages such as HTML or TeX will find that the basic idea of MDL is no different from these languages.

Prerequisites

To use this manual, it is required that you have a basic knowledge of Maconomy. In addition, you should be able to use an editor (for example, TextPad or Notepad) and Maconomy's layout windows as described in the section about the Set-Up module in the Maconomy Reference Manual.

This manual is meant as an introduction as well as a work of reference, meaning that it might at times be rather technical. On first reading, it might therefore be a good idea just to glance quickly through the reference section, because most concepts are explained later by means of examples.

Preprocessor

As of TPU 53, MDL supports the use of *preprocessor directives*. A preprocessor directive enables the layout programmer to check for the value of system parameters, the value of system information options, and the existence of add-on programs before the layout is displayed. Depending on whether, for example, a certain system parameter is `true`, the layout can display different information.

The preprocessor functionality applies both to MDL and to MPL, and is therefore described separately in “MDL and MPL Preprocessor” in this manual.

Getting Started

MDL is a tag-based language that consists of elements and attributes, similar to XML. Unlike XML, attributes in MDL have an associated type, and both tags and attributes can have a short form.

In addition, MDL is case-sensitive, meaning that the use of lower-case and upper-case letters is significant. All tags and attributes that are used in MDL must be written in lower-case letters. Names of database fields and variables are not case-sensitive.

Any tags that are mentioned in this section are explained in more detail in the reference section.

Installation

This manual concentrates on the MDL language. For a description of how MDL layouts are validated and imported into Maconomy, see the Set-Up module in the Maconomy Reference Manual and “Server Options” in the Maconomy System Administrator’s Guide.

Error Handling

Error messages are saved to the file `LAYERR.TXT` in the Maconomy folder at the time of validation. If you are using the Maconomy client for the Java™ platform, layout errors are shown in a separate browser window. If no errors occur when importing a layout, no file is generated. Error messages are generated in case of errors in the layout syntax or semantics. See “Error messages” for a list of each possible error message, with an explanation of each message, including any problem-solving suggestions.

Parenthetical and Simple Tags

Maconomy makes a distinction between *parenthetical* and *simple* tags. Parenthetical tags (or *block tags*) are characterized by the fact that they only come in pairs. For instance, the tag `<layout>` is parenthetical—it has a starting tag `<layout>` and an ending tag `<end layout>`. Neither can be omitted.

Simple tags are contained within a single set of angle brackets. All attributes are listed within the tag. Example: the tag `<title` *attributes*`>`. They are most often used in their short form.

Attributes

Attributes are MDL’s way of connecting data to tags, and are specified as name/value pairs inside tags:

attributename=attributevalue

Attribute names are always written in lower-case characters. With attributes, you can control the formatting and other aspects of the layout. In the reference section, all attributes are described with the tag where they are used.

Attribute Types

Every attribute has an associated type. The type of each attribute is specified in the description of each tag in the reference section. The table below describes each type. The attribute that illustrates each type is highlighted in **bold**.

Attribute Type	Description
ID	Attribute word. An attribute word is a unique identifier, consisting of a maximum of 100 alphanumeric characters and underscores, which has a certain meaning for a certain attribute. Example: `<var **data=MonthVar** width=12>` Here, the `data` attribute is of the type *ID*, where the attribute value is the name of a variable or a database field. An ID is a unique identifier, consisting of alphanumeric characters and underscores.
STRING	Text strings. A text is specified by placing a maximum of 100 characters between double quotes. If you want the text to contain quotes, write two double quotes right after each other in the text where you want the quotes to appear. Example: "He said, ""You can use double quotes""." The above text string will result in the following text: He said, "You can use double quotes". Example: <text title="Employee No.">
FSTRING	A text in single quotes—for example, `'/'`. See “Very Short Texts.”
BOOLEAN	Truth value – true or false. Example: <text title="Employee No." **italic=true**>
INTEGER	Integer value (theoretical max.: 21474836467). Example: <layout title="My layout" **split=200**>
REAL	Floating-point number. Floating-point numbers are specified by a period. Furthermore, floating-point numbers have a maximum of 3 decimals. This means that `1.0` is allowed, while `1.0000` is not allowed. Theoretical range: 1.7E308 to -1.7E308. Example: `<var data=MonthVar **width=12.5**>` You can specify integers instead of decimal numbers. This means that the following expression is allowed: <var data=MonthVar **width=12**>
PAIR	Values of the type *PAIR* are used for specifying coordinates. A pair is written as (*n*,*m*) where *n* and *m* are of the type *INTEGER*. Example: <layout title="Example" **position=(100,200)**>

Attribute Type	Description
TRIPLE	Values of the type *TRIPLE* are used for specifying the RGB value of colors. Triples are written as (*2.0*,*x2*,*x3*) where *2.0*,*x2*, and *x3* are of the type *REAL* (floating-point numbers). Integers can be used instead of floating-point numbers. Example: <text title="Example" **rgb=(100,50,100)**>

Nameless Attributes

Nameless attributes are attributes for which the name of the attribute is implied. The general rule is that the attribute name of a nameless attribute is implied by the type of the attribute value. This means that instead of:

```
<text title="Time Sheet Summary">
```

you can use:

```
<text "Time Sheet Summary">
```

The `"Time Sheets"` attribute is a *nameless attribute* because the attribute name `title` has been left out. In general, you can say that if an attribute is nameless, it is sufficient to specify the value of the attribute.

Example:

```
attribute_name=attribute_value
```

This attribute is nameless and it is therefore sufficient to use:

```
attribute_value
```

A maximum of one nameless attribute per type can be used per tag. For instance, the `tooltip` attribute cannot be nameless, because the `title` attribute has the same type (*STRING*) and can be used nameless in the `action` tag. Note that there may be a difference between which attribute is nameless in long or short form. For instance, `justification` is nameless in the short form of the tag `field`, whereas `data` is nameless in the long form.

The following table shows examples of tags where the attributes are not abbreviated, followed by the corresponding attributes used as nameless.

Full-Length Attributes	Nameless Attributes
`<layout title="Time Sheets">`	`<layout "Time Sheets">`
`<text title="Employee No." justification=right width=15.0>`	`<text "Employee No." right 15.0>`

See the description of each tag to see which attributes can be used as nameless.

Boolean Attributes

Boolean attributes are usually specified as follows:

```
attributename=true
attributename=false
```

However, Boolean attributes can have the following short forms:

```
attributename+
```

```
attributename-
```

For instance, you can abbreviate

```
<text title="A bold text" bold=true>
```

to

```
<text title="A bold text" bold+>
```

Units of Measure

Some attributes, such as `width`, require that you specify a number. The default unit of measure is *em,* meaning the width of the letter “M” in the selected font and font size. Since “M” (usually) is the widest letter of the alphabet, you can be sure that the space that you set aside in *em* will be sufficient for the text you specify whether the number of letters match the number of *em*s. Hence, the tag `<text title="A new text" width=10>` specifies that a text box with the text “My text” should be drawn, and that it should be 10 *em* wide.

Short Forms

For convenience, a short form notation exists for most simple tags in MDL. The short form is recommended over the long form, because it is more compact and readable. For instance, the tag `<text title="Hello world!">` is simply abbreviated to `"Hello world!"`. Short forms are different from tag to tag. The following reference section specifies both the long form and the short form of each MDL tag.

Comments and Whitespace

You can insert comments in your code by entering two hyphens. The rest of the line after the hyphens is ignored by the MDL compiler. See the following example. Remember to comment your code liberally to document it for the future.

```
-- Line comment – any text between the dashes and the next
-- newline is considered a comment.
```

Whitespace characters, that is blank characters, tab characters, and carriage returns, have no influence on the result of a MDL layout; only the logical structure has. Consider the following example:

```
<row>  "Name" Name1Var     <end row>
```

and

```
<row>
   "Name"
   Name1Var
<end row>
```

These examples result in the same position in the window (a row that contains a text string and a variable), because the `<row> ... <end row>` tag specifies the inclusion of a row, not whether the contents are represented on one or more lines.

How to Read the Syntax

In the following reference section, an example of the usage of the tag described is shown first. This is followed by a table that shows all of the available attributes to the tag. The table shows the type of the attribute, whether it is mandatory (“M”), whether it is nameless (“N”), and a description of the tag. If the attribute is mandatory or nameless, it is marked by a “+” in the table (or a “(+)” if special conditions apply). For a description of each attribute type, see “ Attribute types.”

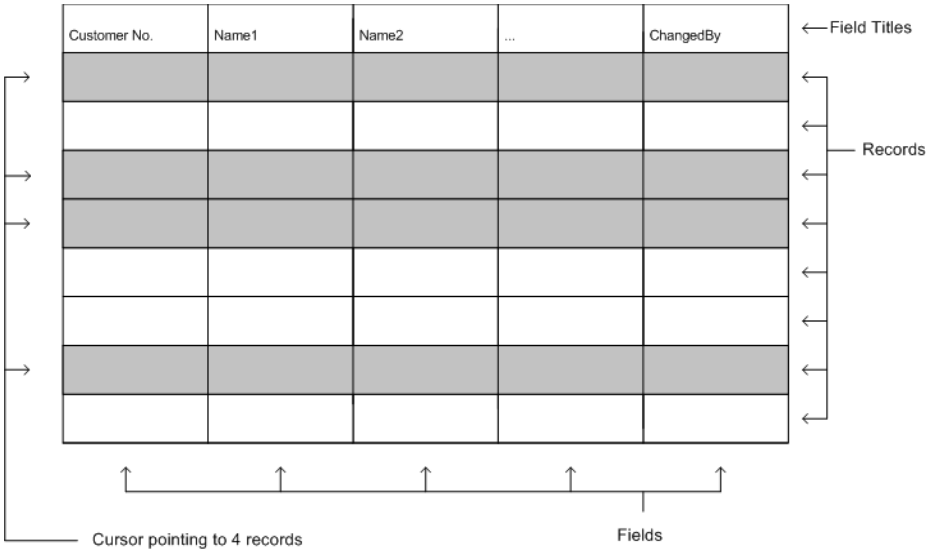
Before you use the reference section, you should be familiar with some general MDL concepts.

General Concepts

This section introduces terminology and concepts that are necessary for the understanding of this manual and for using MDL.

Database Fields and Variables

- **Relations** — Relations are collections of data in the Maconomy database. In this context, a relation can be perceived as a table of data. Such a table consists of a fixed number of columns and a varying number of rows. Each cell in a row contains data, and each column has a unique name, for example, CustomerNo. or Name1, signifying the contents of each column.
- **Fields and Records** — In SQL terminology, columns are called *fields*, and rows are called *records*. Thus a relation is a collection of records, each with a number of named fields containing data as shown in the figure below.



Maconomy is built on a number of different relations. For instance, one relation contains customers, and another relation contains vendors.

A *cursor* points out a number of the records that are contained in a relation. Sometimes a cursor points out all records, but often a cursor is subject to certain assigned *conditions*, which ensure that only specific records are included.

When using Maconomy, the system often displays values that are not contained in a relation. These values are calculated from data in the relations and values entered by the user. For example, the discount amount on an item line is calculated from the discount percentage and the extended price.

- **Variables** — The results of these calculations are put into *variables* that can be shown on the screen exactly like database fields.

Window Types

- **Parameter windows** — There are three types of windows in Maconomy that can be modified by MDL. Windows in which you specify selection criteria and perform an action are called parameter windows. Parameter windows typically result in a printout. Some

parameter windows are available from the Menu menu, and all windows that are opened using the function “Print...” in the File menu are parameter windows. Parameter windows can only contain variable fields.

- **Dialog windows** — Apart from parameter windows, all “normal” windows from the Menu menu are called *dialog windows*, such as the Time Sheets or Customer Information Card windows. Dialog windows can contain variables as well as database fields.
- **Search windows** — When you press Ctrl+F to perform a search in a dialog window, a search pane opens. Even though the search pane is integrated into the dialog window, it has a separate MDL specification. Pressing Ctrl+G opens a separate search window. For more information, see “Panes.”

Panes

All of Maconomy’s windows are constructed of *panes*. A window contains one, two, or three panes. If there is more than one pane in a window, the panes are separated by a split line that can be moved by the user. There are three kinds of panes that can be described as follows:

- **Card panes** — In a *card pane* (or tab), the information is presented as fixed texts followed by fields. A card pane usually corresponds to the content of an entry (record) in a database relation, but it frequently also contains variables. Card panes are usually arranged in *islands* that have a logical grouping of texts and fields. An island has a frame and usually a heading. The islands are placed in one or two columns in the pane.
- **Table panes** — In a *table pane* (or sub-tab), the information is presented as a table that has columns, each of which has a heading. A table pane conceptually corresponds to a database cursor, and each row corresponds to an entry (record) in the cursor.
- **Search panes** — In a *search pane*, the information is presented as a table that has columns, similar to the table pane. In the Maconomy client, the user can choose to see more or fewer columns from the list of columns that are specified in the MDL layout. By default, the first ten columns are shown. Search panes have separate MDL layouts, and are not part of the dialog window layout.

