ISA Dialog Manager

Resource Reference

A.06.03.b

In this manual all resources of the ISA Dialog Manager are explained. Resources are objects like cursors, colors, texts and fonts, which are used to define certain properties concerning the appearance or behavior of IDM objects.

ISA Informationssysteme GmbH

Meisenweg 33

70771 Leinfelden-Echterdingen

Germany

Notation Conventions

DM will be used as a synonym for Dialog Manager.

The notion of UNIX in general comprises all supported UNIX derivates, otherwise it will be explicitly stated.

< >	to be substituted by the corresponding value
color	keyword
.bgc	attribute
{ }	optional (0 or once)
[ ]	optional (0 or n-times)
<A> | <B>	either <A> or <B>

Description Mode

All keywords are bold and underlined, e.g.

variable **integer** **function**

Indexing of Attributes

Syntax for indexed attributes:

[I]

[I,J] meaning [row, column]

Identifiers

Identifiers have to begin with an uppercase letter or an underline ('_'). The following characters may be uppercase or lowercase letters, digits, or underlines.

Hyphens ('-') are ***not*** permitted as characters for specifying identifiers.

The maximal length of an identifier is *31* characters.

Description of the permitted identifiers in the Backus-Naur form (BNF)

<identifier>	::=	<first character>{<character>}
<first character>	::=	_ | <uppercase>
<character>	::=	_ | <lowercase> | <uppercase> | <digit>

<digit> ::= 1 | 2 | 3 | … 9 | 0

<lowercase> ::= a | b | c | … x | y | z

<uppercase> ::= A | B | C | … X | Y | Z

Table of Contents

1 Introduction

In this manual you will find a summary of the definition of the Dialog Manager

- layout resources,
- programming resources,

Every object to be accessed in the Dialog Manager must have a valid identifier. In contrast to objects, resources (with few exceptions) can be accessed only if they have a name. This name has to be unambiguous in the module or dialog.

Resources are objects which are assigned to other objects as attribute values or which are necessary to program a dialog run. These resources are defined with a logic name and may have different physical properties.

The following resources are available for the dialog designer:

- Layout resources
  - Accelerator
    Enables keyboard access to objects.
  - Color
    The object appearance can be defined with different colors.
  - Cursor
    Different cursors indicate certain dialog states to the user, e.g. waiting states.
  - Display
    With display resources windows can be mapped to different screens in multiscreen systems. Multiscreen support is available in the IDM for Motif only.
  - Font
    Different fonts may be assigned to individual objects.
  - Format
    To be able to format strings in editable texts and tablefields, you can define format resources.
  - Text
    Different languages can be defined with text resources.
  - Tile
    Tiles can be defined by means of this resource.
- Programming resources
  - Message
    With this resource messages can be defined that can be sent to objects as external events.
  - Source

This resource is used to mark an object as being moveable per Drag&Drop. This resource defines the action (cut, copy) with which the movement is triggered.

» Target
This resource is used to mark an object as being the target of a Drag&Drop operation.

2 Layout Resources

The layout resources are used to hide operating-system and graphics-system details from the user. This means that every resource gets a DM-internal identifier from the dialog designer. The dialog designer then works with this identifier.

Layout resources are:

- accelerator
- color
- cursor
- display
- font
- format
- text
- tile

The general formalism for the resource definition is as follows:

Resource Definition

```
{export | reexport}  <resource class>  <Identifier>  <resource description>;
```

Note

The keyword **export** means that the corresponding object can even be referenced outside the module where it has been defined. **export** is only allowed, if the object or the resource has been defined within a module.

See Also

Chapter “Modularization” in manual “Programming Techniques”

The definition above allows one single value to be assigned to a resource.

In order to define portable applications for most varying platforms (e.g. as far as different keyboards, screens, languages and window systems are concerned), there is a second method to define several variants for the value of the highly platform-specific resources color, cursor, font and tile.

Since two options are available in the definition of a dialog, a high degree of flexibility is achieved. The simple cases can be covered by the method described above, complex tasks will be solved by the method described in the following.

Note

The programming resources function and variable do not depend on the platform the dialog is executed on. Variant notations are therefore not supported for these resources.

Keyword:

<resource class>

In addition to the above described specification format (e.g. specification of colors using RGB values, or specification of fonts using the logical font format), there is another specification format that allows variants for the resources accelerator, color, cursor, font, text and tile.

The non-variant syntax of the resource definition can be written in the variant notation using the same syntactical elements, as shown in the following definitions.

Non-variant Definition

```
<resource-class>  <resource-label>  <resource-value>;
```

Variant Definition

```
<resource-class>  <resource-label>
{
  0:            <resource-value>;
  <cardinal>:   <resource-value>;
  [<cardinal>:  <resource-value>;]
}
```

The DM stores a variant notation with only one variant 0 as non-variant resource, therefore such a notation has also to be defined as non-variant notation.

The variant 0 has to be defined for all variant resources except for text, where it is not necessary.

The following sections will specify first the non-variant description and then the variant description of each resource.

2.1 accelerator and Mnemonic

Accelerators and Mnemonics enable practiced users to work quickly and efficiently with an application. Accelerators and mnemonics are used primarily in the menu system. The DM supports accelerators and mnemonics for all selectable objects. Their usability, however, depends above all on the used window system.

If accelerators and mnemonics are used, the marginal conditions multilinguality and use of different keyboards have to be considered, because alphanumeric characters are language-dependent and function and special keys are keyboard-dependent. There is no reliable subset of function keys and special keys on keyboards.

Remarks

- The Dialog Manager application and mnemonics do not distinguish between capital and small letters. This is why it makes no difference whether you type “a” or “A”.
- Mnemonics, similar to the accelerators, trigger the object immediately.
- If a mnemonic or accelerator is defined for a non-selectable statictext, the selection of the corresponding mnemonic or accelerator will put the focus on the object following the statictext (provided that this object is visible and selectable).

2.1.1 accelerator

An accelerator is defined as a key or a key combination linked to a dialog object. By pressing an accelerator key or

- alphanumeric keys, which are language-dependent, but may also be keyboard-dependent,
- alphanumeric keys which are not language-dependent, but may be keyboard-dependent, the dialog object linked to this accelerator is selected and the action linked to the dialog object is executed.

To provide an object with an accelerator, you first have to define a resource ***accelerator***. This resource ***accelerator*** consists of the keyword **accelerator**, an identifier, an opening brace, of one or several key specifications, and the closing brace. The key specification is preceded by a number and a colon. This number stands for the keyboard type. The relevant key specification is selected by setting the keyboard type.

By using the key combination the dialog object which is linked to this accelerator will be selected and the action specific to the object will be executed.

Definition

```
{ export | reexport } accelerator <Identifier> <acceleratorSpec> |
<variantDef>

acceleratorSpec ::=
  <key definition> { , "<display name>" | <text resource> } ;
```

```
key definition ::=
  { <modifier> + } { <modifier> + } { <modifier> + }
  & | '<alphanumeric key>' | <function key> | <special key>

modifier ::= alt | cntrl | shift

function key ::= F1 | F2 | … | F98 | F99

variantDef ::=
{
           0 : <acceleratorSpec>
  [ <number> : <acceleratorSpec> ]
}
```

In some dialog objects, the key combination for an accelerator is displayed on the right of the text (depends on the window system).

The optional string or text resource after the actual definition of the button can be used to specify the text that is to appear as accelerator text in the object. This text will then be displayed accordingly in menu items.

In contrast to mnemonics, accelerators do not have a visual feedback. The attribute *.real_visible* has to be *true*, but the object does not have to be in the visible area. Only dialog objects with accelerators in the active window can be selected. The accelerators of all dialog objects in a window have to be clearly defined. If an accelerator is used more than once in a window, it cannot be predicted which accelerator will actually be activated.

Accelerators are language-dependent as well as keyboard-dependent, and are thus treated as normal resources (e.g. colors and fonts), which are hardware-dependent as well.

The DM differentiates three types of keys:

- function keys which are keyboard-dependent;
- alphanumeric keys which are language-dependent, but may be keyboard-dependent;
- alphanumeric keys which are not language-dependent, but may be keyboard-dependent.

The keyboard type can be set if the Dialog Manager is called with the option

```
-IDMkeyboard [<Number>]
```

0 is the default keyboard type, which is used if no keyboard type was set or if the accelerator does not have a variant for the chosen keyboard type. This is why it is useful to use the 0 variant in the accelerator definition in general
An accelerator key combination consists of a key which can be preceded by an arbitrary number of modifier keys with the + symbol. Modifier keys are *cntrl*, *alt*, and *shift*. Alphanumeric keys are specified by 'single quote - character - single quote'.

Example

```
accelerator A
```

```
{
  0:  alt+'a';
}
```

In the current Motif version of the Dialog Manager, accelerators are labeled only in the object menu-item.

Example

```
accelerator A_QUIT
{
  0:  cntrl+F2;
  1:  cntrl+F5;
}
```

If an accelerator shall be language-dependent and shall change automatically if the language changes, a “&” has to be specified in place of the key. This specification has the effect that the mnemonic character for the dialog object is also used for the accelerator (provided that this object has a mnemonic).

Example

```
accelerator A_QUIT
{
  0:  cntrl+&;
  1:  cntrl+alt+&;
}
text  "E&xit"
{
  1:  "&Beenden";
  2:  "&Finito";
}
child menuitem M1
{
  .text         E&xit;
  .accelerator  A_QUIT;
}
```

This resource, identified as A_QUIT, can now be assigned to a dialog object. The dialog object is automatically labeled with the accelerator text (provided that the used window system allows the labeling for this object).

