INTENS / Engineering Manager

System Administration Manual

February 2026
Version 6.0.7

SEMAFOR Informatik & Energie AG
Sperrstrasse 104b
4057 Basel
Switzerland

Tel: +41 61 690 98 88
Fax: +41 61 690 98 80
Email: info@semafor.ch
WWW: http://www.semafor.ch

The information in this document is subject to change without notice and should not be construed as a commitment by SEMAFOR AG. SEMAFOR AG assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Copyright ©1994-2025 by SEMAFOR AG

All Rights reserved. No part of this document covered by copyright hereon may be reproduced in any form or by any means without written permission from the copyright owner.

INTENS is a registered trademark of Semafor Informatik & Energie AG
All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

This document was prepared using LaTeX.

Contents

1 Release Notes

This document focuses on the **INTENS** syntax, so these release notes mainly lists syntax changes. Most **INTENS** releases also fix bugs, but these are rarely listed here. This explains why there are releases in these release notes without listed changes.

1.1 Release Notes 6

1.1.1 Release Notes 6.0

The following features have been changed or added in Version 6.0:

(see section set_statement page 200)

6.0.8 (2026-??-??)

- Added function statement **SET**(**TITLE**, fieldgroup, “new title”) to change the **TITLE** of a **FIELDGROUP** at runtime. See set_statement on page 200.
- Added function statement **SET**(**EXPAND**, fieldgroup, **TRUE** or **FALSE**) to expand or collapse an **ACCORDION FIELDGROUP**. See set_statement on page 200.

6.0.7 (2026-02-10)

- TBD

6.0.6 (2026-01-29)

- Added **FIELDGROUP** option **ACCORDION**. See ui_fieldgroup_option on page 78.

6.0.5 (2026-01-20)

- TBD

6.0.4 (2025-12-04)

- TBD

6.0.3 (2025-11-21)

- Allow **HIDDEN**=**INTEGER**in **FOLDER** tab options. See ui_folder_button_option_list on page 137.

6.0.2 (2025-11-07)

6.0.1 (2025-10-31)

- New datapool item option **PROGRESS**. See data_item_options on page 40.

6.0.0 (2025-10-24)

1.2 Release Notes 5

1.2.1 Release Notes 5.3

The following features have been changed or added in Version 5.3:

5.3.4 (2025-10-16)

5.3.3 (2025-09-24)

5.3.2 (2025-08-20)

- Removed `LANGUAGE`.
- Added `UNIT_MAGAGER`. See Unit Manager on page 242.

5.3.1 (2025-07-16)

- Removed `DB_MANAGER`, Python func call.

5.3.0 (2025-06-11)

- Added protocol buffer support for webtens operation.
- LOG4CPP replaced by LOG4CPLUS.
- USERGROUPS removed.

1.2.2 Release Notes 5.2

The following features have been changed or added in Version 5.2:

5.2.16 (2025-05-21)

5.2.15 (2025-03-06)

- added function statement **CLASS**. See data_statement on page 195.

5.2.14 (2025-01-20)

1. **FOLDER** option **BUTTON** = **TRUE** added. See ui_folder_option_list on page 136.
2. Possibility to **MAP** and **UNMAP** a line or column of a **TABLE** added. See job_map_element on page 204.
3. **TABLE** option **AUTO_WIDTH** added. See ui_tbl_option on page 85.
4. **FIELDGROUP** field and **FIELDGROUP** option **CLASS** added. See ui_field_additional_attributes on page 73 ui_field_options on page 75 and ui_fieldgroup_option on page 78.
5. Added possibility to provide a title for a **MESSAGEBOX**. See messagebox_statement on page 210.
6. Added possibility to hide MenuButtons (PulldownMenu). See job_unmap_element on page 204.
7. Added possibility to check **EDITABLE** state of a **FIELDGROUP**. See job_editable_action on page 231.
8. Option **FILE** for **STREAM** added. See st_option_list on page 53.