MDL Panes

In MDL terminology, the search pane is described as a table pane. For the card pane, a distinction is made between single and double column panes. In a single-column pane, all of the pane elements are placed in one column in the pane. In double-column panes, the elements are placed in two columns. Single-column panes contain *boxes*; double-column panes contain *groups* (see the next section). Only single-column panes in parameter windows and in dialogs that have only one card pane can be empty. All pane types are described by a pane tag and an end-pane tag. Layout elements between these tags are placed in the specified pane.

Groups as formatting elements do not make sense in connection with single-column panes. Note that single-column pages must contain at least one box. However, parameter dialogs and card dialogs can contain an empty single-column pane.

Note that when you specify a pane in a parameter window, Maconomy draws the Cancel and Print buttons. Search windows are always one table pane, where Maconomy draws the rest.

See more information in the description of the pane tags in the reference section.

Window Formatting

You can set the default way in which panes and windows are to be formatted with regard to size and position, and so on. Apart from the `widthfactor` (connected to the `dualpane` tag), these

attributes only apply the first time that the window is opened, because Maconomy saves the current size when the window is closed. For more information, see the description of the `layout` tag.

Window Layouts in Maconomy

Maconomy is shipped with one or two original layout(s) for every window. There is always a *standard* layout, and sometimes there is a *solution* layout (which applies to one of the Maconomy solutions).

Often you want to design a new window in such a way that it resembles the original layout; you might want to move a few fields or islands around, leave out some fields, or change some fixed texts. In this case, the adjustment can be made in a few minutes by using the original layout for the window in question. This is done by exporting the “standard” or “custom” layout and making the adjustments in a text editor. When an original layout is exported, the MDL layout is automatically named “Original.”

Even if the window layout has to be changed radically, it is usually a good idea to use the original layout as a template. This is because it makes it easy to find out which variables and database fields are available, because they are all included in the original layout.

For a description of exporting layouts, see the description of the Window Layouts window in the Set-Up module in the Maconomy Reference Manual.

Reference

This section contains a description of the general structure of an MDL layout and a description of each MDL tag and its possible attributes. For each tag, a table of attributes is shown with name, type, description, and indication whether the attribute in question is mandatory (“M”) or nameless (“N”). In addition, a subsection of the BNF structure of the tag in question is shown. For the full BNF structure, see “MDL Syntax.”

Note that a number of attributes apply to the Maconomy client for the Java™ version only. When such attributes are encountered by other Maconomy clients, they are silently ignored.

This section contains all of the information that you need to design your own layouts. Examples of layouts are discussed in “Layout Examples.”

General Structure

All layouts begin with a special *root tag*:

```
<mdl 4>
```

In addition to the root tag, MDL specifications have the following structure:

```
<mdl>
<layout>
   <pane>
            ...
   <end pane>
   <pane>
      ...
   <end pane>
<end layout>
```

Remember that tag names must be in lower case. Each of the tags that are mentioned previously is shown without attributes.

mdl

The `mdl` tag is the root tag and can be written as:

```
<mdl 4>
...
```

Attribute	Type	M	N	Description
N/A	*INTEGER*	+	+	MDL version. Can be less than 4, but `4` is recommended.

This tag is special in that it does have an attribute, but the attribute can only be specified as nameless.

The current version of MDL is backward-compatible with earlier versions of the language.

Structure

start = `<mdl` *INTEGER*> *layout* Version no. 1-4.

layout

The `layout` tag is a parenthetical tag. It follows immediately after the `mdl` tag and can be written as:

```
<mdl 4>
<layout>
    ...
<end layout>
```

Attribute	Type	M	N	Description
findsuppressed	BOOLEAN			If set and `true`, the Navigate menu is disabled in the current window in the Java™ client.
initialfocusincard	BOOLEAN			If set and `true`, when you open the window in the Java™ client the initial focus is in the card part, and not in the table part which is the default.
name	ID	(+)	+	MDL 4. This attribute is mandatory in MDL 4 and above. The name is the internal identifier of the current layout. Example: `name=Key_Employees`
position	PAIR			Specifies the position of the window on the screen the first time the window is opened. The attribute is specified as follows: `position=(`*`x,y`*`)`, where the coordinates are specified in pixels (integers). If you do not specify this attribute, a suitable position is automatically calculated.
size	PAIR			Specifies the size of the window on the screen the first time the window is opened. The attribute is specified as follows: `size=(`*`width,height`*`)`, where the coordinates are specified in pixels (integers). If you do not specify this attribute, the size is automatically calculated on the basis of the window contents.
split	INTEGER			Only used for layouts with two panes. It specifies the position of the split line the first time the window is opened. The attribute is specified as follows: `split=`*`height`*, where the height is an integer specifying the number of pixels from the top of the window. If the window size does not allow the split line to be placed as specified, the line is moved upward until it fits.
toolbarsuppressed	BOOLEAN			If set and `true`, the toolbar is removed in the current window in the Java™ client.

Attribute	Type	M	N	Description
title	STRING	+	+	The `title` attribute is mainly used for specifying texts to be shown on the screen. However, in the `layout` tag the title is not printed to the screen, but is only used for reference. The title is displayed in some of the window layout dialogs in Maconomy. You cannot use under- scores (_) in the title, and the title cannot begin or end with spaces. MDL 4: As opposed to the `name` attribute, this attribute is external, can be localized and edited. Example: `title="My new layout"`
windowname	ID	(+)		MDL 4. This attribute is mandatory in MDL 4 and above. The window name is the internal name of the window with which the current layout is associated. If you attempt to import a layout for a different window than what is specified here, the import will fail. Example: `windowname=Jobs`
windowtype	ID	(+)		MDL 4. This attribute is mandatory in MDL 4 and above. Specifies the type of the window to which the current layout applies. If you attempt to import a layout for a window of a different type than what is specified here, the import will fail. You can choose between the following options (types of window): `DialogWindow`, `SelectionWindow`, `ParameterWindow` (print window) and `Report` (Analyzer report).

Example

```
<mdl 4>
<layout Original "Original"
   windowtype=DialogWindow windowname=BudgetInspection>
   ...
<end layout>
```

Structure

Tag	Structure	Comments
layout	= <layout *attribute*$_1$ .. *attribute*$_n$> *availableactions* *availableshortcuts* *pane* <end layout>	*availableactions* and *availableshortcuts* can be omitted.
	| <layout *attribute*$_1$ .. *attribute*$_n$> *availableactions*	*pane*$_1$ must be *singlepane* or *dualpane*, *pane*$_2$ must be *tablepane*.

Tag	Structure	Comments
	availableshortcuts *pane*$_1$ *pane*$_2$ <end layout>	

availableactions

Affects the Java™ client only. In windows that contain actions, you can define which of the actions should be available for selection in the current window. By default, all actions that are specified by Maconomy developers are available, as are any actions that are specified using the Maconomy extension language MEXL. However, if you specify available actions using MDL, only the actions specified here are available (or unavailable, depending on the attribute `exclude`).

The `availableactions` tag is a parenthetical tag and can be written as:

```
<mdl 4>
<layout>
   <availableactions>
      <action>
      ...
   <end availableactions>
   ...
```

Attribute	Type	M	N	Description
exclude	BOOLEAN			This attribute is by default `false`, meaning that the actions specified in the `<action>` tags below appear in the Action menu of the current window. If `exclude` is set to `true`, the actions specified in the `<action>` tags below are not available in the window.

Example

See the `action` tag for an example.

Structure

Tag	Structure	Comments
availableactions	= <availableactions> *action*$_1$ .. *action*$_n$ <end availableactions>	*availableactions* need not contain any actions.

action

Affects the Java™ client only. The simple `action` tag can be written as:

```
<mdl 4>
<layout>
   <availableactions>
```

```
      <action>
      ...
   <end availableactions>
   ...
```

Attribute	Type	M	N	Description
actionname	ID	+	+	This is the internal name of the action as specified in MEXL.
title	STRING		+	This string specifies the menu title of the current action. If it is not specified, the title is the default title as specified in MEXL.
tooltip	STRING			This string specifies the tooltip shown when the mouse hovers above the action in the menu (or the button if the action is defined as a button).

Examples

The following examples are from the Time Sheets window.

Example 1

```
--The Action menu contains the actions specified in DDL/MEXL:
<mdl 2>
<layout "Standard">
   <dualpane>
      <group>
      ...
```

Example 2

```
-- The available actions are "Release_TimeSheet", "Copy_TimeSheet", and "B"
(where "B" is a Custom Action):
<mdl 2>
<layout "Standard">
   <availableactions>
      <action actionname=Release_TimeSheet title="Release TS">
      <action B title="B Button">
      <action Copy_TimeSheet>
   <end availableactions>
   <dualpane>
      <group>
      ...
```

Example 3

```
-- The actions "Release_TimeSheet" and "Copy_TimeSheet" are removed
-- from the list of available actions:
<mdl 2>
<layout "Standard">
   <availableactions exclude+>
```

```
      <action actionname=Release_TimeSheet>
      <action Copy_TimeSheet>
   <end availableactions>
   <dualpane>
      <group>
      ...
```

Structure

Tag	Structure	Comments
action	= <action *attribute*$_1$ .. *attribute*$_n$>	

availableshortcuts

Affects the Java™ client only. In windows that contain actions you can define shortcuts for one or more of the available actions. For each shortcut, you can specify a shortcut key within the range a..z, A..Z, and 0..9. Note that function keys (F1-F12) cannot be specified as shortcut keys. Note also that Maconomy does not validate your choice of shortcut key. If there is a conflict with an operating system shortcut key, the operating system prevails.

The `availableshortcuts` tag is a parenthetical tag and can be written as:

```
<mdl 4>
<layout>
   <availableshortcuts>
      <shortcut>
      ...
   <end availableshortcuts>
   ...
```

It does not take any attributes.

Example

See the `shortcut` tag for an example.

Structure

Tag	Structure	Comments
availableshortcuts	= <availableshortcuts> *shortcut*$_1$ .. *shortcut*$_n$ <end availableshortcuts>	*availableshortcuts* need not contain any shortcuts.

shortcut

Affects the Java™ client only. The simple `shortcut` tag can be written as:

```
<mdl 4>
<layout>
   <availableshortcuts>
      <shortcut>
```

```
   ...
<end availableshortcuts>
...
```

Attribute	Type	M	N	Description
actionname	*ID*	+	+	This is the internal name of the action as specified in DDL/MEXL to be executed when the shortcut is pressed in the client.
key	*STRING*	+	+	This is the shortcut key. One letter or digit only in the range a..z, A..Z, and 0..9.
shift	*BOOLEAN*			If specified, the Shift key must be pressed with the shortcut key defined in `key`.
alt	*BOOLEAN*			If specified, the Alt key must be pressed with the shortcut key defined in `key`.
ctrl	*BOOLEAN*			If specified, the Ctrl key must be pressed with the shortcut key defined in `key`. On the Macintosh, the Ctrl key maps to the Apple Command (⌘) key.

Example

The following example is from the Time Sheets window. Alt+S submits the current time sheet, and Ctrl+R executes the **Reopen Time Sheet** action:

```
<mdl 4>
<layout TSShortcuts "Time Sheets Shortcuts" windowtype=DialogWindow
windowname=TimeSheets>
   <availableshortcuts>
      <shortcut Submit "S" alt=true>
      <shortcut key="R" ctrl+ actionname=Reopen_TimeSheets>
   <end availableshortcuts>
   <dualpane>
      <group>
      ...
```

Structure

Tag	Structure	Comments
shortcut	= <shortcut *attribute*$_1$ .. *attribute*$_n$>	

singlepane

Specifies a single-column card part. The parenthetical `singlepane` tag can be written as:

```
<mdl 2>
<layout>
   <singlepane>
```

```
      <island>
      ...
   <end singlepane>
   ...
```

It does not take any attributes.

Example

For several examples of the use of `singlepane`, see “Layout Examples.” Note that `singlepane` is also the preferred tag for creating windows that have more than two columns. For an example of this, see “Islands in more than Two Columns.”