Remark

The key combination `Ctrl` + `Alt` + `Del` should never be used.

Modifier keys cannot be used as accelerators. Individual alphanumeric characters and cursor keys should not be used alone as accelerators, i.e., they should be used in combination with other keys.in Kombination mit anderen Tasten verwendet werden.

Keys or key combinations already having a predefined function in the corresponding window system should not be used.

Please also refer to the section “Note on problematic situations”.

Alphanumeric characters cannot be specified in connection with the *shift* modifier.
Function keys and **special keys** can be specified by the following identifiers:

BackSpace	/* back space, back char */
Begin	/* BOL */
Break	
Cancel	/* Cancel, stop, abort, exit */
Clear	
Delete	/* Delete, rubout */
Down	/* Move down, down arrow */
End	/* EOL */
Escape	
Execute	/* Execute, run, do */
Find	/* Find, search */
Home	
Insert	/* Insert, insert here */
Left	/* Move left, left arrow */
Linefeed	/* Linefeed, LF */
Next	/* Next, Page Down */
Pause	/* Pause, hold, scroll lock */
Print	
Prior	/* Prior, previous, PageUp */
Redo	/* redo, again */
Return	/* Return, enter */
Right	/* Move right, right arrow */
Select	/* Select, mark */
Tab	

Undo	/* Undo previous*/
Up	/* Move up, up arrow */

Function Keys:

F1 - F99

Modifier Keys:

- alt
- cntrl
- shift

Note concerning the Caps Lock Key

The `Caps Lock` key will be ignored by the function keys. If you want to use the combination `Shift` + `F3`, you must use the `Shift` key, it is not possible to use the `Caps Lock` key to trigger the accelerator.

Note on problematic situations

System accelerators should not be used, otherwise multiple triggering or interception of the event (on key or select) may occur; moreover, this may vary depending on the accelerator and window system.

Examples

- Under Microsoft Windows, both `F1` and `Ctrl` + `F1` are system accelerators and should therefore not be used. In both cases, an on help event is sent to the respective object, but the event is generated in Microsoft Windows itself.
- The `F10` accelerator calls the system menu of the active window under Windows, but in most cases the menu is not opened (it opens when you press the `Curosr Down` arrow key after the accelerator). It should be noted that the event processing of the application is stopped as soon as the system menu of the window is called.

In addition, the use of an accelerator

- on several objects in one *on key* rule
- on one or more objects in an *on key* rule and assigned attribute *.accelerator* (triggering the *select* event)

should be carefully considered. Depending on the window system used, the current dialog state, and the way or order in which the windows are made visible, the accelerator concerned could be triggered twice on different objects or on the same object.

2.1.1.1 Menus and accelerators under Microsoft Windows

Under Microsoft Windows, accelerators are used to select menu functions without opening a menu. As a result, the accelerators are no longer triggered when a menu is open. Instead, you can now use the respective mnemonics to select a menu item.

2.1.2 Mnemonic

Mnemonics do not belong to the resources as such; they have to be defined as attributes. Because of their close relationship to accelerators, their definition is described here.

A mnemonic is exactly one alphanumeric character which is marked in the text of a dialog object. The corresponding object can be activated with this character.

Example

In a menubox: **F**ile

In a pushbutton: **E**xit

The corresponding menu opens by hitting a special key, the so-called mnemonic key, together with the character marked in the menu bar. A menuitem can now be activated by hitting a key which corresponds to the character marked in the menu. No special key needs to be hit now.

Usually mnemonics have a visual feedback. This means that the activated dialog object appears on screen in a selected state, and the action linked to the object is executed by pressing the assigned key. Only the dialog objects located in the currently active window can be activated.

The mnemonic character is always taken from the language-dependent text and is specified together with this text. The letter following the *&* symbol is the mnemonic character. It appears highlighted on the screen. The *&* symbol will not be displayed. If *&* shall appear in the dialog object on the screen, two *&* symbols in succession have to be specified.

The language is set if the DM is called with the option

```
-IDMlanguage [<Number>]
```

Example

```
text  "E&xit"
{
  1:  "&Beenden";
  2:  "&Finito";
}
child menuitem M1
{
  .text  E&xit;
}
```

The availability of mnemonics depends on the used window system and will only work if the object can be selected by the user.

Microsoft Windows

Mnemonics are supported for the following objects (if *.real_sensitive* = *true*):

- image
- checkbox
- menubox
- menuitem
- pushbutton
- poptext
- radiobutton
- statictext

Mnemonics can be activated by pressing

- `Alt` together with the mnemonic character or
- first `Alt` and then the mnemonic character.

As an alternative, you can also use the function key `F10`.

Mnemonic characters must be used without `Alt`, if a menu is highlighted.

Motif

Mnemonics are supported for the following objects:

- menubox
- menuitem
- poptext item
- pushbutton

Mnemonics can be activated by pressing

- `Alt` together with the mnemonic character.

If a menu is open, mnemonic characters must be used without `Alt`.

You can only choose the mnemonics which are visible on screen.

Qt

Mnemonics are supported for the following objects:

- ***Checkbox***
- ***Menubox***
- ***Menuitem***
- ***Pushbutton***
- ***Radiobutton***
- ***Statictext***

2.2 color

All colors you want to use have to be declared as a resource. The number of available colors depends on your graphics system.

The declaration of a color starts with the keyword **color**, followed by the color identifier and the color definition. The declaration ends with a ; (semicolon).

Definition

```
{ export | reexport } color <Identifier> <colorSpec> | <variantDef>

colorSpec ::=
  { rgb(<R>, <G>, <B>) | hls(<H>, <L>, <S>) | gradient("<kind> [ , <arg> ]")
|
    "<predefined color>" }
  { , grey(<N>) } { , black | white } ;

variantDef ::=
{
          0 : <colorSpec>
  [ <number> : <colorSpec> ]
}
```

Examples

Non-variant

```
color RED         rgb(255, 0, 0), grey(50), white;
color AQUA        hls(90, 127, 255), grey(127), white;
color YELLOW      rgb(255, 255, 0);
color BLUE        "blue", black;
color ClMagenta   "magenta";
color BEAUTYCOLOR "orchid", grey(10), white;
```

Variant

```
color BEAUTYCOLOR
{
  0: rgb(100,200,200);
  1: "orchid", grey(10), white;
  2: rgb(150,100,190);
  3: "pink";
}
```

Different kinds of displays use the color specifications in a different way. Color displays use exclusively the color specification, greyscale displays use greyscales, and monochrome displays use black-and-white values.

rgb(<R>, <G>, <B>)

R, G, B are the color intensities of the portions of the colors **r**ed, **Gg**reen and **b**lue. The value range for all three values is 0 to 255. Brackets and commas are components of the color declaration.

Depending on your choice of RGB values, Microsoft Windows uses grid mixed colors. This can mostly be avoided by choosing suitable values, e.g. 0, 63, 127, 191, 255.

```
color  MyGrey  rgb(127,127,127);
color  Azure  rgb(63,127,191);
```

hls(<H>, <L>, <S>)

In the HLS color model, colors are defined by hue **H**, lightness or luminance **L**, and saturation **S**. The range for all three values is *0* to *255*.

The colors are arranged within a cylinder whose top plane is white and whose bottom plane is black. Lightness L increases from *0* (dark) at the bottom to *255* (bright) at the top. For colors within the circle intersecting the center point of the cylinder, L is *127*. These colors are neither brightened (by mixture with white) nor darkened (by mixture with black).

Hue H is determined by the angle on the circular area. The range for H from *0* to *255* and the angle A in the color circle (actually ranging from *0* to *359* degrees) relate as given below:

- H = A / 2.
- A = (H modulo 180) * 2; that is the values from *180* to *255* match the color shades from *0* to *75*.

Red has a value of *H = 0* (or *180*), *H = 60* (or *240*) yields green and *H = 120* yields blue.

The shades of gray lie on the vertical line through the center of the cylinder, all having a saturation of *S = 0*. The colors on the lateral area of the cylinder have the highest saturation *S = 255*. The colors on the perimeter of the middle circle with *L = 127* and *S = 255* sometimes are referred as pure colors.

grey(<N>)

This is the grayscale value which is to be used on a grayscale screen for the color. N is a number from 0 to 255.

black | white

These are the color values for monochrome monitors.

gradient("<kind> [, <arg>]")

Availability

IDM FOR QT only

color resources support gradients on Qt.

The parameter <*kind*> defines the gradient type. Only one gradient type may be specified.

The other parameters may include:

» Supplementary parameters for the gradient type
» Stop point definitions
» Color definitions

These **gradient types** are available:

***Table 1:** Types of gradients*

Gradient	Definition	Explanation
linear	`gradient("Linear, ...")` `gradient("LinearV, ...")`	vertical gradient
	`gradient("LinearH, ...")`	horizontal gradient
radial	`gradient("Radial, ...")` `gradient("Radial, <R>, ...")`	radial gradient with default radius 50% radial gradient with **supplementary parameter** radius R% radius is given as a percentage of the available space
conical	`gradient("Conical, ...")` `gradient("Conical, <S>, ...")`	conical gradient with start at 90° conical gradient with **supplementary parameter** S°, which indicates the starting angle

The **color definitions** are always appended after the gradient type and any supplementary parameters. You can use color names, HTML notation, and the *rgb(…)*, *hls(…)*, and *grey(…)*notations known for color resources.