5.2.13 (2024-10-03)

1. Added possibility to check **EDITABLE** state of a variable. See job_editable_action on page 231.
2. **LIST FUNCTION** is called when a **LIST** is sorted differently, with **REASON_SORT**. See List on page 92 and reason_expression on page 235.
3. Added function statement **GET_SORT_CRITERIA**. See gui_more_statement on page 205.
4. **FIELDGROUP** option **INDEX** added. See ui_fieldgroup_option on page 78.

5.2.12 (2024-07-17)

1. Option **OMIT_ACTIVATE** for **MAP** folder tab added. See job_map_element on page 204.

5.2.11 (2024-03-14)

1. Function Expression **GETTEXT** added (see function_expression on page 228).
2. Possibility added to **ENABLE** and **DISABLE** a **DIAGRAM** (see gui_statement on page 203 and Navigator Diagram on page 131).
3. Set **STYLESHEET** of **FORM** now possible (see gui_more_statement on page 205).

5.2.10 (2023-12-01)

1. WebApi: Alternate Background Color in Table and List GuiElements.
2. PASTE webtens improved (performance).

5.2.9 (2023-11-10)

1. PASTE works with webtens.
2. Error label in password dialog spans all columns.
3. Improve jwt login.

5.2.8 (2023-10-26)

1. Fix bug in **TABLE** > right click menu > Clear all.

5.2.7 (2023-10-04)

1. Dependencies added between input (**REQUEST**) and output (**RESPONSE**) **STREAM**s of **MESSAGE_QUEUE REQUEST**s.
   STREAMs can be excluded from dependencies using the option **NO_DEPENDENCIES** (see job_message_queue_option on page 214 or job_plugin_option on page 214).
2. Commandline option defaultMessageQueueDependencies added (default: true) See Commandline options on page 247.

5.2.6 (2023-09-13)

1. Fix bug in **TABLE** > right click menu > Clear selected rows/columns.
2. Added Clear to **TABLE** > bottom right > right click menu. See bottom right menu on page 91.
3. Added function statement **BEEP**. See system_statement on page 210.

5.2.5 (2023-07-26)

1. Fix table layout problem.

5.2.4 (2023-04-04)

5.2.3 (2023-03-21)

5.2.2 (2023-03-08)

1. Added **REST_JWT_LOGON**. See restService_action on page 220.

5.2.1 (2023-01-19)

5.2.0 (2022-11-24)

1. **FOLDER** button option **HIDDEN** added. See ui_folder_button on page 137.

1.2.3 Release Notes 5.1

The following features have been changed or added in Version 5.1:

5.1.19 (2022-10-27)

1. added **REASON_FUNCTION**, **REASON_TASK**and **REASON_GUI_UPDATE**. See reason_expression on page 235.

5.1.18 (2022-09-30)

5.1.17 (2022-09-26)

5.1.16 (2022-09-22)

5.1.15 (2022-09-08)

1. **INTENS** commandline option --persistfileREST added. See Commandline options on page 247.
2. Added HELPFILE functionality lost since **INTENS** version 4. See HELPFILE on page 31.
3. Added statement **ERASE** to clear values and attributes. See data_statement on page 195.
4. Added statement to copy a ui_form_element_identifier or a Form to the clipboard. See copy_paste_statement on page 218.
5. Added variable **PLOT2D_SYMBOLSIZE** to overwrite the symbol size of all **PLOT2D** curves. See Global Symbolsize on page 109.

5.1.14 (2022-07-07)

5.1.13 (2022-07-01)

5.1.12 (2022-05-30)

1. Added options **HIDDEN** and **TRANSIENT** for **JSON STREAM**s. See st_json_options on page 59.

5.1.11 (2022-01-31)

1. Constistency Check with Message-Queue Requests. See job_message_queue_option on page 214.
2. Version control in RestService. See Rest Service: Version control on page 222.
3. (bugfix) **NAVIGATOR**: text formatting was ignored

5.1.10 (2021-10-27)