Structure

Tag	Structure	Comments
singlepane	= <singlepane> *Buttons* *Dropdownbutton* *bo2.0 .. box$_n$* <end singlepane>	*singlepane* need not contain any boxes, buttons, or dropdown button.

dualpane

Specifies a double-column card part. The parenthetical `dualpane` tag can be written as:

```
<mdl 2>
<layout>
   <dualpane>
      <island>
      ...
   <end dualpane>
   ...
```

Attribute	Type	M	N	Description
widthfactor	*REAL*		+	Specifies the width of double-column panes. Width is specified in relation to the standard width: If, for instance, you want the pane to be 20% larger than the standard size, you can write `widthfactor=1.2`. If the pane is to be 20% smaller, use the expression `widthfactor=0.8`. The width of a dual pane has influence on the format of the groups in the pane; particularly for the floating of the elements (typically for the way in which islands are placed). The attribute value for `widthfactor` is *REAL* or *INTEGER* (as for the `width` attribute).

Example

For several examples of the use of `dualpane`, see “Layout Examples.”

Structure

Tag	Structure	Comments
dualpane	= <dualpane attribute$_1$ .. attribute$_n$> *buttons* *dropdownbutton* *group$_1$ .. group$_n$* <end dualpane>	*dualpane* need not contain any groups , buttons, or dropdownbutton.

tablepane

Specifies a table part. The parenthetical `tablepane` tag can be written as:

```
<mdl 2>
<layout>
   <tablepane>
      <field>
      ...
   <end tablepane>
   ...
```

Attribute	Type	M	N	Description
hidelineid	*BOOLEAN*		+	If the window uses hierarchical table structure, you can set this attribute to `true` if you do not want to show the line ID of each line. The line ID only makes sense in hierarchical table structures. Note that if `hidelineid` is `true`, the ability to show or hide the line ID in the client interface disappears.

Example

For several examples of the use of `tablepane`, see “Layout Examples.”

Structure

Tag	Structure	Comments
tablepane	= <tablepane *attribute*> *Buttons* *Dropdownbutton* *column$_1$ .. column$_n$* <end tablepane>	*tablepane* must contain at least one column. Buttons and dropdownbutton can be omitted.

Buttons

You can create buttons in MDL that execute actions. The buttons are only displayed in the Maconomy client for the Java™ platform.

Buttons can be placed as the first element in the three *pane* tags: `dualpane`, `singlepane`, and `tablepane`. The buttons are displayed at the top of the pane, either the card part or the table part. By defining buttons, you make it easier for the user to select actions in the window.

A special type of button, the drop-down button, collects a number of actions in one button. The user first clicks the button and then selects an action.

The actions that are specified as buttons must be included in, or not excluded by, the `<availableactions>` tag in the beginning of the layout, if it exists. In addition, a number of other actions are shown as buttons, but are beyond the control of MDL:

- In the card part, a “Save” button is shown.
- If navigation is enabled by the Maconomy developer, buttons for navigating through records, a “New” button and a “Delete” button are added in the card part.
- In the table part, table action buttons that correspond to the File menu commands “Insert Line,” “Add Line,” and “Delete Line” are added.
- In addition, if the window uses hierarchical table structure, buttons that correspond to the Edit menu commands “Move Line Up” and “Move Line Down” are shown.

The `<buttons>` tag is a parenthetical tag and can be written as:

```
<mdl 2>
<layout>
   <dualpane>
      <buttons>
         <action>
         ...
      <end buttons>
   <end dualpane>
...
```

It does not take any attributes.

Example

The following example is from the Time Sheets window.

```
--Three buttons
<mdl 2>
   <layout "Standard">
      <dualpane>
         <buttons>
            <action actionname=Release_TimeSheet title="Release TS"
            tooltip="Release this time sheet">
            <action Reopen_TimeSheet title="Reopen timesheet" tooltip="Reopen
            this time sheet">
            <action Copy_TimeSheet>
         <end buttons>
         ...
```

Structure

Tag	Structure	Comments
buttons	= <buttons> *action*$_1$ .. *action* <end buttons>	*buttons* need not contain any actions.

dropdownbutton

This parenthetical tag contains one or more button definitions in the form of actions. The buttons are only displayed in the Maconomy client for the Java™ platform.

A dropdown button can be placed as the first element in the three *pane* tags: `dualpane`, `singlepane`, and `tablepane`. In the Java client, this is shown as one button that has a small arrow on the right. Clicking the arrow opens a list of the actions that are defined for the button. There is no default action for the button. The dropdown button is positioned to the right of any normal buttons. By defining a dropdown button, you can save space in your layout.

The actions that are specified in the dropdown button must be included in, or not excluded by, the `<availableactions>` tag in the beginning of the layout, if it exists.

The `<dropdownbutton>` tag is a parenthetical tag and can be written as:

```
<mdl 4>
<layout>
   <dualpane>
      <dropdownbutton>
         <action>
         ...
      <end dropdownbutton>
   ...
```

Attribute	Type	M	N	Description
title	*STRING*		+	The title of the button when it is closed (not “dropped down”).

Example

The following example is from the Time Sheets window.

```
--Two buttons and a dropdown button with three actions
<mdl 2>
   <layout "Standard">
      <dualpane>
         <buttons>
            <action actionname=Release_TimeSheet>
            <action actionname=Reopen_TimeSheet
         <end buttons>
         <dropdownbutton title="Actions">
            <action actionname=Approve_TimeSheet title="Approve Time Sheet">
            <action Copy_TimeSheet title="Copy Time Sheet">
```

```
      <action Transfer_Plan>
<end dropdownbutton>
<group>
...
```

Structure

Tag	Structure	Comments
dropdownbutton	= <dropdownbutton> *action*$_1$ .. *action*$_n$ <end dropdownbutton>	*dropdownbutton* need not contain any actions.

Group

A group does not correspond to any window element, but is only used to control the formatting of panes that have two columns (see `dualpane`).

If a pane that has two columns is read from left to right, the pane elements (typically islands) appear exactly in the order in which they appear in the MDL layout. More explicitly:

1. The first element of each group is placed to the left.
2. Subsequent elements are placed below the lowest column if the column width is large enough; otherwise they are placed to the left under both columns.

This process is called *floating*.

The parenthetical `group` tag can be written as:

```
<mdl 2>
<layout>
   <dualpane>
      <group>
         <island>
         ...
      <end group>
      …
   <end dualpane>
   ...
```

It does not take any attributes.

Example

For several examples of the use of `group`, see “Layout Examples.”

Structure

Tag	Structure	Comments
group	= <group> *bo2.0* .. *box*$_n$ <end group>	*group* must contain at least one box.

text

The most simple tag of MDL layouts describes text elements. The following expression, for instance, describes the text element `Employee No.`, which will be shown in the dialog window in accordance with the remaining attributes and enclosing formatting tag:

```
<text title="Employee No.">
```

Attribute	Type	M	N	Description
title	*STRING*	+	+	This nameless attribute specifies the title of the text element, that is the actual text written to the screen.
text formatting				A number of text formatting attributes are available: ▪ bold ▪ color ▪ fontname ▪ fontsize ▪ italic ▪ justification ▪ rgb ▪ showasclosedfield ▪ stretch ▪ underline ▪ underline

Short Form

To further simplify the syntax of texts, MDL enables you to leave out the tag completely. If you write a string where a box is expected, Maconomy will regard it as a text.

Example: `"Employee No."` is regarded as

```
<text title="Employee No.">
```

Attributes are separated by colons. In the case of texts, `justification` and `width` are nameless. The following text uses all attributes:

```
<text title="My text"
   justification=center
   width=10.0
   stretch=high
   bold=true
   italic=true
   underline=true
   color=blue
   fontname=roman
   fontsize=12>
```

In the short form, the preceding expression will look like the following (the short form for Boolean attributes is also used in this example):

```
"My text":center:10.0:stretch=high:bold+:italic+:underline+:
color=blue:fontname=roman:fontsize=12
```

Very Short Texts

You often want a text to be as short as possible. This is especially the case when using separation characters such as '/' in a date specification or '-' in a range specification. In these cases, you might also want to center the separation characters. This can be done in the following way:

```
<text title="/" width=0.0 stretch=low justification=center>
```

Because this is used frequently, it also exists in a short form:

```
'/'
```

The `justification`, `width`, and `stretch` attributes cannot be used in connection with this short form; other attributes are specified as in the usual short form for texts.

Structure

Tag	Structure	Comments
text	= <text *attribute*$_1$ .. *attribute*$_n$> | *STRING:attribute*$_1$*: .. :attribute*$_n$ | *FSTRING:attribute*$_1$*: .. :attribute*$_n$	Long form. Double-quote text. Single-quote text.

field

A `field` tag is a reference to a database field. Fields can be used in both the card pane and in the table pane. However, in the card pane the field label is specified as a separate `text` tag (or using the `title` tag. See `title`.). In the table pane, the field label (that is, the column heading) is specified directly in the field tag as an attribute.

If, for instance, you want to show a field in the card part containing the database field `EmployeeNumber` and labeled with the text "Employee No.," use the following expression:

```
<text title="Employee No.">
<field data=EmployeeNumber>
```

A similar field in the table part would look like the following (where `title` is the column heading):

```
<field title="Employee No." data=EmployeeNumber>
```

Attribute	Type	M	N	Description
data	*ID*	+	+	This nameless attribute specifies the database field to be source of content of the field.
fieldtitle	*ID*		(+)	This nameless attribute specifies the column heading of the field when used in the table part as the content of a field in the card part. Requires MDL version 2 or above.

Attribute	Type	M	N	Description
mandatory	*BOOLEAN*			This attribute is used for specifying that the user *must* fill in a database field or a variable. If the field/variable is open in *new* mode (see below), you cannot leave the *new* mode unless data is entered in the field/variable. If the field is open in *update* mode, and the user presses Enter, Maconomy will check that the field was not *changed* to be empty (cleared). In this way, the use of `mandatory` will ensure that fields that are open in *new* mode always contain a value. In connection with fields that are only open in *update* mode, `mandatory` only makes sense if the system completes the field with a value (a check will then be performed that this value is not deleted). The attribute can be used for ensuring that important information is entered. For instance, you might want to require a central national register number (social security number) to be entered when creating a new employee in Maconomy. This functionality is ensured by writing: `<field data=CNRNumber mandatory=true>` Note that the `mandatory` attribute cannot be used in connection with check boxes and pop-up fields.[1] Certain database fields are already made mandatory by Maconomy, and this cannot be changed. It is not possible to make closed fields mandatory, as the user will not be able to enter data as required. However, it is possible to close fields which are mandatory by Maconomy, as long as the format contains one occurrence of the field which is open in *new* mode, if the field is open in *new* mode by system default.
readonly	*BOOLEAN*			Because parameter windows have no *new* and *update* modes, the `ronew` and `roupdate` attributes (see below) have no meaning here. Instead, the `readonly` attribute, which closes variables for entry, is used (parameter windows can only contain variables).

[1] A check box always has a value (selected or not) and is therefore always mandatory. Certain pop-up fields are made mandatory by Maconomy. These pop-up fields are characterized by the fact that the blank value cannot be chosen. If MDL made it possible to make a pop-up field mandatory, the blank value would disappear—even in already entered data—resulting in potential conflicts.

Attribute	Type	M	N	Description
ronew	*BOOLEAN*			"ronew" is short for Read Only New and is used for specifying that no information can be entered in a field in the *new* mode. In Maconomy, this is, for instance, the case for all database fields and variables in the Time Sheets dialog except for `EmployeeNumber`, `WeekVar`, and `YearVar`. The `ronew` attribute can close fields that are usually open, but not the other way around. This attribute can only be used in dialog windows. In parameter windows, the `readonly` attribute is used (see above).
roupdate	*BOOLEAN*			"roupdate" is short for Read Only Update and is used for specifying that no information can be entered in a field in the *update* mode. In Maconomy, this is, for instance, the case for `EmployeeNumber`, `WeekVar`, and `YearVar` in the Time Sheets dialog (that is, the only database fields/variables that were not `ronew`).
summed	*BOOLEAN*			Table part of Java client only. If this attribute is set and `true`, a sum field is shown below the current table column, summing the fields above it, provided that the fields are of the type *REAL*, *INTEGER*, or *AMOUNT*. This is for instance used in the window Time Sheets in the Java client.
title	*STRING*		(+)	This nameless attribute specifies the column heading of the field when used in the table part.
variabletitle	*ID*		(+)	This nameless attribute specifies the column heading of the field when used in the table part. The heading is the content of a variable in the card part. Requires MDL version 2 or above.
visualizeastime	*BOOLEAN*			Java client only. If this attribute is set and `true`, the content of the field is formatted as time, respecting the "Interpret as minutes if above" setting in the Preferences window, provided that the field is of the type *REAL*. In the client, you can for example enter "90" and Maconomy will interpret it as 1 hour 30 minutes. This is for instance used in the SpeedSheet in the Java client.
zerosuppression	*BOOLEAN*			With this attribute, you can specify that zero values should be suppressed (omitted) from fields in the card part or the table part, provided that the fields are of the type *REAL*, *INTEGER*, or *AMOUNT*. Note that you cannot specify `mandatory` and `zerosuppression` for the same field at the same time.