In addition, a **stop point** can be specified for each color definition, which determines the weighting of the color. A stop point is a percentage with an optional percentage sign that always precedes each color and affects how much space that color occupies in the gradient. A gradient starts at *0%* and ends at *100%*, based on the area it fills. The specification *… , 20%, green, ...* means, for example, that after *20%* of the area to be filled, the color green is set. If no other color is set for the *0–20%* range, the first *20%* of the range is colored green. If another color is already set before *20%*, then a transition between this color and green is displayed.

Stop points should always be set in ascending order. If several colors are defined with the same stop point, the color that is furthest at the end of the parameter list applies. For example, the definition *... , 20%, green, 40%, blue, 20%, red, ...* produces a gradient with a shade of red at *20%*, which changes to a blue tone that is shown as saturated at *40%*.

Examples

```
gradient("Linear, green, yellow, red");  named colors, evenly distributed
gradient("Linear, #00FFFF, #00FF00, #FF0000");  HTML notation,
                                                colors evenly distributed
```

```
gradient("Linear,red, #00FF00, rgb(0,0,255)");  mixed notations,
                                                colors evenly distributed
gradient("Linear, 20%, green, 60%, yellow, 80%, red");  named colors with
                                                        percentage stop
points
gradient("Linear, 20, green, 60, yellow, 80 ,red");  the % sign at the
                                                     stop points is
optional
```

Notes

Qt allows the setting of gradients in most places, but does not necessarily apply them. Whether a gradient is displayed depends very much on the object and the UI style. In this case, areas (e.g. backgrounds) are usually unproblematic, with delicate structures (e.g. texts) the gradient is often replaced by a single color. For grouping objects, gradients as a background are usually displayed.

<predefined color>

A color identifier defined by the window system that can be used in the ISA Dialog Manager.

2.2.1 Predefined Colors

A list of the predefined cursors available on the respective system can be requested via the attribute .colorname[integer] on the setup object.

2.2.1.1 Independent UI-colors

The UI color resources are WSI-independent predefined colors that are available on all systems with similar characteristics or meaning. The default colors used by the current desktop/window manager or theme are used for this purpose. The uniform COLOR resources are defined as follows:

- UI_BG_COLOR
  (general background color)
- UI_FG_COLOR
  (general foreground/text color)
- UI_BUTTONBG_COLOR
  (background color of button-like widgets)
- UI_BUTTONFG_COLOR
  (foreground/text color of button-like widgets)
- UI_INPUTBG_COLOR
  (background color of input/selection widgets)
- UI_INPUTFG_COLOR
  (foreground/text color of input/selection widgets)
- UI_ACTIVEBG_COLOR
  (background color of active elements of input/selection widgets)

- UI_ACTIVEFG_COLOR
  (foreground/text color of active elements of input/selection widgets)
- UI_MENUBG_COLOR
  (background color of menus)
- UI_MENUFG_COLOR
  (foreground/text color of menus)
- UI_TITLEBG_COLOR
  (background color of title bars, dialog frames)
- UI_TITLEFG_COLOR
  (foreground/text color of title bars, dialog frames)
- UI_HEADERBG_COLOR
  (background color of table headers)
- UI_HEADERFG_COLOR
  (foreground/text color of table headers)
- UI_BORDER_COLOR
  (border color)
- UI_NULL_COLOR
  (no color)

Note for Windows

The UI_*_COLOR colors access the Windows system colors. It should be noted that with the introduction of Visual Styles, the individual objects no longer use the system colors, but rather bitmaps defined by the theme. Depending on the selected theme there can be strong discrepancies. Therefore no colors (or UI_NULL_COLOR) should be set under Windows mach Möglichkeit. On the other hand, it should be noted that with Windows 10 the color variety has been reduced. For example, the colors UI_HEADERBG_COLOR, UI_HEADERFG_COLOR, UI_MENUBG_COLOR, UI_MENUFG_COLOR, UI_TITLEBG_COLOR and UI_TITLEFG_COLOR are no longer supported. This means that these colors can still be used, but they are not used by any object.

2.2.1.2 Motif (X-Windows)

When using system names for color resources, it is possible to address Motif's default colors. The following system names are available:

- BACKGROUND
- FOREGROUND
- TOP_SHADOW
- BOTTOM_SHADOW
- SELECT
- ACTIVE_BACKGROUND

- ACTIVE_FOREGROUND
- ACTIVE_TOP_SHADOW
- ACTIVE_BOTTOM_SHADOW
- ACTIVE_SELECT
- TEXT_BACKGROUND
- TEXT_FOREGROUND
- TEXT_TOP_SHADOW
- TEXT_BOTTOM_SHADOW
- TEXT_SELECT
- INACTIVE_BACKGROUND
- INACTIVE_FOREGROUND
- INACTIVE_TOP_SHADOW
- INACTIVE_BOTTOM_SHADOW
- INACTIVE_SELECT
- SECONDARY_BACKGROUND
- SECONDARY_FOREGROUND
- SECONDARY_TOP_SHADOW
- SECONDARY_BOTTOM_SHADOW
- SECONDARY_SELECT

On Motif (X Windows) systems all colors from the color definition file **rgb.dat** can be used.

- black / Black
- blue / Blue
- light blue / LightBlue
- medium blue / MediumBlue
- midnight blue / MidnightBlue
- navy blue / NavyBlue
- navy / Navy
- sky blue / SkyBlue
- coral / Coral
- cyan / Cyan
- gold / Gold
- yellow / Yellow
- green / Green
- dark green / DarkGreen

- forest green / ForestGreen
- lime green / LimeGreen
- pale green / PaleGreen
- spring green / SpringGreen
- dark slate grey / DarkSlateGrey
- dim grey / DimGrey
- light grey /LightGrey
- magenta/ Magenta
- maroon / Maroon
- orange / Orange
- pink / Pink
- red / Red
- indian red / IndianRed
- orange red /OrangeRed
- violet red / VioletRed
- salmon / Salmon
- sienna / Sienna
- tan / Tan
- turquoise / Turquoise
- violet / Violet
- blue violet / BlueViolet
- wheat / Wheat
- white / White

2.2.1.3 Microsoft Windows

- BLACK
- BLUE
- CYAN
- DARKBLUE
- DARKCYAN
- DARKGREEN
- DARKGREY
- DARKPINK
- DARKRED
- GREEN

- PINK
- RED
- WHITE
- YELLOW

These system colors are available in the IDM FÜR WINDOWS:

- CLR_ACTIVEBORDER
- CLR_ACTIVETITLE
- CLR_ACTIVETITLETEXT
- CLR_APPWORKSPACE
- CLR_BACKGROUND
- CLR_BUTTONFACE
- CLR_BUTTONHIGHLIGHT
- CLR_BUTTONSHADOW
- CLR_BUTTONTEXT
- CLR_GRAYTEXT
- CLR_HIGHLIGHT
- CLR_HIGHLIGHTTEXT
- CLR_INACTIVEBORDER
- CLR_INACTIVETITLE
- CLR_INACTIVETITLETEXT
- CLR_MENU
- CLR_MENUTEXT
- CLR_SCROLLBAR
- CLR_WINDOW
- CLR_WINDOWFRAME
- CLR_WINDOWTEXT

2.2.1.4 Qt

As color identifiers all SVG color names can be used (http://www.w3.org/TR/SVG/types.html#ColorKeywords), where upper and lower case is ignored. When using these color names (for example, green), the colors are realized according to the SVG color names of the W3C. Therefore, some colors differ in tone and intensity from the X11 colors.

In addition, the following Qt default colors depending on the current settings are available:

- BASE background of input fields and certain container objects (lists)
- BASETEXT foreground of input fields and certain container objects (lists)
- BUTTON button background
- BUTTONTEXT button foreground
- HIGHLIGHT background for selection and active entry
- HIGHLIGHTTEXT foreground for selection and active entry
- SHADOW shadow color, mostly very dark to black
- WINDOW window background
- WINDOWTEXT window foreground
- BRIGHTTEXT Text color for low contrast
- ALTERNATEBASE alternatively background color (mostly for tables)
- TOOLTIPBASE tooltip background
- TOOLTIPTEXT tooltip foreground
- LINK link foreground
- LINKVISITED visited link foreground
- PLACEHOLDERTEXT placeholder foreground for various input windgets, since Qt 5.12
- ACTIVE_BASE
- ACTIVE_BASETEXT
- ACTIVE_BUTTON
- ACTIVE_BUTTONTEXT
- ACTIVE_HIGHLIGHT
- ACTIVE_HIGHLIGHTTEXT
- ACTIVE_SHADOW
- ACTIVE_WINDOW
- ACTIVE_WINDOWTEXT
- ACTIVE_BRIGHTTEXT
- ACTIVE_ALTERNATEBASE
- ACTIVE_TOOLTIPBASE
- ACTIVE_TOOLTIPTEXT
- ACTIVE_LINK
- ACTIVE_LINKVISITED
- ACTIVE_PLACEHOLDERTEXT since Qt 5.12
- DISABLED_BASE
- DISABLED_BASETEXT

- DISABLED_BUTTON
- DISABLED_BUTTONTEXT
- DISABLED_HIGHLIGHT
- DISABLED_HIGHLIGHTTEXT
- DISABLED_SHADOW
- DISABLED_WINDOW
- DISABLED_WINDOWTEXT
- DISABLED_BRIGHTTEXT
- DISABLED_ALTERNATEBASE
- DISABLED_TOOLTIPBASE
- DISABLED_TOOLTIPTEXT
- DISABLED_LINK
- DISABLED_LINKVISITED
- DISABLED_PLACEHOLDERTEXT since Qt 5.12
- INACTIVE_BASE
- INACTIVE_BASETEXT
- INACTIVE_BUTTON
- INACTIVE_BUTTONTEXT
- INACTIVE_HIGHLIGHT
- INACTIVE_HIGHLIGHTTEXT
- INACTIVE_SHADOW
- INACTIVE_WINDOW
- INACTIVE_WINDOWTEXT
- INACTIVE_BRIGHTTEXT
- INACTIVE_ALTERNATEBASE
- INACTIVE_TOOLTIPBASE
- INACTIVE_TOOLTIPTEXT
- INACTIVE_LINK
- INACTIVE_LINKVISITED
- INACTIVE_PLACEHOLDERTEXT since Qt 5.12

Normal and active states are identical.

The identifier *DEFAULT* serves as null color. If this color is assigned, the respective color is reset to the corresponding default color (depending on the current system settings).

2.2.2 Dynamically changeable attributes

The table below shows those atrributes of ***color*** resources, that can be changed dynamically at runtime. Please note, that *.rgb[enum]*, *.hls[enum]* and *.name* are mutually exclusive so that "get" returns *"can't get value"*.

Table 2: *Modifiable attributes of the color resource*

Attribute	Data Type	Index Range	Description
.rgb[enum]	*integer* *0 … 255*	*color_red*, *color_green*, *color_blue*	red, green, blue
.hls[enum]	*integer* *0 … 255*	*color_hue*, *color_light*, *color_sat*	hue, lightness, saturation
.gradient	*string*	–	kind and parameters of a gradient
.name	*string*	–	system name of the color
.grey	*integer* *0 … 255*	–	grey scale value
.bw	*enum* *color_black*, *color_white*	–	black and white value

With the attributes *.rgb[enum]* and *.hls[enum]* the color is not updated before the last value (*color_blue* respectively *color_sat*) is set, thus giving the opportunity to define the color as a whole.

2.3 cursor

The graphic cursors you want to use have to be defined before. The keyword is **cursor**, followed by the cursor identifier and the cursor definition.

Definition

```
{ export | reexport } cursor <Identifier> <cursorSpec> | <variantDef>

cursorSpec ::= <cursorBitmap> | "<predefined cursor>" ;

cursorBitmap ::= <x>, <y>,
                 <cursorString>
             [ , <cursorString> ]

variantDef ::=
{
           0 : <cursorSpec>
  [ <number> : <cursorSpec> ]
}
```

Example

Non-variant

```
cursor MyCursor "XC_cross";
```

Variant

```
cursor MyCursor
{
  0: "XC_cross";
  1: "Cross";
}
```

2.3.1 Bitmap-Cursor

<x> and <y> identify – as in tiles – the corresponding extensions (in pixels) on the x- and y-axes.

The IDM allows any widths (<x>) and heights (<y>) for the cursors. Some graphics systems have a cursor size of 32 * 32 pixels (e.g. MS-Windows). Other graphics systems, however, limit the cursor size to 16 * 16 pixels (e.g. Motif).

<cursorString> identifies a character string which consists of exactly <x> characters. You should specify exactly <y> strings with <x> characters each.

Permitted characters in <cursorString> are point (“.”), space ( ), plus (“+”), hash (“#”), and the uppercase letter X. The individual characters mean the following:

“.”

“ ” (space)

The cursor will adopt the background color, i.e. the cursor will be transparent at this point.

“+”

At this point the cursor will be displayed in the inverse background color. This is important if the cursor foreground color is identical with the background color.

“#”

At this point the cursor will be displayed in the foreground color which is defined by the window system.

“X”

This character marks the hotspot of the cursor. If you click on an object with the cursor, the hotspot has to touch the object, otherwise the clicking will not be registered. This character has to occur only once in the cursor definition.

Example

```
cursor Pointer 16, 16,
  "........+.......",
  ".......+++......",
  "......++X++.....",
  ".....++#####....",
  "....++#######...",
  "...++#########..",
  "..++##########+.",
  "..+++++#########",
  ".....++#####....",
  ".....++#####....",
  ".....++#####....",
  ".....++#####....",
  ".....++#####....",
  ".....++#####....",
  ".....++#####....",
  ".....+++++++....";
```

Note for the IDM for Windows

Cursor means mouse pointer, regardless of the different meaning that cursor has in Microsoft Windows.

The cursor size in Microsoft Windows is 32 x 32 (with EGA or VGA resolution). No other size is allowed! If the cursor exceeds the permitted size, the DM converts the cursor by filling the cursor with transparents or by clipping the right and the bottom border.

2.3.2 Predefined Cursor

A list of the predefined cursors available on the respective system can be requested via the attribute .cursorname[integer] on the setup object.

2.3.2.1 Independent UI-Cursor

The UI cursor resources are WSI-independent predefined cursors that are available on all systems with similar characteristics or meaning. The default cursors used by the current desktop/window manager or theme are used for this purpose. The uniform CURSOR resources are defined as follows:

- UI_CURSOR
  (default cursor mostly arrow)
- UI_IBEAM_CURSOR
  (Edittext insertion marker)
- UI_WAIT_CURSOR
  (indication that the application is busy)
- UI_CROSS_CURSOR
  (Cross)
- UI_UP_CURSOR
  (arrow up)
- UI_SIZEDIAGF_CURSOR
  (sizing arrow from top left to bottom right)
- UI_SIZEDIAGB_CURSOR
  (sizing arrow from bottom left to top right)
- UI_SIZEVER_CURSOR
  (sizing arrow from left to right)
- UI_SIZEHOR_CURSOR
  (sizing arrow from top to bottom)
- UI_MOVE_CURSOR
  (sizing arrow in all directions)
- UI_STOP_CURSOR
  (forbidden sign/ crossed out circle)
- UI_HAND_CURSOR
  (hand symbol/ index finger)
- UI_HELP_CURSOR
  (arrow with question mark)

2.3.2.2 Motif

When the cursor is defined with a name, the corresponding X Cursor is loaded from the X Windows cursor fonts:

- XC_X_cursor
- XC_arrow
- XC_based_arrow_down
- XC_based_arrow_up
- XC_boat
- XC_bogosity
- XC_bottom_left_corner
- XC_bottom_right_corner
- XC_bottom_side
- XC_bottom_tee
- XC_box_spiral
- XC_center_ptr
- XC_circle
- XC_clock
- XC_coffee_mug
- XC_cross
- XC_cross_reverse
- XC_crosshair
- XC_diamond_cross
- XC_dot
- XC_dotbox
- XC_double_arrow
- XC_draft_large
- XC_draped_box
- XC_exchange
- XC_fleur
- XC_gobbler
- XC_gumby
- XC_hand1
- XC_hand2
- XC_heart

- XC_icon
- XC_iron_cross
- XC_left_ptr
- XC_left_side
- XC_left_tee
- XC_leftbutton
- XC_ll_angle
- XC_lr_angle
- XC_man
- XC_middlebutton
- XC_mouse
- XC_pencil
- XC_pirate
- XC_plus
- XC_question_arrow
- XC_right_ptr
- XC_right_side
- XC_right_tee
- XC_rightbutton
- XC_rtl_logo
- XC_sailboat
- XC_sb_down_arrow
- XC_sb_h_double_arrow
- XC_sb_left_arrow
- XC_sb_right_arrow
- XC_sb_up_arrow
- XC_sb_v_double_arrow
- XC_shuttle
- XC_sizing
- XC_spider
- XC_spraycan
- XC_star
- XC_target
- XC_tcross

- XC_top_left_arrow
- XC_top_left_corner
- XC_top_right_arrow
- XC_top_side
- XC_top_tee
- XC_trek
- XC_ul_angle
- XC_umbrella
- XC_ur_angle
- XC_watch
- XC_xterm

Example

```
cursor Clockcursor "XC_clock";
```

2.3.2.3 Qt

There are the following named cursors where the corresponding Qt CursorShape is loaded:

- arrow
- blank
- busy
- closedhand
- cross
- dragcopy since Qt 4.7
- dragmove since Qt 4.7
- draglink since Qt 4.7
- forbidden
- ibeam
- openhand
- pointinghand
- sizeall
- sizebdiag
- sizefdiag
- sizehor
- sizever

- splith
- splitv
- uparrow
- wait
- whatsthis

The appearance of the cursors may vary depending on the UI style.

2.3.2.4 Microsoft Windows

When the cursor is defined with a name, the corresponding cursor is taken from the Microsoft Windows objects.

The following cursors are predefined system cursors on Microsoft Windows:

- APPSTARTING
- ARROW
- CROSS
- HAND
- HELP
- IBEAM
- ICON
- NO
- SIZE
- SIZEALL
- UPARROW
- WAIT
- SIZENESW
- SIZENS
- SIZEWE
- SIZENWSE

Notes

- If you have used the identifier HAND for an **individually defined** cursor then the system cursor will be loaded **instead** of your individual cursor when using IDM A.05.02.f or higher.