Attribute	Type	M	N	Description
text formatting				A number of text formatting attributes are available: ▪ `bold` ▪ `color` ▪ `fontname` ▪ `fontsize` ▪ `italic` ▪ justificationon ▪ `rgb` ▪ `showaslabelifclosed` (in card part only) ▪ `stretch` (in card part only) ▪ `underline`

Exactly one of the attributes `title`, `fieldtitle`, and `variabletitle` can be used in the `field` tag when the tag is used in the table part. When the tag is used in the card part, use a text as a fixed text label.

If you omit the title specification in the table part:

```
<field data=JobNumber>
```

the heading of the column will be the Maconomy internal default title for the job number field.

A database field can be used several times in the same card pane, but only if at most one of the field occurrences in the pane is open in each mode (see “Modes”).

In the table part, you can also use a field or variable value from the card part as a column heading. This allows you to for example create a column whose heading corresponds to the date currently shown in a given field in the card part. If the date changes, either as a result of changing the value or browsing to another card part entry, the column heading changes accordingly. See the preceding attribute table. Example:

```
<field fieldtitle=fieldname data=JobNumber>
```

where *fieldname* is replaced by the name of a field that is available in the database relation that is used in the card part of the window. The preceding example results in a table column where you enter job numbers, but the heading of the column depends on the value of the specified field in the card part.

Usually, the same database field cannot appear more than once in the same table pane.

The description of the `mandatory`, `ronew`, `roupdate` and `readonly` attributes mentions the concept of *modes*. For more information about modes, see “Modes.”

Short Form

In the card pane, the short form for database fields consists of the database field with a leading dot, meaning that:

```
<field data=EmployeeNumber>
```

can be written as:

```
.EmployeeNumber
```

The dot tells MDL that this is a database field.

In the table pane, the short form for database fields is designed to resemble a fixed text (the heading) followed by a field.

Thus, the expression:

```
<field title="Job No." data=JobNumber>
```

can be written as:

```
"Job No." .JobNumber;
```

Note the semicolon that indicates the end of the short form. Or you can just enter:

```
.JobNumber;
```

if you want to use the default field title as heading.

Attributes are separated by colons, and are specified after the database field name.

Structure

Tag	Structure	Comments
field (card part)	= <field *attribute*$_1$ .. *attribute*$_n$> | *.ID:attribute*$_1$*: .. :attribute*$_n$	Long form. Short form.
column (table)	= <field *attribute*$_1$ .. *attribute*$_n$> | *STRING .ID:attribute*$_1$*:..:attribute*$_n$*;* | *ID .ID:attribute*$_1$*:..:attribute*$_n$*;* | *.ID .ID:attribute*$_1$*:..:attribute*$_n$*;*	Field column, long form. Field column with text heading from variable. Field column with heading derived from field.

Var

You can refer to MDL variables by using the `var` tag. This is done in exactly the same way as with database fields. If you want to have a field that contains, for instance, the variable `DepartmentNumberVar`, the following expression can be used:

```
<var data=DepartmentNumberVar>
```

For possible attributes, see the table in the section `field`.

The same distinction between variables in the card part and the table part is made as for database fields.

A variable can be used several times in the same card pane if only one of the occurrences is open in each mode (see “Modes”).

In the table part, a column that has the heading “Job Name” that contains the variable JobText10Varcan be specified as follows:

```
var title="Job Name" data=JobText10Var> fieldtitle , variabletitle
```

and omitting the title reference works as described for `field` above. Usually, the same variable cannot be used more than once in the same table pane.

Short Form

The short form for variables in the card pane corresponds closely to the short form for database fields. The only difference is that you do not place a dot before the variable name (the dot is used by MDL to distinguish between variables and database fields). Thus,

```
<var data=DepartmentNumberVar>
```

can be written as:

```
DepartmentNumberVar
```

The short form for variables in tables resembles the form for database fields. However, no period is used here.

```
<var title="Job Name" data=JobText10Var>
```

can be written as:

```
"Job Name" JobText10Var;
```

or simply:

```
JobText10Var;
```

if you want to use the default title as heading.

As in texts, attributes are separated by colons. `justification` and `width` are nameless.

Structure

Tag	Structure	Comments
variable (card part) =	= <var *attribute*$_1$ .. *attribute*$_n$> | *ID:attribute*$_1$*: .. : attribute*$_n$	Long form. Short form.
column (table)	| <var *attribute*$_1$ .. *attribute*$_n$> | *STRING ID:attribute*$_1$*:..:attribute*$_n$*;* | *ID ID:attribute*$_1$*:..:attribute*$_n$ | *.ID ID:attribute*$_1$*:..:attribute*$_n$*;*	Variable column long form. Variable column with text heading. Variable column with heading derived from variable. Variable column with heading derived from field.

title

In Maconomy, a default title is associated with some fields and variables. You can access this default title in MDL through the `title` tag. If you want the default title for the database field `EmployeeNumber`, the following expression can be used:

```
<title fieldtitle=EmployeeNumber>
```

Similarly, you can get the default title for a variable using:

```
<title variabletitle=DepartmentNumberVar>
```

Attribute	Type	M	N	Description
title	*STRING*	(+)	(+)	This nameless attribute is the title text.
fieldtitle	*ID*	(+)	(+)	This nameless attribute gets the built-in title of a field in the card part.
variabletitle	*ID*	(+)	(+)	This nameless attribute gets the built-in title of a variable in the card part.
text formatting				A number of text formatting attributes are available: ▪ `bold` ▪ `color` ▪ `fontname` ▪ `fontsize` ▪ `italic` ▪ justificationon ▪ `rgb` ▪ showasclosedfieldon ▪ `stretch` ▪ `underline`

The use of variable and field names in title tags does not influence the number of times that they can be used in the same card pane.

Short Form

As a short form for titles in card panes, use brackets ("`[`" and "`]`") to enclose the short form of the field or variable you want the default title for.

Thus:

```
<title fieldtitle=EmployeeNumber>
```

can be written as:

```
[.EmployeeNumber]
```

Structure

Tag	Structure	Comments
title	`=` `<title` *attribute*$_1$ `..` *attribute*$_n$`>` `|` *[.ID]:attribute*$_1$`: .. :` *attribute*$_n$ `|` *[ID]:attribute*$_1$`: .. :` *attribute*$_n$	Long form. Short form for title associated with field. Short form for title associated with variable.

image

The image tag is used to display pictures in the card pane. To show a picture, the picture file must have been imported into Maconomy using a document archive. In the document archive, documents are organized in document groups, so that all documents are identified by their name and document group. A reference to an image document in MDL takes the form:

DocumentGroup\DocumentName

where *DocumentGroup* is the name of a document archive, and *DocumentName* is a member of that archive. Maconomy currently supports these image formats:

- JPEG (Joint Photographic Experts Group). The usual file extensions are `.jpg` and `.jpeg`.
- PNG (Portable Network Graphics). The usual file extension is `.png`.

If you have imported a picture file of one of these formats into a document archive, you can show it in the card pane using the following expression:

```
<image title="MyImages\MyLogo.jpg" height=100 width=80>
```

or

```
<image fieldtitle=field1 height=100 width=80>
```

Attribute	Type	M	N	Description
title	*STRING*	(+)	+	The title specifies a direct reference to the image in the document archive, in the form *DocumentGroup\Document- Name* as described above.
fieldtitle	*ID*	(+)		This attribute specifies a database field which contains a direct reference to the image in the document archive, in the form *DocumentGroup\DocumentName* as described above.
variabletitle	*ID*	(+)		This attribute specifies a variable which contains a direct reference to the image in the document archive, in the form *DocumentGroup\DocumentName* as described above.
height	*REAL*	+		This attribute specifies the height in pixels of the frame for the current image. The manner in which the picture is shown within the frame can be adjusted using the `scale` attribute.
width	*REAL*	+		This attribute specifies the width in pixels of the frame for the current image. The manner in which the picture is shown within the frame can be adjusted using the `scale` attribute.

Attribute	Type	M	N	Description
scale	*ID*			This attribute specifies how the image is to be printed within the picture frame. It has the following possible values: ▪ `natural` (default): With this setting, the picture is maximized within the frame while keeping the natural image proportions. ▪ `fillbox`: With this setting, the picture is scaled to fill out the entire specified frame, irrespective of image proportions.

Exactly one of the `title`, `fieldtitle`, or `variabletitle` attributes must be present in the `<image>` tag.

By referring directly to an image using the `title` attribute, the same picture will be shown for all records. It is also possible to show data dependent images by referring to an image title from a field or a variable.

Example

```
<image variabletitle=var2 height=100 width=80 scale=fillbox>
```

Structure

Tag	Structure	Comments
image	= <image *attribute*$_1$ .. *attribute*$_n$>	

island

An island is a frame, placed around a box. The following expression draws an island with the title “Period.”

```
<island title="Period">
   ...
<end island>
```

Attribute	Type	M	N	Description
title	*STRING*	+	+	With this nameless attribute you specify the title of the island.
titlevisible	*BOOLEAN*			If this attribute is set and `true`, the title is suppressed in the current layout. This applies to the Java™ client only.

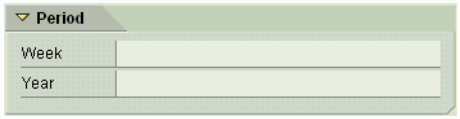
Attribute	Type	M	N	Description
text formatting				A number of text formatting attributes are available: ▪ `bold` ▪ `color` ▪ `fontname` ▪ `fontsize` ▪ `italic` ▪ `rgb` ▪ `underline`

Example

The layout for the Period island in the Time Sheets dialog can, for instance, be made by using the following expression:

```
<island "Period">
   <array>
      <row>
         <text title="Week">
         <var data=WeekVar>
      <end row>
      <row>
         <text title="Year">
         <var data=YearVar>
      <end row>
   <end array>
<end island>
```

This creates the following island:



You can find more examples of islands in “Floating.”

Structure

Tag	Structure	Comments
island	= <island *attribute*$_1$ .. *attribute*$_n$> *box* <end island>	

array

An array is used for placing elements in rows and columns. An array consists of one or more rows that are enclosed by an `array` tag and an `end array` end tag:

```
<array>
   rows
<end array>
```

Each row in an array is placed above each other and is left-aligned. Note that the boxes in the array are aligned. If a text is right-aligned in its box, the left side of the text will not be aligned.

Arrays do not take attributes.

Example

See the example in `island`.

Short Form

An array is abbreviated by placing rows in brackets. Hence,

```
<array>
   rows
<end array>
```

can be written as

```
{ rows  }
```

The content of the Period island in the Time Sheets window can be written like this (the long form of the same array was shown previously):

```
{
   "Week"   WeekVar
   "Year"   YearVar;
}
```

Structure

Tag	Structure	Comments
array	= <array> row_1 .. row_n <end array> | { row_1 .. row_n }	*array* must contain at least one row. Short form.

row

A row consists of one or more boxes (typically texts, database fields, and variables, but the boxes can also be islands or arrays).

```
<row>
   boxes
<end row>
```

Each box in a row is placed on the same line, meaning that the upper edges of the boxes are aligned.

If an array consists of more than one row, the first element of each row will be placed above each other (left-aligned), the second element of each row will be placed above each other, and so on.

Each row in an array must contain exactly the same number of elements. Arrays are also used to align boxes horizontally and vertically.

Rows do not take attributes.

Example

See the example in the section `island`.

Structure

Tag	Structure	Comments

row	`= <row>`
	`  bo2.0 .. box`$_n$
	`<end row>`
	`|  bo2.0 .. box`$_n$`;`
		row must contain at least one box.
		Short form.

Text Formatting Attributes

A group of text formatting attributes is used with a number of different tags. Instead of describing every text formatting attribute each time it can be used with the tags described previously, the descriptions are collected here, and references are made from the earlier description.

Maconomy usually chooses a reasonable text formatting (both text fields and the content of Maconomy fields), but in certain cases it can be convenient to deviate from the standard. To do this, you can choose from the following text formatting attributes. For each attribute, a list of applicable tags is given.

bold

The `bold` attribute is used for specifying that text (including the text in database fields, variables, and island titles) is to be bold. The attribute is specified as follows:

```
bold=true
bold=false
```

A `bold` attribute on a column specifies that both heading and content are to be written in **boldface**.

Examples

```
<text title="Month" bold=true>
<island title="Period" bold->
```

Applies to

```
<text>
<field>
<var>
<island>
<title>
```

color

The `color` attribute is used to specify text color (including the text in database fields, variables, and island titles). The attribute is specified as follows:

```
color=red
```

Possible values are:

```
black
yellow
magenta
red
cyan
green
blue
white
```

When a color for a column in a table pane is specified, both the data and the heading in the column are given the color in question. Note that colors do not apply in the Maconomy client for the Java™ platform.