2.4 display

The ***display*** resource is used for the allocation of a window to the screen stated in the resource definition or set by the *.screen* attribute of the display resource. It is meant to be used in **multiscreen environments** (MOTIF only). The *.screen* attribute can be changed dynamically in Rule Language to move a window to another screen. The actually selected screen can be queried via the *.real_screen* attribute.

Definition

```
{ export | reexport } display <Identifier> { screen(<number>) } ;
```

attributes

Attribut	RSD	PSD	Eigenschaft	Kurzbeschreibung
.screen	integer	integer	S,G/-/-	screen number
.real_screen	integer	integer	-,G/-/-	actual screen number

In cases of invalid screen numbers the window is displayed on the default screen (usually, but not necessarily, 0). Valid screen numbers are greater than or equal to 0. This means that e.g. when stating -1, the default screen is used.

Particularities

The .screen attribute can also be implemented dynamically by rule language.

Availability

Multiscreen support is only available with IDM for MOTIF. Valid screen numbers can be determined with the program **xdpyinfo** and the .screen[integer] attribute of the setup object.

See also

Chapter „Multiscreen Ssupport untder Motif“ in manual „Programmiertechniken“

2.4.1 Example

The dialog shows how two windows can be placed on different screens and how e.g. a single-screen configuration can be handled.

```
dialog MultiScreen

display DpyDef screen(-1);
display DpyPrim screen(0);
display DpySec screen(1);
```

```
default window WINDOW
{
  .width 200;
  .height 100;

  on close { exit(); }
}

// primary window on screen#0
// (not necessarily the default screen!)
window WiPrim
{
  .display DpyPrim;
  .title "Primary Window";

  statictext StDefScreen { }
}

// secondary window on screen#1
window WiSec
{
  .display DpySec;
  .title "Secondary Window";

  listbox LbScreens
  {
    .xauto 0;
    .yauto 0;

    // move secondary window to another screen dynamically
    on select
    {
      this.window.display.screen := this.userdata[thisevent.index];
    }
  }
}

on dialog start
{
  variable integer I;
  StDefScreen.text := "Default Screen: " + setup.screen;

  // position second window below primary
  // when running in single-screen configuration
  if (setup.screencount = 1) then
    WiSec.ytop := WiPrim.ytop + WiPrim.real_width + 50;
  endif
```

```
  // list available screens
  for I:=1 to setup.screencount do
    LbScreens.content[I] := "Screen#" + setup.screen[I] + "  " +
      setup.screen_width[I] + "x" + setup.screen_height[I];
    LbScreens.userdata[I] := setup.screen[I];
  endfor

  LbScreens.activeitem := LbScreens:find(.userdata, DpySec.real_screen);
}
```

2.5 font

All fonts you want to use have to be declared as a resource. The number of displayable fonts depends on the graphics system you use. The declaration of a font begins with the keyword **font**, followed by the identifier describing the font within the DM and by the font definition.

Definition

```
{ export | reexport } font <Identifier> <fontSpec> | <variantDef> ;

fontSpec ::=
  "<fontName>" { , <size> { , <modifier> } } |
  "<Predefined UI-Font>" |
  "<X Logical Font Description (XLFD)>" |                          // Motif,
Qt
  "<fontName> [ , <parameter> ]" { , <modifier> } |                // Qt
  "<fontName>" { , <size> { , <modifier> { + <modifier> } } } |
  "<size>.<fontName>" | "<systemFont>"                             // Windows
  { x:= * <xscale> % + <xoffset> | / <xdivider>} { y:= * <yscale> %
        + <yoffset> | / <ydivider>}
  { propscale }
  { r:= "<RefString >" } ;

variantDef ::=
{
          0 : <fontSpec>
  [ <number> : <fontSpec> ]
}
```

The name determined by the window system as well as size and attributes of the font are specified in the font definition.

Depending on your graphics system, it may not be possible to define font size and attributes independently of the selected font. In this case only the first font variant can be defined. Other following font sizes and attributes are ignored.

In addition to the system's means of viewing the available fonts, the list of fonts can also be viewed in the IDM Editor. Depending on the system, it is possible to filter the list of fonts by UI Fonts, XFT and XLFD.

Examples

Non-variant

```
font BIG          "white_shadow-48";
font KilterBold   "-adobe-courier-bold-r-normal*";
font KilterItalic "kilter 12i";
font UI "UI_FONT";
```

```
// Defines the font vtsingle of the window system X as font NORMAL in the IDM
font NORMAL "vtsingle";

// Defines the font Courier in size 12 as resource Courier
font Courier "Courier", 12;

// As in the previous definition, with the additional attribute bold
font Bold "Courier", 12, bold;

// Defines the font with the system font usually used for UI objects;
// additionally with the attribute propscale to control the font-raster
font UIpropscale "UI_FONT" propscale;

// Defines an XFT font (e.g. on an Ubuntu system); additionally the size 12
// and the attribute bold for bold display are defined
font XFT "DejaVu Sans", 12, bold;
```

Variant

```
font BIG
{
  0: "white_shadow-48";
  1: "walla_walla-12";
  2: "helvetica", 24, bold;
  3: "harakiri",  48, roman;
}
```

2.5.1 Calculating the Grid Size from a Reference Font

On calculating the grid size from a reference font, you can specify correction factors at the font definition and thus can arbitrarily change the calculation base for the grid which was determined by the font.

It is also possible to specify a reference string for calculation of grid width. The grid width will then be calculated as width of the entire string, divided by the number of characters (with mathematical rounding). Specifying a reference string makes the calculation of grid width independent of the average and maximum character widths from the font description, which may vary between different systems. A reference string is defined by adding `r:="<RefString>"` to the font definition.

Based on the above font definition the grid size calculated as follows:

```
xraster = ((width of font) * xscale) / 100 + xoffset
yraster = ((height of edittext with this font) * yscale) / 100
          + yoffset
```

The effect of this calculation that the font size will be provided with the indicated factor and will then be enlarged according to the given constant.

The <xdivider>/<ydivider> allows the division of a raster without rounding effects. The IDM ensures that a raster in the full divider size is at least as large as the raster without divider. This ensures a complete object representation.

```
xraster = ((width of font) + (xdivider - 1 )) / xdivider
yraster = ((height of edittext with this font) + (ydivider - 1))) / ydivider
```

Example

```
font Font "6x13" x:=*150%+2 y:=*200%+10;
```

In this case, the grid size resulting from this font is calculated as follows:

```
xraster = (6 * 150) / 100 + 2
        = 11
yraster = (13 * 200) / 100 + 10
        = 36
```

Important: improved calculation of the raster width under Windows

With IDM version A.06.03.a, the calculation of the raster width has been substantially changed. If no reference string is set, the raster width is now calculated from an internal reference string (“M”) to avoid excessive width growth due to excessively wide letters within a font.

For compatibility reasons, however, the *opt_fontraster_compat* option, the **-IDMfontraster_compat** startup option, or the IDM_FONTRASTER_COMPAT environment variable can temporarily reactivate the old raster width calculation (with the drawback that excessive width growth may occur again).
When using the *opt_fontraster_compat*, the size calculation is partly based on the system font, which is not High DPI capable, so High DPI capable applications created with IDM FOR WINDOWS 11 should not use this option.

2.5.2 Predefined UI-Fonts

The UI font resources are WSI-independent predefined fonts that are available on all systems with similar characteristics or meaning. The default fonts used by the current desktop/window manager or theme are used for this purpose. The uniform FONT resources are defined as follows:

- UI_FONT
  (font used by default for UI elements)
- UI_FIXED_FONT
  (font with a fixed letter width)
- UI_MENU_FONT
  (font used for menus)
- UI_TITLE_FONT
  (font used in the window header)

» UI_SMALLTITLE_FONT
(font used in a small/low window header - if supported by WSI)

» UI_STATUSBAR_FONT
(font used in the status bar - if supported by WSI)

A list of the fonts available on the respective system can be requested via the attribute .fontname [integer] on the setup object.

2.5.3 Font Definition

There are the following definition methods for a font:

» By specifying a string and additional modifiers according to the following pattern:

```
fontSpec ::=
        "<fontName>" { , <size> { , <modifier> { + <modifier> } } } ;
```

In doing so, you can choose a font with a defined name, size (in points), and corresponding modifiers - in form of font weight or font face/slant.
Valid font modifiers are:

» **normal**
» **light**
» **medium** (font weight between **normal** and **demibold**)
» **demibold** (font weight between **medium** and **bold**)
» **bold**
» **black**
» **italic**
» **normal** (default)
» **oblique** (if necessary is mapped on **italic**)
» **roman** (is ignored)

Note:

It should be mentioned that it depends on the selected character set whether and to what extent the selected modifiers are applied.

2.5.4 Font Definition for Microsoft Windows

There are the following definition methods for a font:

» By specifying a string according to the following pattern:

```
fontSpec ::= "<size>.<fontName>" ;
```

In doing so, you can choose a font with a defined name and size (in points).

» by using a standard system font by giving a string according to the following pattern:

```
fontSpec ::= "<systemFont>" ;
```

The following system fonts are possible:

- » SYSTEM_FONT
- » ANSI_FIXED_FONT
- » ANSI_VAR_FONT
- » DEVICE_DEFAULT_FONT

Remark

You can find a list of existing and usable system fonts along with their sizes in *"Control Panel"* → *"Fonts"*.

Remark

If the specified font is not available, the system font will be chosen instead.

Warning

All fonts available in the system can be used, since there is no limitation to the code page when choosing fonts.

This may lead to choosing fonts with a wrong code page so that the wrong characters are displayed. The IDM on MICROSOFT WINDOWS has no possibility to make a code page conversion since only the WINANSI code page is defined.

See also

Chapter "Note on the use of symbol fonts under Microsoft Windows"

2.5.5 Font Definition for Qt

Fonts can be defined in the following ways:

Qt Style with Comma-separated Parameter List

```
Family, PointSize, PixelSize, QFont::StyleHint, QFont::Weight, QFont::Style,
Flag underline, Flag strikeout, Flag pixedPitch, Flag rawMode
```

```
"Helvetica, 12, -1, 5, 75, 1, 0, 0, 0, 0"
```

Not all parameters have to be specified, `"Helvetica, 12"` is sufficient, for example.

Font with Independent Display Attributes (Size and Style)

```
"Helvetica", 12, bold
```

The styles *bold*, *italic* and *oblique* are supported.

Qt style specifications can be combined with independent display attributes:

```
"Helvetica, 12", bold
```

X Logical Font Description (XLFD)

```
"-adobe-helvetica-medium-r-normal--12-*-*-*-p-*-iso8859-1"
"*adobe-helvetica-bold-r-normal-*-100-*-iso8859-1"
```

Notes

- QT has problems processing wildcards in XLFD's.
- If QT can't match a font exactly, it takes the font family that is most likely to match. Inevitably, substitution of values occurs.
- Font aliases (file **fonts.alias** in the font directory of X11) are not supported by Qt. Therefore, for example, font *fixed* cannot be used.

Attention

- Since QT5, supporting functionality for processing XLFD fonts has been depricated.
- Using XLFD fonts is not recommended.

2.5.6 Dynamically changeable attributes

For fonts, the following attributes can be set and queried dynamically at runtime:

Table 3: *Modifiable attributes of the font resource*

Attribute	Data Type	Index Range	Description
.face	*enum face_ default*, *face_italic*, *face_ oblique*, *face_ roman*,	–	font face/ slant

Attribute	Data Type	Index Range	Description
.height[enum]	*integer*	*scale_factor*, *scale_offset*	y-scaling and offset
.name	*string*	–	font name
.refstring	*string*	–	reference string for calculating the grid width
.propscale	*boolean*	–	raster control
.size	*integer*	–	font size
.style	*enum* *face_default*, *face_light*, *face_normal*, *face_medium*, *face_demibold*, *face_bold*, *face_black* *face_italic*, *face_oblique* *face_roman*,	–	font style

Attribute	Data Type	Index Range	Description
.weight	*enum face_default*, *face_light*, *face_normal*, *face_medium*, *face_demibold*, *face_bold*, *face_black*	–	font weight
.width[enum]	*integer*	*scale_factor*, *scale_offset*	x-scaling and offset

2.6 format

You can define format resources in order to format strings in editable texts and tablefields.

A format resource connects a string as a format description with an optional format function which interprets the string.

Definition

```
{ export | reexport } format <Identifier> <formatSpec> | <variantDef>

formatSpec ::=
  <format string> [ <format function> ] ;

variantDef ::=
{
           0 : <formatSpec>
  [ <number> : <formatSpec> ]
}
```

If no format function is given, the format string is interpreted internally by the Dialog Manager. If a format function is given, it has to be defined as `formatfunc` (please see chapter “Format Function” in manual “Rule Language”).

Variants can be used, for example, to make adaptations to different languages.

The following format descriptions are permitted, if the Dialog Manager has to interpret the format string internally:

Empty string

If the format is an empty string, the input strings will not be formatted.

Input pattern

The following characters are available within the format string:

A	alphabetic characters
C	numbers and uppercase letters
H	hexadecimal digits
N	digits
U	alphabetic characters, only uppercase letters
X	any input permitted
9	digits (like N)

/	formatting character
,	formatting character
.	formatting character
-	formatting character
:	formatting character
Space (BLANK)	formatting character

The formatting characters are inserted into the display string for display and input. They are skipped during input and thus cannot be overwritten. After the input, they are not adopted in the contents string and are not available there.

Example

Time input:

```
.format       "NN:NN:NN";
```

Input of *"120300"*

Hidden formatting

If the format string starts with an "S", the contents string is displayed and edited invisibly.

The character following "S" is interpreted as a special sign for hiding. If no further character is following "S", "*" is chosen as hiding sign. Then, the following format strings are possible:

- empty string
- input pattern
- numeric format

With hidden formatting, you can e.g. make a password input.

Example

```
.format     "SxAAA";
```

allows the input of three characters which, however, are all displayed as "x".

Numeric format

allows the formatting of numbers. The format string has to be specified according to the following pattern (element in square brackets [ ] is optional, i.e. it may be left out):

```
% [<|>] [+|-] [ [0] 0] length ['_ [sepch] sepcount ] [ ( . | , ) [0] trail ]
[u|s] (b|d|f|h|o|x|X) [<|>] [+|-] [/(c[0]|p|t|z)+]
```

Meaning of the Various Elements

Formatzeichen	Zeichen	Bedeutung
%		Indicating initial character for numeric formats.
[<|>]		Alignment with left margin:
	no entry	Padding with blanks before sign.
	>	Padding with blanks between sign and digits.
	<	No padding, left-justified.
[+|-]		Leading sign:
	no entry	Display of negative sign only, if no trailing sign is defined.
	-	Display of negative sign only.
	+	Display of positive and negative sign.
[[0]0]		Display of leading zeros:
	no entry	No leading zeros are displayed.
	0	A single leading zero is displayed, if the integral part is zero.
	00	Displays zeros for all non-occupied digits of the integral part.
length		Overall number of digits (not counting special characters).
[' [sepch] sep-count]		At each sepcount-position of the integral part a sepch-character or a “” if no sepch is specified is displayed (e.g. as thousands marker in digit grouping). If sepcount is 0 then 3 will be used by default.
[(.|,) [0] trail]		Description of the fractional part: Defines either a point or a comma as decimal mark.
		trail defines the number of decimal places. If the optional zero is used, non-occupied places will be filled with zeros.
[u|s]		Input of negative numbers:
	u	“unsigned”, no negative numbers permitted
	s	“signed”, negative numbers are permitted
(b|d|f|h|o|x|X)		Formatting signs indicating the number system:

Formatzeichen	Zeichen	Bedeutung
	b	Binary numbers.
	d	Decimal numbers.
	f	Floating point values with decimal mark in the input string.
	h	Same as x.
	o	Octal numbers.
	x	Hexadecimal numbers with lowercase letters.
	X	Hexadecimal numbers with uppercase letters.
		Except when using the formatting sign “f”, the content string may contain no decimal mark. The decimal mark is only inserted into the display string for display and input.
[<|>]		Alignment with right margin:
	no entry	Padding with blanks after sign.
	>	Padding with blanks between sign and digits.
	<	No padding, right-justified.
[+|-]		Trailing sign:
	no entry	No display of sign.
	-	Display of negative sign only.
	+	Display of positive and negative sign.
[/(c[0]|p|t|z)+]		Format modifier:
		The format behavior can be determined by specifying one or more modifiers. Default behavior: During input, the digits are shifted over the decimal mark. Empty strings are allowed as input and are different from zero. The decimal mark is always displayed, even if the content string is empty.

Formatzeichen	Zeichen	Bedeutung
	c	Digit deleting mode: Deleting the very last digit in a string leads to an empty string and not to a string with the value “0”. This modifier has no impact when it is combined with the “z” format modifier (zero-mode).
	c0	0-deletion-mode: When a string represents the numerical value “0” after an entry or a deletion, then the string will be replaced with an empty string. In 0-deletion-mode a content string can change to an empty string when only one digit not equal to “0” exists and this digit is deleted. Example: “7000.00” will be changed to an empty string after deleting the “7” by using the `Del` or `Backspace` key. This modifier has no impact when it is combined with the “z” format modifier (zero-mode).
	p	Point-mode: if the content string is empty, no decimal mark is displayed.
	t	Technical number input: the places before and after the decimal point are edited separately.
	z	Zero-mode: empty strings are converted to zeros, empty strings cannot be input, depending on the format, zeros may be displayed as empty string.

Remarks

- Leading and trailing zeros are inserted into the display string only. They will not be taken into the content string, not even after the entry has been finished and therefore are not present in it.
- Given a one-digit content string, this digit may not be situated rightmost within the formatted string. Rather it can be the first digit left of the comma, e.g. at technical number input.
- In an entry field characters that are inserted into the display string by a format cannot be deleted using the `Backspace` or `Del` keys.
  A sign inserted by a format can be altered using the `+` and `-` keys. In doing so `+` switches to a positive sign or retains it. Entering `-` toggles between positive and negative sign (just like multiplying by *-1*).

Regular Expression

Availability

Since IDM version A.06.02.g

A matching expression of the form "/ … /" is accepted in order to allow the content or input to be validated.

If the content does not match the pattern, an empty string is displayed or the input is prevented.

Examples

» `format FmtKommaZahl "/^(\\d+(\\.\\d*)?)?$/";`
Permitted strings e.g. *""*, *"1"*, *"07654.321"*

» `format FmtHexZahl "/^[abcdefABCDEF0123456789]*$/";`
Permitted strings e.g. *""*, *"AF4513c"*, *"a"*, *"7EF"*

» `format FmtSpecial "/^([[:alpha:]]*\n\\d*\n[01]*)?$/";`
Permitted strings e.g. *""*, *"City\n999\n01"*, *"Milwaukee\n53288\n01101110"*

See also

Built-in function regex and chapter “PCRE Library for Support of Regular Expressions”.