See also the description of the `rgb` attribute.

Example

```
<island title="Employee" color=blue>
```

Applies to

`<text>`
`<field>`
`<var>`
`<island>`
`<title>`

fontsize

The `fontsize` attribute is used to specify the font size (including the size of text in database fields, variables, and island titles). The attribute is specified as follows:

```
fontsize=9
```

The font size is specified as an *INTEGER*. The default size is 9 (if nothing else is specified). The theoretical range is 0-32767. The specified font size influences both heading and column data.

Example

```
<field data=EmployeeNumber fontsize=12>
```

Applies to

`<text>`
`<field>`
`<var>`
`<island>`
`<title>`

italic

The `italic` attribute is used to specify that text (including the text in database fields, variables, and island titles) is to be in *italics*. The attribute is specified as follows:

```
italic=true
italic=false
```

The `italic` attribute on columns has an effect in both heading and content.

Example

```
<var data=MonthVar italic=true>
```

Applies to

`<text>`
`<field>`
`<var>`
`<island>`
`<title>`

justification

The `justification` attribute is nameless and specifies whether a text should be aligned left, right, or centered. Note that if a justification is not specified, the justification depends on the type of the field/variable (for instance, amounts are right- aligned), so in practice, it is seldom necessary to use this attribute.

You can select from three justification types:

```
justification=left
justification=right
justification=center
```

The `justification` attribute can be used on columns; if you do this, both the heading and the content are justified in the specified manner. In search windows, the field in the search line is also justified.

Example

```
<var data=MonthVar justification=right>
```

This right-aligns the content of the `MonthVar` field.

Applies to

`<text>`
`<field>`
`<var>`
`<island>`
`<title>`

rgb

The `rgb` attribute is nameless and makes it possible to use more color hues than when using the `color` attribute. An RGB color is specified as a triple (*R*,*G*,*B*), where *R*, *G,* and *B* are percentages between 0 and 100, specifying the intensity of red, green, and blue, respectively. The values can be specified with or without decimals (that is as *INTEGER* or *REAL*). Consider the following example that will result in a burgundy color:

```
rgb=(50,0,25)
```

Like `color`, `rgb` can be used on texts, database fields, variables, and islands (in islands the attribute only applies to the heading of the island). Note that colors do not apply in the Maconomy client for the Java™ platform.

Example

```
<island title="Employee" rgb=(30,100,50)>
```

Applies to

`<text>`
`<field>`
`<var>`
`<island>`
`<title>`

showasclosedfield

The design of the card part of some Maconomy dialog windows relies on the identical appearance of labels and closed fields. In the user interface of the Java client, labels and closed fields no longer have the same appearance. However, with this attribute and the `showaslabelifclosed` attribute you can explicitly change the appearance: labels may be changed to appear as closed fields, and closed fields may be specified to appear as labels.

The `showasclosedfield` attribute for the label tags text and title makes the card label appear as if it were a closed field.

This attribute only has effect in the Maconomy client for the Java platform. The Maconomy client for Windows ignores the attribute.

Example

```
<text "Employee No." showasclosedfield+>
```

Applies to

`<text>`
`<title>`

stretch

The `stretch` (extensibility) attribute is used to specify how extra space is to be divided between box elements that are placed next to each other. If a row of boxes (database fields, variables, or texts) has equal extensibility, the extra space is equally divided between the boxes; otherwise, the extra space is divided between the boxes with the highest extensibility.

Extensibility is specified as follows:

`stretch=low`
`stretch=medium`
`stretch=high`

If you do not use the `stretch` attribute, the text extensibility is `medium`, whereas the extensibility of variables and database fields is `high`. This means that in the typical scenario where database fields or variables follow fixed texts, the text gets its own size, and the field gets all of the extra space.

By specifying a row (fixed text followed by a variable), as in the following example, the variable field will have a width of 2 em, and any extra space will be used by the text (because it has a higher extensibility).

```
<row>
   <text title="Month">
   <var data=MonthVar stretch=low width=2.0>
<end row>
```

Applies to

`<text>`
`<field>`
`<var>`
`<title>`

underline

The `underline` attribute is used to specify that text (including the text in database fields, variables, and island titles) is to be underlined. The attribute is specified as follows:

```
underline=true
underline=false
```

The underline attribute on columns has effect in both heading and content.

Example

```
<field data=EmployeeNumber underline=true>
```

Applies to

```
<text>
<field>
<var>
<island>
<title>
```

width

The `width` attribute is used to specify the width of a text element or an image. The width is measured in *em*, except for images, which are specified as pixels.

```
width=10.0
```

The width of a box will never be smaller than its content. Hence, `"abc":width=0` will be at least as wide as `"abc."` See also “Very Short Texts.”

Example

```
<field title="Job No." data=JobNumber width=10>
```

Applies to

```
<text>
<field>
<var>
<image>
<island>
<title>
```

Working with Layouts

Floating

Groups

A group does not correspond to a window element, but is only used to control the formatting of panes with two columns (see the previous section concerning double-column panes).

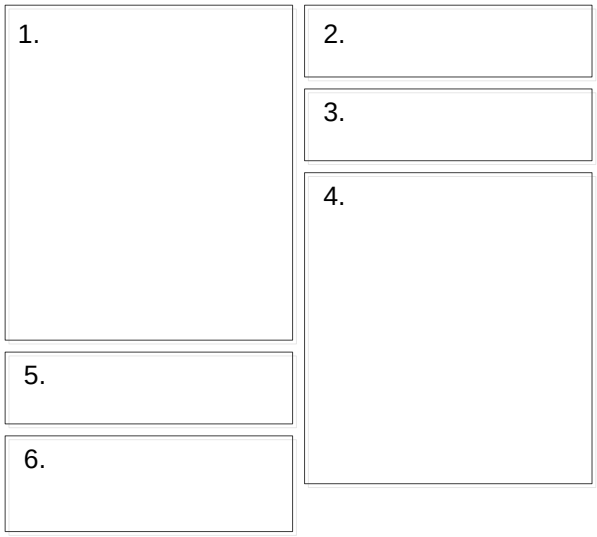
If a pane with two columns is read from left to right, the pane elements (typically islands) appear exactly in the order in which they appear in the MDL layout. More explicitly:

1. The first element is placed to the left.
2. Subsequent elements are placed below the lowest column if the width is large enough; otherwise they are placed to the left under both columns.

This process is called *floating*. Consider the following example:

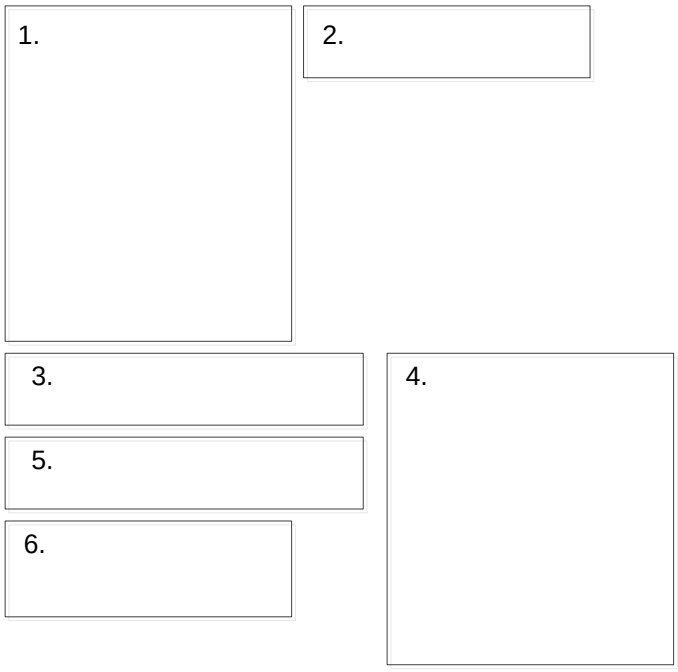
```
<dualpane>
   <group>
      <island title="1.">
         ...
      <end island>
      <island title="2.">
         ...
      <end island>
      <island title="3.">
         ...
      <end island>
      <island title="4.">
         ...
      <end island>
      <island title="5.">
         ...
      <end island>
      <island title="6.">
         ...
      <end island>
   <end group>
<end dualpane>
```

The result of the preceding example is the following layout (provided that all islands are narrower than half of the window):



This is the result of the preceding code example. All islands fit in two columns.

If, instead, it is assumed that the first island is exactly half the size of the window width, but that the third island is larger and the fourth smaller, you would get the following layout:



This is the result of the preceding code example. Some islands are wider than half of the window.

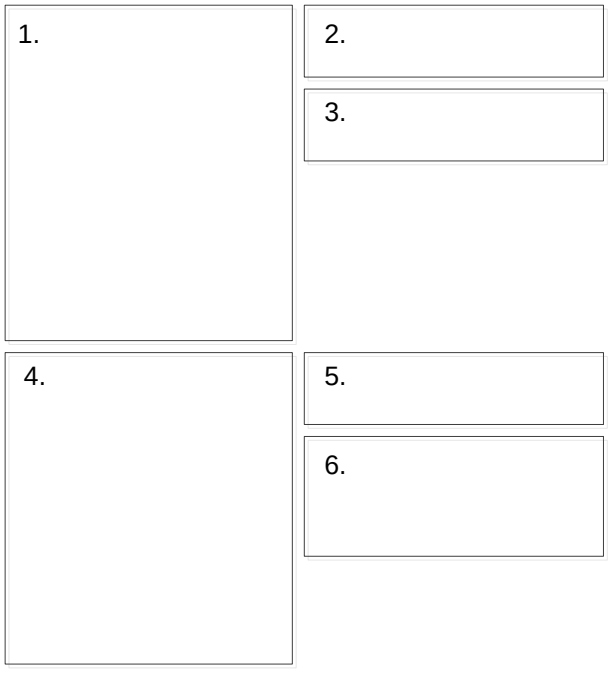
If you want to control the exact layout yourself, you can use arrays. However, you might often let Maconomy handle the precise formatting, and only indicate that certain elements are to be grouped.

If, for instance, islands 1, 2, and 3 are logically connected, and 4, 5, and 6 are connected, you could write:

```
<dualpane>
   <group>
```

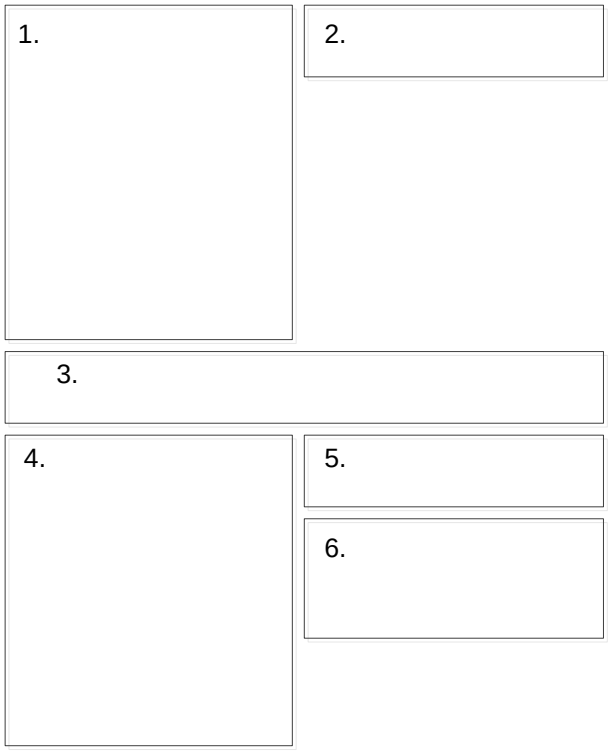
```
      <island title="1.">
         ...
      <end island>
      <island title="2.">
         ...
      <end island>
      <island title="3.">
         ...
      <end island>
      <island title="4.">
         ...
      <end island>
      <island title="5.">
         ...
      <end island>
      <island title="6.">
         ...
      <end island>
   <end group>
<end dualpane>
```

This yields (with the original assumption that all islands are narrower than half of the window) the following layout:



This is the result of the preceding code example. Islands are split into two groups.

If the third island is placed in a group of its own, the result is as follows:



This is the result of the preceding code example. Islands are split into three groups.

The preceding result is due to the fact that the width of an island that is placed in a group of its own is set to the total width of the window.

Modes

Using the `mandatory`, `ronew`, `roupdate` and `readonly` attributes on the `field` tag, you can specify whether it should be possible to edit fields or not. You can deny access to open fields (but not grant access to closed fields), and you can require that information is entered in certain fields.

Each pane in a dialog window is always in one of the following two modes: *New* mode or *Updating* mode.