2.7 text

To support multilingual dialogs the DM provides the resource ***text***. The dialog is initially developed in one language.

The resource ***text*** is introduced during the translation into a new language. A string from the original language is defined together with its translations in this process.

Definition

```
{ export | reexport } text { <Identifier> }  "<original string>"
  <variantDef> | ;

variantDef ::=
{
          1 : "<translated string>" | nlscat(<message number>) ;
  [ <number> : "<translated string>" ; ]
}
```

The identifier is optional, i.e. it does not have to be indicated necessarily. By means of the identifier, the text can be referenced in the dialog.

The first translation can be either a <translated string> or an entry from a message or text catalog nls-cat(<message number>), further translations are defined as <translated string>.

The access of the Dialog Manager to the text is as follows:

The function which was installed with DM_InstallNlsHandler() is called. Then the text is stored internally. The access applies only to texts actually used.

The texts have to be defined in ISO 8859/1 standard, which is suitable for almost all European languages, so that as many different characters as possible can be represented. Then the DM converts this representation into the representation of the corresponding window system, so that the characters are correctly represented.

The characters can be input according to the following pattern:

All 7-bit characters are directly entered with the relevant editor, i.e. all ASCII characters can be directly entered. This kind of character input guarantees that all editors can correctly represent these characters, because many editors can only represent 7-bit characters.

All 8-bit characters have to be entered in octal notation, i.e. characters that are larger than 127 have to be entered as octal number with a leading backslash "\".

	xx0	xx1	xx2	xx3	xx4	xx5	xx6	xx7
00								
01x								
02x								
03x								
04x								
05x								
06x								
07x								
10x								
11x								
12x								
13x								
14x								
15x								
16x								
17x								
20x								
21x								
22x								
23x								
24x		¡	¢	£	¤	¥	¦	§
25x	¨	©	ª	«	¬	-	®	¯
26x	°	±	²	³	´	µ	¶	·
27x	¸	¹	º	»	¼	½	¾	¿
30x	À	Á	Â	Ã	Ä	Å	Æ	Ç
31x	È	É	Ê	Ë	Ì	Í	Î	Ï
32x	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×
33x	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
34x	à	á	â	ã	ä	å	æ	ç
35x	è	é	ê	ë	ì	í	î	ï
36x	ð	ñ	ò	ó	ô	õ	ö	÷
37x	ø	ù	ú	û	ü	ý	þ	ÿ

Figure 1: *Table for Octal Values*

To describe a certain character, e.g. "A", you first have to specify the corresponding value on the y-axis:

-> 10x

"x" then has to be substituted by the corresponding value on the y-axis:

-> xx1 (xx stands for the above results of 10)

It results in:

=> \101

Further Examples

```
ö     36x     xx6     =>     \366

#     04x     xx3     =>     \043
```

Example

```
text "Hello world"
{
  1:  "Hallo Welt";
  2:  "Allo le monde";
  3:  "Buon giorno il mondo";
  4:  "Holá mundo";
}
```

By selecting one of the languages 1–4 with the option **-IDMlanguage**, the string *"Hello World"* is replaced with the corresponding translation throughout the dialog. You can get the defined text either by inputting the original text or by inputting the identifier.

Example

```
text WindowText "Window"
{
  1:  "Fenster";
  }
Window.title := WindowText;
```

is the same as

```
text "Window"
{
  1: :"Fenster";
}
Window.title := "Window";
```

Remark

With the attribute .text you can get the string in the current language variant.

2.7.1 Note on the use of symbol fonts under Microsoft Windows

When using fonts that contain symbols (e.g. Wingdings) the following should be observed:

1. The name must correspond to the actual name of the font (no alias). If Windows executes a font exchange, IDM recognizes this and forces the replacement to a font that supports the WIN ANSI character set. This is necessary to prevent the unintentional selection of an illegible font.
2. A font that contains symbols is allocated to the Windows SYMBOL character set. This is mapped on the Unicode rage \uF000–\uF0FF (user-defined range in Unicode). In order to display the

character 0xFE, \uF0FE must be defined.

Remark

This is generally not necessary on a western European Windows system as the system code page CP1252 is almost identical to the lower Unicode range and contains almost all characters. This is why this does not lead to a conversion and the desire character is shown – quasi by chance. This however is not the case on other Windows language versions.

2.8 tile (pattern)

Graphics, pictures and patterns used within the IDM must be declared as resources. The keyword for this is **tile**, followed by an identifier and the tile definition. The tile definition is either included directly in the IDM file or in a separate graphics file.

Definition

```
{ export | reexport } tile <Identifier> <tileSpec> | <variantDef>

tileSpec ::= <tileBitmap> | "<file path>" | "<graphics resource>" { scale |
               noscale | numscale | propscale | } ;

tileBitmap ::= <x>, <y>,
               "<pattern line>"
           [ , "<pattern line>" ]

variantDef ::=
{
           0 :  <tileSpec>
  [ <number> :  <tileSpec> ]
}
```

For the definition of variants it is recommended to always define a variant with the number *0*, since this is the default variant if no variant, a non-existent variant, or an invalid variant is set. The other numbers may be arbitrary natural numbers.

The ***tile*** variant to be used can be set with the command line option **-IDMtile** when starting the application. During runtime of the application it can be queried and set with the .tile attribute of the setup object.

2.8.1 Internal pattern (Bitmap)

<*x*> and <*y*> represent the number of pixels in horizontal and vertical direction. Width (<*x*>) and height (<*y*>) may be chosen arbitrarily.

<*pattern line*> denotes a string of <*x*> characters in which each character represents one pixel of the tile. Only rectangular tiles are possible, therefore exactly <*y*> pattern lines with <*x*> characters each have to be defined.

The pattern lines consist of the characters period (“.”) space (“ ”) and hash (“#”), which have the following meaning:

“.”

“ ” (Leerzeichen)

Adopts the background color, i.e. the pattern will be transparent at this point.

If the ***tile*** resource is used in the *.picture* or *.picture[enum]* attributes of an ***image*** object whose .imagebgc attribute is set, a pixel in the color *.imagebgc* is displayed here.

“#”

At this point, a pixel in the foreground color will be displayed.

When using the ***tile*** resource in the *.picture* or *.picture[enum]* attributes of an ***image*** object, this will be the color set in the .imagefgc attribute of the ***image*** object.

Example for the Definition of an Internal Pattern

```
tile TiGray 16, 16,
  "# # # # # # # # ",
  " # # # # # # # #",
  "# # # # # # # # ",
  " # # # # # # # #",
  "# # # # # # # # ",
  " # # # # # # # #",
  "# # # # # # # # ",
  " # # # # # # # #",
  "# # # # # # # # ",
  " # # # # # # # #",
  "# # # # # # # # ",
  "# # # # # # # # ",
  " # # # # # # # #",
  "# # # # # # # # ";
```

2.8.2 External pattern

With the definition using a file name (including path), the IDM assumes that the definition of the tile is in a separate graphics file. The IDM loads the corresponding file and displays its contents using the system functions of the respective toolkit. Therefore it depends on the platform which graphics formats can be displayed.

The following table shows which graphics formats are supported on each platform:

Table 4: *Graphics formats supported by the tile resource*

Graphics Format	File Extension	Microsoft Windows	Motif	Qt
Graphics Interchange Format	.gif	yes	yes	yes
Device Independent Bitmap	.bmp	yes	no	yes
X PixMap	.xpm	no	since Motif 2.1	no
JPEG File Interchange Format	.jpg	yes	since Motif 2.3	yes
Portable Network Graphics	.png	yes	since Motif 2.3	yes

Graphics Format	File Extension	MICROSOFT WINDOWS	MOTIF	QT
Enhanced Metafile Windows Metafile	.emf .wmf	yes	no	no
Aldus Placeable Metafile	.apm	yes	no	no
Windows Icon Resource File	.ico	yes	no	no
Windows Icon Resource Windows Bitmap Resource	–	yes	no	no
Scalable Vector Graphics	.svg	yes	no	yes

The definition using a graphics resource is only supported by the IDM FOR WINDOWS. The identifiers “IDM_AppIcon” and “IDM_DefIcon” of the resources defined in the IDM libraries may be used here (see also attribute .icon in the “Attribute Reference”).

Additionally, on MICROSOFT WINDOWS, the following identifiers for predefined bitmap resources of the system can be specified:

BTNCORNERS	BTSIZE	CHECK	CHECKBOXES	CLOSE
COMBO	DNARROW	DNARROWD	DNARROWI	LFARROW
LFARROWD	LFARROWI	MNARROW	OLD_CLOSE	OLD_DNARROW
OLD_LFARROW	OLD_REDUCE	OLD_RESTORE	OLD_RGARROW	OLD_UPARROW
OLD_ZOOM	REDUCE	REDUCED	RESTORE	RESTORED
RGARROW	RGARROWD	RGARROWI	SIZE	UPARROW
UPARROWD	UPARROWI	ZOOM	ZOOMD	

Note

The images of the ***tile*** resource are now automatically enlarged according to the configured scaling factor. It is assumed hereby that the images were designed for a DPI value of *96*. If the images of the application are designed for a higher resolution, then this can be set in the setup object with the attribute .tiledpi.