Example: If you have opened the Exchange Rate Table window and you select “New Exchange Rate Table” in the Index or File menu, the card pane will be ready for entry of a new exchange rate table. When you subsequently enter the name of an exchange rate table and press Enter, the card pane will change to *Updating* mode, where it is possible to update (change) fields.

In the same way, a row in the table part is said to be in the *New* mode when it is being created (for instance by clicking on a new row). Once you have entered data and pressed Enter, the table pane is also put into *Updating* mode.

If the user cannot enter data into a field, the field is *closed*. A field can be open in *New* mode and closed in *Updating* mode (or another combination of open and closed). Fields in the search window cannot be closed. When a field is *mandatory* (required), you cannot leave the window before the field has been completed. Therefore, a field cannot be mandatory if it is closed in both New and Updating mode.

In the description of the `field` tag, it was noted that a variable or a database field can only be used several times in the same card pane if at most one of the instances of the field or variable is open in each of the two states. This means that if you want a field to be included several times in a card pane, you must specify for all of the instances of the field (or all of them except one) that it is

closed in *New* mode. Similarly, you must specify that all instances of the field (or all of them except one) are closed in *Updating* mode. Please note, however, that it is possible to have one instance of a field open in *New* mode and another instance open in *Updating* mode.

Layout Examples

This section contains a number of MDL layout examples. In the first part, it is illustrated that most layouts are quite simple to make. The second part explains and illustrates how some special layout types are handled.

When constructing layouts, it is convenient to use the standard layout as the basis for the dialog.

In the examples, the short form is generally used.

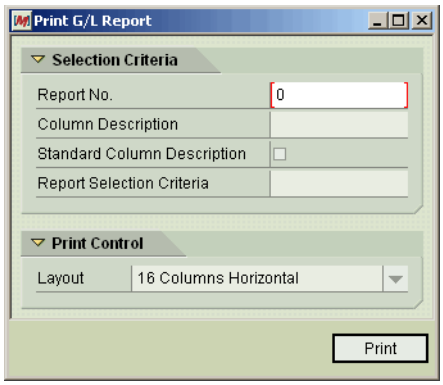
One Card Pane

In this section, we construct a simple layout for the Print G/L Report parameter dialog. The dialog consists of one card pane. The dialog contains five variables: `ReportNumberVar`, `ColumnDescriptionNameVar`, `StandardColumnDescriptionVar`, `TargetGroupNameVar` and `Layoutname`. In `ReportNumberVar`, `ColumnDescriptionNameVar`, `StandardColumnDescriptionVar`, and `TargetGroupNameVar` you enter information about the content of the G/L report. It is therefore natural to place these four variables in a separate island called Selection Criteria. To the left of each variable, you then place a fixed text that describes the variable.

Each island contains an array, in which each row consists of a `text` tag (the fixed text), and a `var` tag. These considerations result in the following layout:

```
<mdl 4>
<layout Example "Example" windowtype=DialogWindow windowname=ExchangeRateTables>
   <singlepane>
      <island "Selection Criteria">
         <array>
            <row>
               "Report No."     ReportNumberVar
            <end row>
            <row>
               "Standard Column Description" StandardColumnDescriptionVar
            <end row>
            <row>
               "Report Selection Criteria" TargetGroupNameVar
            <end row>
         <end array>
      <end island>
      <island "Print Control">
         <array>
            <row>
               "Layout"  LayoutName
            <end row>
         <end array>
      <end island>
   <end singlepane>
<end layout>
```

This layout results in the following window:

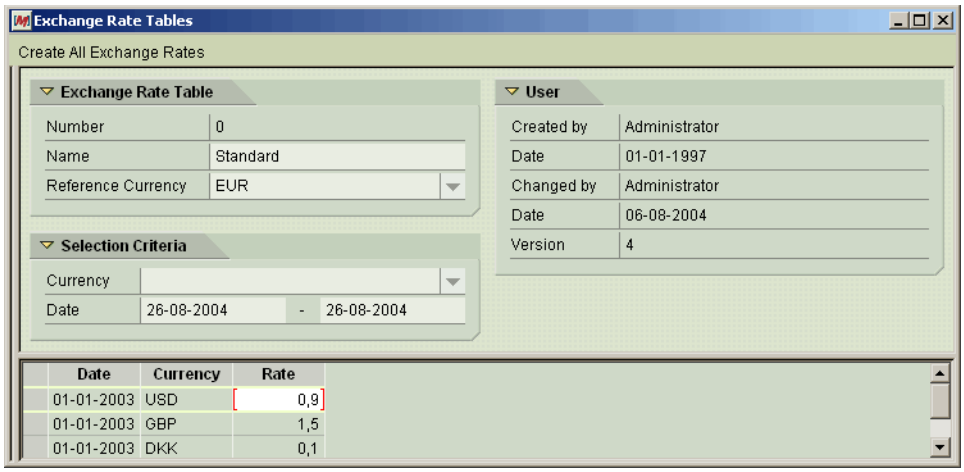


This is the result of the preceding code example: Print G/L Report.

Making Layout Changes

Suppose that you want to change the Exchange Rate Tables window. The original layout of the window looks as follows:

However, you want to change the layout to have the window look as follows:



This section gives a step-by-step description of how this layout is constructed. The layout contains two panes: A card pane and a table pane. If the layout is called *Example*, the `layout` tag is specified as follows:

```
<mdl 4>
<layout Example "Example" windowtype=DialogWindow windowname=ExchangeRateTables>
    ...
<end layout>
```

The Card Part

Because two islands are placed next to each other, the card part is a dual pane—meaning that the expression looks as follows:

```
<dualpane>
    ...
<end dualpane>
```

Islands and Floating

The islands are placed in such a way that the order corresponds to the order of the layout when the islands are read in the usual reading order (left to right, top to bottom). This means that the islands are to be written in the following order: Exchange Rate Table, User, Selection Criteria. Maconomy automatically places the islands in such a way that only one group is needed.

```
<group>
   <island "Exchange Rate Table">
      ...
   <end island>
   <island "User">
      ...
   <end island>
   <island "Selection Criteria">
      ...
   <end island>
<end group>
```

Elements

The Exchange Rate Table island contains six elements (three texts and three fields) that must be placed next to each other. The elements are placed as an array containing three rows:

```
{
   "Number" .ExchangeRateTableNumber;
   "Name"   .Name;
   "Reference Currency" .ReferenceCurrency;
}
```

The User island is also placed in an array, this time containing five rows, all having two elements:

```
{
   "Created by"   .CreatedBy;
   "Date"   .CreatedDate;
   "Changed by"   .ChangedBy;
   "Date"   .ChangedDate;
   "Version".VersionNumber;
}
```

This type of island is very typical: Many islands can be described by means of a two-column array: fixed texts followed by database fields or variables.

The last island, Selection Criteria, is different, because the second row contains four elements: A fixed text, a database field, a hyphen, and a second database field. This can be done by letting the island contain two arrays: The first with two elements in one row, and the second with four elements in one row. However, this would not be a good idea because of the following reasons:

1. You could not be certain that the left side of the `CurrencyVar` pop-up field would align perfectly with the left side of the `FromDateVar` field.
2. The Selection Criteria island would be wider than the User island, because the Selection Criteria island would be perceived as containing several fields.

The following solution is more usable: The Selection Criteria island contains one array that has two columns and two rows; the second element in the second row is an array itself, containing three elements in one row. The result is as follows:

```
{
   "Currency"     CurrencyVar;
   "Date"   {FromDateVar '-' ToDateVar;};
}
```

Such arrays-in-arrays are called *embedded* arrays. They are frequently used for advanced element formatting (more examples of embedded arrays later in this section).

This island also illustrates the use of short texts: The use of the apostrophes in `'-'` ensures that no space is placed around the hyphen.

The Table Part

The table part is constructed in a very simple way:

```
<tablepane>
   "Date"   .StartingDate;
   "Currency"     .Currency;
   "Rate"   .ExchangeRate;
```

```
<end tablepane>
```

Each line is a heading that is followed by a database field name.

The Entire Layout

The entire layout is now as follows:

```
<mdl 4>
<layout Example "Example" windowtype=DialogWindow windowname=ExchangeRateTables>
   <dualpane>
      <group>
         <island "Exchange Rate Table">
            {
               "Number"  .ExchangeRateTableNumber;
               "Name"    .Name;
               "Reference Currency"    .ReferenceCurrency;
            }
         <end island>
         <island "User">
            {
               "Created by"     .CreatedBy;
               "Date"    .CreatedDate;
               "Changed by"     .ChangedBy;
               "Date"    .ChangedDate;
               "Version" .VersionNumber;
            }
         <end island>
         <island "Selection Criteria">
            {
               "Currency"CurrencyVar;
               "Date"    {FromDateVar '-' ToDateVar;};
            }
         <end island>
      <end group>
   <end dualpane>
   <tablepane>
      "Date" .StartingDate;
      "Currency"    .Currency;
      "Rate" .ExchangeRate;
   <end tablepane>
<end layout>
```

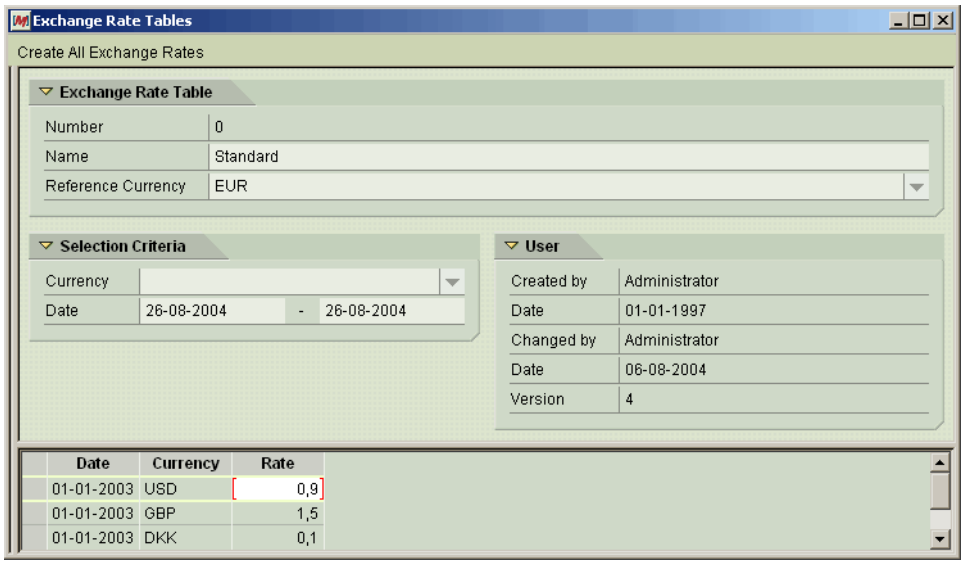
Use of Groups

If you place all islands in a double-column pane in one group, they will all be placed in two columns. You might not always want this, for example, if you want more space in one island than in others. Assume that you want to make more space for the exchange rate table name in the layout from the previous section. In this case, you can use the following expression:

```
<mdl 4>
<layout Example "Example" windowtype=DialogWindow
```

```
windowname=ExchangeRateTables>
   <dualpane>
      <group>
         <island "Exchange Rate Table">
            {
               "Number"  .ExchangeRateTableNumber;
               "Name"    .Name;
               "Reference Currency"    .ReferenceCurrency;
            }
         <end island>
      <end group>
      <group>
         <island "Selection Criteria">
            {
               "Currency"CurrencyVar;
               "Date"    {FromDateVar '-' ToDateVar;};
            }
         <end island>
         <island "User">
            {
               "Created by"     .CreatedBy;
               "Date"    .CreatedDate;
               "Changed by"     .ChangedBy;
               "Date"    .ChangedDate;
               "Version" .VersionNumber;
            }
         <end island>
      <end group>
   <end dualpane>
   <tablepane>
      "Date" .StartingDate;
      "Currency"    .Currency;
      "Rate" .ExchangeRate;
   <end tablepane>
<end layout>
```

This results in the following window layout:



If you place the Exchange Rate Table island in a group of its own, the layout allows the island to stretch across the entire window width. In addition, the Selection Criteria island is written before the User island to place the islands in the right order.

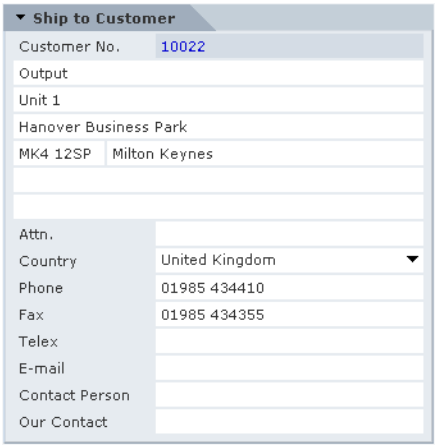
Using Absolute Lengths

The template for layout design that was described in the two previous sections is usually sufficient for you to express the window designs that you want. Groups are used for island placing, whereas arrays (occasionally embedded) are used for specific element placing.