When specifying an external file, it is recommended not to define the absolute path but rather to use an environment variable, e.g. *"IDM_IMAGEPATH:Check.gif"*.

Other graphics formats can be processed using a self-implemented graphics handler (GFX handler; see function DM_PictureHandler in manual “C Interface - Functions”).

2.8.3 Scaling

By specifying a **scalestyle** in the tile definition, the tile or picture will be resized to fit the available space according to specific criteria.

Best practice: If possible, patterns should already be available in the correct or used size, as any scaling is associated with a loss of sharpness or richness of detail.

"scale | noscale | numscale | propscale"

Determines the type of scaling of the pattern or image.

Definition on Tile	Dynamic setting	Meaning
noscale	scalestyle_ none	The pattern or image is not scaled. setup.tiledpi has no impact.
scale	scalestyle_ any	The height and width of the pattern or image are fully enlarged to fit the available area.
propscale	scalestyle_ prop	Height and width of the pattern or image are enlarged to the available area, while height and width proportions of the pattern or image are maintained in any case. I.e. free spaces can be created above and below or left and right.
numscale	scalestyle_ num	The scaling of the pattern or image is done by a numerical scaling divider. The scaling is done in quarter steps, i.e. 1.25-fold, 1.5-fold, 1.75-fold, 2-fold, 2.25-fold, 2.5-fold, and so on. A downscaling is done down to a maximum of 0.25-fold.
dpi	scalestyle_ dpi	The pattern or image is always scaled according to the set screen scaling.
Not provided	scalestyle_ auto	The pattern or image is scaled according to the set screen scaling. A scaling compatible to the previous version takes place. **Default value**

See also

Attribute .scalestyle in manual "Attribute Reference"

2.8.4 Dynamically Changeable Attributess

The modifiable attributes can be found in the following table. It should be noted that *.name* on the one hand and *.width*, *.height* and *.pattern* on the other hand are mutually exclusive, so that "get" leads to a *"can't get value"* error message.

For variant ***tile*** resources, the attributes of the current ***tile*** variant are accessed; the current variant can be queried and set using the .tile attribute of the setup object.

Table 5: *Modifiable attributes of the tile resource*

Attribute	Data Type	Index Range	Description
.name	*string*	–	File path of an image file.
.width	*integer*	–	Width of the pattern in pixels.
.height	*integer*	–	Height of the pattern in pixels.
.pattern	*string*	–	Complete pattern as string.
.pattern[integer]	*string*	*1 … .height*	Single lines of the pattern. The length of each string must be equal to *.width*.
.scale	*boolean*	–	Resizing to the available space. Default value ***false*** (no resizing).
.scalestyle	*enum*	–	Controls the display and scaling of a tile. Default value ***scalestyle_auto***

2.8.5 SVG Support

On WINDOWS and QT, the tile resource now also supports vector images in SVG format.

2.8.5.1 QT

The IDM for QT was already able to display SVG images, but only via a pixel-based proxy image that was scaled accordingly. The .svg file extension is now recognized as an SVG image format and vector-based drawing is used when displaying it in the various objects. This ensures the best possible reproduction of vector images for applications that are displayed on HiDPI screens.
For performance reasons, the pixel-based proxy image is used when drawing background tiles of grouping objects in the *tilestyle_tiled* style.

> Installation note: The QT package **qt5-qtsvg** must already be installed in order to make the corresponding SVG support library available and to ensure that an IDM application can be started.

2.8.5.2 WINDOWS

MICROSOFT DIRECT2D is used to ensure the support of graphic formats. The SVG support of MICROSOFT DIRECT2D depends on the MICROSOFT WINDOWS version. As of the WINDOWS 10 CREATORS UPDATE, support is provided in a limited form.

Noteson the usage of the svg Format as ICON

The SVG format cannot be perfectly converted to an icon, so it is not suitable for displaying as a *.picture* of the treeview or notepage object. Nor is it suitable as an *.icon* of the window object. Unsightly edges may appear at the transition between the transparent and drawn areas.

Possible changes to the previous version:
As this graphics package uses different scaling methods than the WINDOWS GDI, patterns (images) that need to be scaled for display may look different than in the previous versions of the IDM. This applies in particular to patterns that are used as window icons or as images in the **treeview** and **notepage** objects, as MICROSOFT WINDOWS requires the icon data type here.
Best practice: Patterns (images) are already stored in the required size.

The SVG support with MICROSOFT DIRECT2D

Support depends on the MICROSOFT WINDOWS version. As of the WINDOWS 10 CREATORS UPDATE, the following SVG elements and attributes are supported:

Element	Supported attributes
circle	id, style, transform, cx, cy, r
clipPath	id, style, transform, clipPathUnits
defs	id, style, transform
desc	id
ellipse	id, style, transform, cx, cy, rx, ry
g	id, style, transform
image	id, style, transform, x, y, width, height, preserveAspectRatio, xlink:href
line	id, style, transform, x1, y1, x2, y2
linearGradient	id, style, x1, y1, x2, y2, gradientUnits, gradientTransform, spreadMethod, xlink:href
path	id, style, transform, d
polygon	id, style, transform, points
polyline	id, style, transform, points
radialGradient	id, style, cx, cy, r, fx, fy, gradientUnits, gradientTransform, spreadMethod, xlink:href
rect	id, style, transform, x, y, width, height, rx, ry
stop	id, style, offset
svg	id, style, x, y, width, height, viewBox, preserveAspectRatio
title	id
use	id, style, transform, x, y, width, height, xlink:href

SVG presentation attributes

- clip-path
- clip-rule
- color
- display
- fill
- fill-opacity
- fill-rule
- opacity
- overflow
- stop-color
- stop-opacity
- stroke
- stroke-dasharray
- stroke-dashoffset
- stroke-linecap
- stroke-linejoin
- stroke-miterlimit
- stroke-opacity
- stroke-width
- visibility

Supported length units

The user-space length values and percentage length values as well as absolute unit identifiers: px, pt, pc, cm, mm and in.

Image sources

The image element is only supported if its xlink:href attribute is set to a base64-encoded Image.

See also: **https://learn.microsoft.com/en-us/windows/win32/direct2d/svg-support**

3 Programming Resources

3.1 message

With this resource objects can be defined which can be used for the definition of external events. These events can be used in the Rule Language for instance with the built-in function sendevent().

The resource ***message*** is also used with the OLE interface of ISA Dialog Manager; please see chapter “The message resource” of manual “OLE Interface” (definition and usage with the OLE interface).

Definition

```
{ export | reexport } message  <Identifier> { (<messageSpec>) };
```

Note

The argument *messageSpec* is not evaluated in the Rule Language when the message resource is utilized for the definition of an external event.

More information on the argument *messageSpec* can be found in chapter “The message Resource” of manual “OLE Interface”.

Example

```
message EvInformation;

on dialog start
{
  ...
  sendevent(WnMain, EvInformation, “Dialogstart”);
}

window WnMain
{
  ...
  on extevent EvInformation (string Info)
  { ... }
}
```

3.2 source

This resource is used to define the behavior of an object being the source of a Drag&Drop operation. In contrast to other programming resources variants can be defined here.

The assignment to an object is carried out via the attribute *.source*. If this attribute is set you can choose it as a source for a Drag&Drop operation.

Note

Drag&Drop is not supported by the IDM FOR MOTIF.

Definition

```
{ export | reexport } source <Identifier>
{
            0 : .action <action_enum> [ , <action_enum> ] ;
                .type   <type_enum>   [ , <type_enum> ] ;
  { <variant> : .action <action_enum> [ , <action_enum> ] ;
                .type   <type_enum>   [ , <type_enum> ] ; }
}
```

As actions (<action_enum>) the values *action_cut* and *action_copy* are possible.

The following values for the type (<type_enum>) are available:

type_text
: The text attribute of the object is transferred.

type_object
: A DM-ID is transferred.

Example

The following example illustrates the definition of the source resource “Src” with two variants

The first variant permits the copying and cutting of texts. The second variant permits to copy the object-ID.

The resource is then assigned to the ***listbox*** “Lb”.

```
source Src
{
  0: .action action_cut, action_copy;
     .type   type_text;
  1: .action action_copy;
     .type   type_object;
}

listbox Lb
{
  .source Src;
```

```
  ...
}
```

3.3 target

This resource is used to define the behavior of an object being the target of a Drag&Drop operation. In contrast to other programming resources variants can be defined here.

The assignment to an object is carried out via the attribute *.target*. If this attribute is set you can move another object onto this object by using Drag&Drop.

The structure is very similar to the resource ***source***.

Note

Drag&Drop is not supported by the IDM FOR MOTIF.

Definition

```
{ export | reexport } target <Identifier>
{
            0 : .action <action_enum> [ , <action_enum> ] ;
                .type   <type_enum>   [ , <type_enum> ] ;
  { <variant> : .action <action_enum> [ , <action_enum> ] ;
                .type   <type_enum>   [ , <type_enum> ] ; }
}
```

As actions (<action_enum>), *action_copy*, *action_cut* and *action_paste* are possible, with *action_paste = action_cut, action_copy*.

In contrast to ***source*** the order of types and actions is analyzed. With help of the priorities list of types the transfer format is defined. With the priorities list of actions the action is defined.

Example

```
target Tar
{
  0: .action action_paste;
     .type   type_text;
}

listbox Lb
{
  .target Tar;
  ...
}
```

Index

E

F

G

H

I

K

L

M

N

O

U

V

W

X