It is usually sufficient to use these simple elements when constructing a layout, because it ensures homogeneous and easily read windows. In certain situations, however, you may not be able to do a satisfactory placing by means of these elements—and it might be necessary for you to make use of the `width` attributes.

Zip Code

Addresses are frequently used in Maconomy, usually with the following appearance:



What is unusual about the preceding placing is the fact that the fifth line contains two fields, and that these two fields are not of equal length. This can be done in MDL in the following way:

```
<island "Ship to Customer">
```

```
   {
      {
         "Customer No."   .CustomerNumber;
      };
      {
         .Name1;
         .Name2;
         .Name3;
      { .ZipCode:7.0:stretch=medium .PostalDistrict; };
         .Name4;
         .Name5;
      };
      {
         "Attn."   .Attention;
         "Country" .Country;
         "Phone"   .Telephone;
         "Fax"     .Telefax;
         "Telex"   .Telex;
         "E-mail"  .ElectronicMailAddress;
         "Contact Person" .ContactPerson;
         "Our Contact"    .OurContact;
      };
   }
<end island>
```

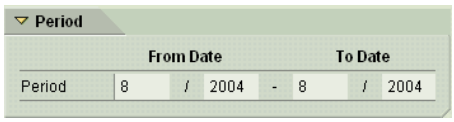
To ensure that the field is not stretched more than the stated width of seven characters (remember that fields usually have high extensibility), set `stretch` to medium on the `ZipCode` database field.

Note that in MDL it is not possible to put the `CustomerNumber` and `Attention` fields in the same column, but the MDL minimum width for texts ensures that they are still placed under each other. If the `"Our Contact"` text had been much longer, this could only have been done by defining the widths of the text. However, you will often get a nicer layout if you do not.

One Heading above Several Array Columns

Many parameter windows have period specifications that correspond to the following example:

In this example, you might want the island to show headings above the dates, as follows:



This can be done by writing the following island specification:

```
<island "Period">
   {
      ''     "From Date":bold+:center   '' "To Date":bold+:center;
      "Period"     {MonthFrom:3.0      '/'    YearFrom:3.0;}      '-'
      {MonthTo:3.0 '/'    YearTo:3.0;};
```

```
   }
<end island>
```

As in the case of the zip code, you here let `MonthFrom`, `'/'`, and `YearFrom` constitute one array, which is embedded in another array (similar for `MonthTo`, `'/'`, and `YearTo`). Note how the first row uses empty strings in the first and the third columns to make sure that the correct number of elements is specified. The apostrophes are specified to ensure that the empty texts do not have an influence on the column widths.

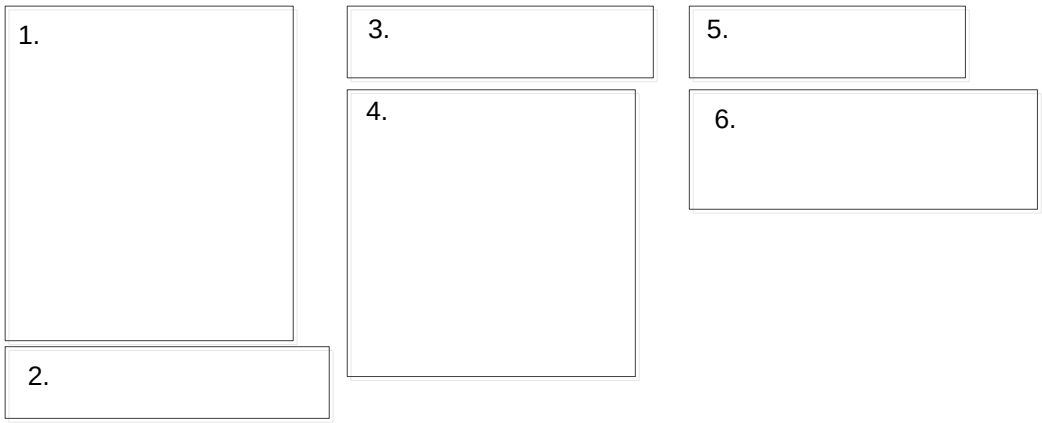
In this example, using “” would result in the same layout.

Islands in More than Two Columns

If you want to create a layout that has islands in more than two columns, use arrays. For example, perhaps you want three columns that have two islands in each column. To accomplish this, create a layout like this:

```
<singlepane>
   { --Array containing a row of 3 columns
      { --First column
         <island title="1.">
            ...
         <end island>;
         <island title="2.">
            ...
         <end island>;
      }
      { --Second column
         <island title="3.">
            ...
         <end island>;
         <island title="4.">
            ...
         <end island>;
      }
      { --Third column
         <island title="5.">
            ...
         <end island>;
         <island title="6.">
            ...
         <end island>;
      }; --This semicolon ends the row of three columns
   }
<end singlepane>
```

The result of this MDL code would be something like the following:



This is the result of the preceding code example. Islands are arranged in one row and three columns, with two islands each.

MDL Syntax

This section describes the formal MDL syntax. A formal syntax gives a brief and precise description that can be used as a definition of MDL and as a reference when you already know the language.

The syntax is described in a variant of the Backus-Naur Form. This notation is a precise method for describing valid language constructs.

Terminal symbols (that is, symbols that should be written in MDL as they appear in the BNF table) are written in `monospace font`, non-terminal symbols (symbols that are defined by the grammar) as *italic lower case*, and grammar primitives (similar to non-terminal symbols, but typically more fundamental and defined less formally) in *ITALIC CAPITALS*. For a description of attribute types, see “Attribute Types.” For a description of comments and whitespace, see “Comments and Whitespace.”

Alternatives are specified by writing | between the options. If symbols can appear more than once, this is specified by means of . . . If a tag can have several attributes, it is written like this: $attribute_1$ . . $attribute_n$.

Syntax

The MDL syntax can be described by means of the following grammar stated. The start symbol of the grammar is *“start.”*

Symbol	Grammar	Comment
start	= <mdl *INTEGER*> *layout*	Version no. 1-4.
layout	= <layout *attribute*+> *availableactions?* *availableshortcuts?* *pane*+ <end layout>	*availableactions* and *availableshortcuts* can be omitted.
	| <layout *attribute*+> *availableactions?* *availableshortcuts?* $pane_1$ $pane_2$ <end layout>	$pane_1$ must be *singlepane* or *dualpane*; $pane_2$ must be *tablepane*.
availableactions	= <availableactions?> *action** <end availableactions>	*availableactions* need not contain any actions.
action	= <action *attribute*+>	
availableshortcuts	= <availableshortcuts> *shortcut** <end availableshortcuts>	*availableshortcuts* need not contain any shortcuts.
shortcut	= <shortcut *attribute*+>	

Symbol	Grammar	Comment
pane	= *singlepane* | *dualpane* | *tablepane*	
singlepane	= <singlepane> *buttons?* *dropdownbutton?* *box** <end singlepane>	*singlepane* need not contain any boxes , buttons, or dropdownbutton.
dualpane	= <dualpane attribute?> *buttons?* *dropdownbutton?* *group** <end dualpane>	*dualpane* need not contain any groups, buttons, or dropdownbutton.
tablepane	= <tablepane attribute> *buttons?* *dropdownbutton?* *column+* <end tablepane>	*tablepane* must contain at least one column. Buttons and dropdownbutton can be omitted.
buttons	= <buttons> *action** <end buttons>	*buttons* need not contain any actions.
dropdownbutton	= <dropdownbutton> *action** <end dropdownbutton>	*dropdownbutton* need not contain any actions.
group	= <group> *box+* <end group>	*group* must contain at least one box.
box	= *text* | *field* | *variable* | *title* | *image* | *island* | *array*	
text	= <text attribute+> | *STRING:attribute** | *FSTRING:attribute**	Double-quote text. Single-quote text.
field	= <field *attribute+*> | *.ID:attribute**	Short form.

Symbol	Grammar	Comment
variable	= <var *attribute+*> | *ID:attribute**	Short form.
title	= <title *attribute+*> | [*.ID*]:*attribute** | [*ID*]:*attribute**	Short form for title associated with field. Short form for title associated with variable.
image	= <image *attribute+*>	
island	= <island *attribute+*> *box* <end island>	
array	= <array> *row+* <end array> | { *row+* }	*array* must contain at least one row.
row	= <row> *box+* <end row> | *box+*;	*row* must contain at least one box.
column	= <field *attribute+*>	Field column, long form.
	| *STRING .ID:attribute**;	Field column with text heading.
	| *ID .ID:attribute**;	Field column with heading derived from variable.
	| *.ID .ID:attribute**;	Field column with heading derived from field.
	| <var *attribute+*>	Variable column long form.
	| *STRING ID:attribute**;	Variable column with text heading.
	| *ID ID:attribute**;	Variable column with heading derived from variable.
attribute	= *ID = attributevalue*	
	| *ID+*	Short form for Booleans. Note: This is a literal +.
	| *ID-*	Short form for Booleans.

Symbol	Grammar	Comment
	| *attributevalue*	Nameless.
attributevalue	= *BOOLEAN* | *INTEGER* | *REAL* | *STRING* | *ID* | (*INTEGER*,*INTEGER*) | (*REAL*,*REAL*,*REAL*)	

Error Messages

This section contains a list of all possible error messages from the MDL compiler. Error messages are saved to the "`LAYERR.TXT`" file in the Maconomy folder. If you are using the Maconomy client for the Java™ platform, the errors are displayed in a new browser window. If no errors occur when importing a layout, no file is generated. An explanation of the individual error messages is given below each message, including any problem-solving suggestions. Note that many error messages are prefixed by "Error in line *nn*:," but this prefix is left out in the following.

Syntax Errors

If an MDL layout does not follow the syntax that is described in the previous section, the following error message is given by Maconomy:

- Error in line *nn*: Syntax error.

  where *nn* specifies the line number in which the error occurs.

  You will often see the error by looking at the line identified by the interpreter, but sometimes the selected line will be correct. The most frequent cause of such an error[2] is that parenthetical tags have not been *balanced*, meaning that a tag has not been closed (or has been closed by another end tag) or that an opening tag is missing in an end tag. Such errors can be difficult to locate, but the process is made less difficult if the content of the tag/end tag is indented as in the examples shown in this manual. Consistent indentation makes the structure much easier to read.

- The layout title cannot be empty.

  This applies to MDL versions below 4. It is not allowed to specify an empty layout title. The layout contains: "`<layout "">,`' which should be changed to '`<layout "layout-title">.`'

- The layout name cannot be empty.

  This applies to MDL version 4 and above. It is not allowed to specify an empty layout name. The layout contains: '`<layout "">,`' which should be changed to '`<layout "layout-name">.`'

- The layout title cannot begin or end with blanks.

  It is not allowed to specify blank characters at the beginning and at the end of the layout title in the `layout` tag.

- The layout name cannot begin or end with blanks.

  It is not allowed to specify blank characters at the beginning and at the end of the layout name in the `layout` tag.

- The layout title cannot contain '_' characters.

  It is not allowed to specify the underscore character in the layout title in the `layout`tag.

- The layout name cannot contain '_' characters.

  This applies to MDL version 4 and above. It is not allowed to specify the underscore character in the layout name in the `layout` tag.

[2] Particularly if the MDL interpreter reports the error on the first or the last layout line.

- Name is too long. Shortened to *nn*characters.

  MDL does not allow names with more than 100 characters. A name can be a database name, a variable name, an attribute name or an attribute value of the *ID* type.

- Text is too long. Shortened to *nn*characters.

  MDL does not allow texts—meaning strings that are enclosed by single quotes (`'...'`) or double quotes (`"..."`)—to consist of more than 100 characters. This will, however, seldom be a problem, because texts cannot contain line feed characters.

- Incomplete text.

  A string is not ended, meaning that a string begins with `"` but is not ended by `"`. Maconomy will try to read on, but is of course not able to find out where the string should have been ended. This can cause some rather odd errors in the later layout. You should just enter the end quote and try to import the layout again.

- Error in integer '*ss*'.

  ss is an integer, which is too large. Integers cannot be larger than 21474836467.

- A real number must have between 1 and 3 digits after the decimal point.

  A decimal figure that has more than three digits has been specified. Note that leaving out the digit after the decimal point will result in a syntax error (and it will be reported as such), if no digits are specified (as in `'32.'`).

- Error in real '*ss*'.

  ss is a floating point number that is much too large (positive or negative). The range of permitted numbers is from 1.7E308 to -1.7E308.

Semantic Errors

Maconomy gives detailed error messages on semantic errors (errors that are related to matters of meaning). The following sections list all of the semantic error messages and their meanings.

Mandatory Fields

If a database field or a variable is defined by Maconomy as being mandatory in a dialog, it must be included in the layout.

In the following, only database field errors are mentioned, but the same errors can occur for variables.

- Error: The mandatory database field '*ss*' must be used in the upper pane.

  A mandatory database field with the name *ss* is omitted from the upper pane.

- Error: The mandatory database field '*ss*' must be used in the lower pane

  A mandatory database field with the name *ss* has been omitted from the lower pane.

- The database field '*ss*' is a check box and cannot be made mandatory.

  Checkboxes cannot be made mandatory.

- The database field '*ss*' is a popup and cannot be made mandatory.

  Pop-up fields cannot be made mandatory.

- The database field '*ss*' cannot be mandatory as well as closed.

A database field cannot be totally closed as well as mandatory, as it will prevent the user from entering data.

- The closed database field '*ss*' cannot be made mandatory.

  A mandatory database field cannot be totally closed, as it will prevent the user from entering data. The database field was closed by Maconomy.

- The database field '*ss*' was already closed in one state. Hence, closing the field in the other state and making it mandatory is illegal.

  The attribute `mandatory` is used with either `ronew` or `roupdate`. As the database field was already closed in the other state by Maconomy, the field is totally closed and mandatory, which is an error.

- Error: The database field '*ss*' in upper pane was open in the new state and mandatory. One instance of the database field must be open in the new state.

  The database field '*ss*' is mandatory and open in *new* mode by system default. There must therefore be one occurrence of '*ss*' in the upper pane which is open in this mode.

- Error: The database field '*ss*' in lower pane was open in the new state and mandatory. One instance of the database field must be open in the new state.

  The database field '*ss*' is mandatory and open in *new* mode by system default. There must therefore be one occurrence of '*ss*' in the upper pane which is open in this mode.

Parameter Windows and Search Windows

In certain areas, parameter windows and search windows differ from the other dialogs.

- Fields or variables cannot be used as column headers in search windows.

  The attribute `fieldtitle` or `variabletitle` was used in a layout description that pertains to a search window. These attributes are only allowed in layout descriptions for card/table windows.

- Only variable fields are allowed in parameter dialogs.

  A database field exists in a parameter dialog. Parameter windows can only contain variables.

- Only database fields are allowed in search windows.

  A variable exists in a search window. Search windows can only contain database fields.

Panes

Using other pane types than those defined by Maconomy will result in an error.

- A card dialog must contain one singlepane or one dualpane.

  Only card panes (not table panes) can be specified in a card dialog.

- There can be only one pane in a card dialog.

  A card dialog must contain exactly one pane.

- A table dialog must contain one tablepane.

  A table dialog can only contain one table pane (single panes and dual panes are not allowed).

- There must be exactly one pane in the layout.

  The current layout can only contain one pane.

- There can be only one pane in a table dialog.

  A table dialog can only contain one pane.

- Unexpected pane in card/card dialog.

  Both the upper and the lower pane in a card/card dialog must be card panes (meaning single panes or dual panes). From the line number, it is possible to determine which pane is actually causing the error.

- There are only two panes in a card/card dialog.

  The layout contains more than two panes.

- Unexpected pane in card/table dialog.

  The upper pane is not a card pane (meaning a single pane or a dual pane) or the lower pane is not a table pane. The line number indicates which pane is actually causing the error.

- There are only two panes in a card/table dialog.

  The layout contains more than two panes.

- A parameter dialog must contain one singlepane or one dualpane.

  A layout for a parameter dialog must contain a pane, which must be a single pane or a dual pane.

- There is only one pane in a parameter dialog.

  A layout for a parameter dialog cannot contain more than one pane.

- A search window must contain one tablepane.

  A layout for a search window has to contain exactly one table pane.

- There is only one pane in a search window.

  A layout for a search window contains more than one pane.

- There must be exactly two panes in the layout.

  Only one pane is specified for a dialog that must contain two panes.

- Only parameter and card dialogs may contain empty singlepanes.

  A single pane can be empty if it is used as (the only) pane in a parameter dialog or a card dialog; otherwise, empty single panes are not allowed.

Rows

- The row has *xx* elements. *yy* was expected.

  The row consists of a wrong number of elements. The previous rows (or the previous row) all contained *yy* elements, but the current row contains *xx* (which is either larger or smaller than *yy*).

Database Fields and Variables

You can only specify database fields and variables that are connected to the pane in which they are specified. In addition, database fields and variables can only be specified more than once if they are always closed.

In the following, only database field errors are mentioned, but the same errors can occur for variables (with a few exceptions).

- The database field ’*ss*’ does not exist.

  This pane contains no database field with the name “*ss*.”
- The database field ‘*ss*’ is open more than once in the new or update state.

  Only one instance of a database field can be open in the new or update state.
- The database field ’*ss*’ is a *tt* and cannot use zero suppression.

  The `zerosuppression` attribute does not apply to database fields of a type different from *INTEGER*, *REAL*, or *AMOUNT*.
- The database field ’*ss*’ is mandatory and cannot use zero suppression.

  The attribute `zerosuppression` attribute does not apply to mandatory database fields.
- The database field ’*ss*’ is a *tt*and cannot be visualized as time.

  The `visualizeastime` attribute does not apply to database fields of a type different from *REAL*.
- The database field ’*ss*’ is mandatory and cannot be summed.

  The `summed` attribute does not apply to database fields of a type different from *INTEGER, REAL,* or *AMOUNT*.
- Repeated use of the database field ’*ss*’ is not allowed.

  Database fields only. The database field ’*ss*’ is used several times. This error message is only given for search windows, where the same database field cannot occur more than once.
- There is no relation in this pane, hence database field ’*ss*’ is not allowed.

  Database fields only. Certain panes have no related relations. In these cases, it is not possible to refer to database fields.

Attributes

When error messages that are related to attributes occur, it is usually convenient to consult the reference section for the current tag for legal attribute specifications.

- Only versions 1 to 4 of MDL are supported.

  The permitted values for the `version` attribute in the `mdl` tag is `1` through `4`.
- Only MDL version 2 and later supports fields or variables as column titles.

  The `fieldtitle` or `variabletitle` attribute was used on line *nn*, but MDL version 1 has been specified at the beginning of the layout description.
- Invalid attribute ‘*ss*’.

  The specified attribute is invalid for a tag in the layout.
- The attribute ‘*ss*’ is invalid for the tag ‘*ss*’.

  The specified attribute is invalid for the tag mentioned in the message.
- Exactly one of attributes title, fieldtitle or variabletitle must be given.

You have specified either none or several of the `title`, `fieldtitle` or `variabletitle` attributes in a tag of the type "`image`." You must specify one of these attributes.

- At most one of attributes title, fieldtitle or variabletitle can be given.

  You have specified more than one of the `title`, `fieldtitle` or `variabletitle` attributes in the tag. You must specify exactly one of these attributes.

- Exactly one of attributes fieldtitle or variabletitle must be given.

  You have specified either none or several of the `fieldtitle` or `variabletitle` attributes in a tag of the type "`title`." You must specify exactly one of these attributes.

- Some necessary attributes are missing for the tag '*ss*'.

  A necessary attribute was omitted in the "*ss*" tag. See the reference section for the current tag for legal and mandatory attributes. Note that the tag name '*ss*' does not necessarily occur in the layout. If short forms are used, the tag is omitted and derived from Maconomy. If the short form is specified in a wrong way, Maconomy might derive a wrong tag name, which might then be the cause of the error.

- The value '*ss*' cannot be used with the attribute '*tt*'.

  An attribute has been specified as `tt=ss`. *ss* has got the correct type, but it has a value that *tt* cannot accept, for example:

- stretch=unknownStretch

  where `unknownStretch` has the type *ID* (which is the correct type for values connected to the `stretch` attribute), but which is not a known value.

- Illegal window type '*ss*'.

  The window type specified in the `layout` tag is invalid. See the reference section for the "layout" tag for valid values.

- Invalid WindowName '*ss*', it must be '*ss*'.

  When importing a layout, Maconomy checks that the window name in the layout matches that of the window into which the layout is imported.

- Invalid WindowType '*ss*', it must be '*ss*'.

  When importing a layout, Maconomy checks that the window type in the layout matches that of the window into which the layout is imported. See the reference section for the "layout" tag for valid values.

- The attribute '*ss*' is not defined for the tag '*tt*'.

  An attempt has been made to use the "*ss*" attribute with the "*tt*" tag. However, such a match is not possible. See the reference section for the current tag for legal attributes. Note that "*tt*" does not necessarily appear in the layout if short forms are used, and that a wrong short form can be the cause of the error.

- The attribute '*ss*' is already set for the present tag.

  The "*ss*" attribute has been used several times. In case of this error message, the name of the attribute will be stated explicitly or you will get the error message described below.

- The attribute '*ss*' of type '*ss*' is already set for the present tag.

  The "*ss*" attribute has been used several times. The attribute name "*ss*" is not explicitly stated, but is specified as nameless. As an example

  ```
  .EntityName:width=20.0:10.0
  ```

will result in the error message

- Error in line nn: The attribute ‘width’ with type ‘Real’ is already set for the present tag.

  as all *REAL*s are regarded as nameless `width` attributes.

- The attribute ’*ss*’ has type ’*tt*’, but the given value has type ’*uu*’.

  An attribute is specified as *ss*=*vv*. The “*vv*” value is a “*uu*” type, but the “*ss*” attribute must be given a value of the “*tt*” type. See the reference section for the current tag to see the type of the “*ss*” attribute.

- The Boolean attribute ’ss’ does not exist.

  An attribute such as *ss*+ or *ss*- has been specified, but no attribute with the name “*ss*” exists.

- The attribute ’*ss*’ is not defined for the tag ’*tt*’.

  An attempt was made to use the “*ss*” attribute in the ”*tt*” tag, but it does not exist. See the reference section for the current tag for legal attributes.

- The attribute ’*ss*’ does not have type Boolean.

  The “*ss*” attribute is specified as *ss*+ or *ss*-, but “ss” does not have the *BOOLEAN* type. See the reference section for the current tag to see which type the “*ss*” should have been.

- No nameless attribute of type ’*ss*’ is defined for the tag ’*tt*’.

  A nameless attribute of type “*ss*” has been specified for the “*tt*” tag. See the reference section for the current tag for nameless attributes that can be connected to each tag. In the table in “Attribute Types,” the type of each attribute is specified.

- A value is missing after the attribute ’*tt*’.

  A “*tt*” attribute is specified without a value. This error message is often caused by a *BOOLEAN* attribute specified without a “+” or “-.”

- RGB values must be between 0 and 100.

  Values for the “`rgb`” attribute are specified as a *TRIPLE* (*r, g, b*), where *r*, *g*, and *b* are numbers stating the intensity of red, green, and blue, respectively, as a percentage. This means that *r*, *g*, and *b* must be between 0 and 100 (both inclusive).

- Negative values cannot be used with attribute ‘height’.

  `height` values cannot be negative.

- Negative values cannot be used with attribute ‘width’.

  `width` values cannot be negative.

- Negative values cannot be used with attribute ‘widthfactor’.

  `widthfactor` values cannot be negative.

- Value out of range for attribute ‘fontsize’.

  Font sizes are specified in pixels and must be larger than 0 and smaller than or equal to 32767.

- The attributes “color” and “rgb” cannot be used simultaneously.

  The `color` and `rgb` attributes are both used for specifying color. It is not allowed to use both attributes in the same tag.

- The mandatory attribute ‘title’ is missing.

Applies to the `layout` tag, MDL versions below 4. The `title` attribute is mandatory.

- The mandatory attribute ‘title’ is missing. The mandatory attribute ‘name’ is missing.—AND—The mandatory attribute ‘windowname’ is missing. The mandatory attribute ‘windowtype’ is missing.

  Applies to the `layout` tag, MDL version 4 and above. The attribute mentioned in the error message is mandatory.

- Action ‘*ss*’ is missing.

  Applies to the `availableactions` or `shortcut` tag, MDL version 4 and above. A reference was made to an undefined action.

- The action ‘*ss*’ is not allowed.

  An action was made available in a context where the action cannot be used.

Deltek is the leading global provider of enterprise software and information solutions for professional services firms, government contractors, and government agencies. For decades, we have delivered actionable insight that empowers our customers to unlock their business potential. Over 14,000 organizations and 1.8 million users in approximately 80 countries around the world rely on Deltek to research and identify opportunities, win new business, optimize resource, streamline operations, and deliver more profitable projects. Deltek – Know more. Do more.®

deltek.com