1. (bugfix) **NAVIGATOR** (**DIAGRAM**): **STRUCT MENU** was missing
2. Removed support for obsolete syntax. Now, the following newer syntax is required:
   - () after PRINT, SET_ERROR, ABORT
   - VALID(...) instead of ... .VALID
   - () after UI_UPDATE, MAP, UNMAP, ALLOW, DISALLOW, ENABLE, DISABLE
   - LABEL(...) instead of LABEL . ...
   - UNIT(...) instead of UNIT . ...
   - PIXMAP(...) instead of PIXMAP . ...
   - REPORT(...) instead of REPORT("...")
   - PREVIEW(...) instead of PREVIEW("...")
   - RUN(...) instead of RUN ...
   - require START/STOP(ID_TIMER...) instead of START/STOP(TIMER = ID_TIMER...)
   - require tID_FORM instead of tFORM tID_FORM (except for **MENU**declaration)
   - require tID_FOLDERGROUP instead of tFOLDER tID_FOLDERGROUP
   - require CLEAR(...) instead of CLEAR ...
   - require PACK(...) instead of PACK ...
3. (bugfix) **ROWSPAN** only worked once

5.1.9 (2021-07-29)

1. Added function statement **ASSIGN_CONSISTENCY** to assign a value to a variable and clear all dependent results. See data_statement on page 195.
2. Added option **AUTO_SCROLL** for multiline textfields. See ui_field_additional_attributes on page 73.

5.1.8 (2021-07-16)

1. (bugfix) Automatic database login (REST) using environment variables REST_SERVICE_USERNAME and REST_SERVICE_PASSWORD with JWT authorization.
2. (bugfix) Compare **CYCLE** (Case) in **PLOT2D**: show correct data when involved **INDEX** is set to different values in the cycles.
3. (bugfix) Position of **CYCLE** (Case) dialog improved.
4. (bugfix) **TOGGLE** in **TABLE**: single click is sufficient to toggle the field now.
5. (bugfix) Bug in writing **JSON** data fixed: Arrays with some **INVALID** values were not written correctly.
6. (bugfix) QSS is now applied to all **FORM**s.

7. (bugfix) Entering or changing a date in a **STRING_DATE** input field works again. (Popup calendar always worked).

5.1.7 (2021-05-12)

1. (bugfix) **DATASET_TEXT** works with **INDEXED_SET** now.
2. Added option **INDENT** for **JSON STREAM**s. See st_json_options on page 59.
3. (bugfix) **PLOT2D** option **ASPECT_RATIO** works with data reference (variable) now. See ui_plot2d_xaxis_options on page 112.

5.1.6 (2021-04-23)

1. (bugfix) Write **REAL** numbers with 12 significant digits to **JSON STREAM**s (just as in **XML STREAM**s) to write 3.0793 instead of 3.0792999999999999.
2. (bugfix) Error message in Log Window improved on calling a not implemented function.
3. (bugfix) Negate fixed: After `j = -i;`, j was i and not -i.

5.1.5 (2021-04-08)

1. **INTENS** commandline option `--withoutTextPopupMenu` added to disable the right click menu of multiline text input fields.

5.1.4 (2021-04-06)

5.1.3 (2021-02-09)

5.1.2 (2021-01-26)

1. Added **FUNCTION ON_CYCLE_EVENT** and **REASON**s
   a) **REASON_CYCLE_CLEAR**
   b) **REASON_CYCLE_DELETE**
   c) **REASON_CYCLE_NEW**
   d) **REASON_CYCLE_RENAME**
   e) **REASON_CYCLE_SWITCH**

   See job_function on page 190 and Reason on page 234.
2. Color for **PLOT2D**symbols Cross, XCross, HLine, VLine and Star1. These symbols are only drawn with a thin line. That line used to black always. Now, that line is drawn in the plot curves symbol color.

5.1.1 (2020-12-15)

1. Added **LIST** option **SORT DISABLE**. See ui_list_option_list on page 92.
2. Added **FUNCTION** statement **DATA_SIZE**. See data_statement on page 195.

1.2.4 Release Notes 5.0

The following features have been changed or added in Version 5.0:

5.0.7 (2020-04-28)

1. RestService: Added HTTP **POST** method. See restService_action on page 220.

5.0.3 (2019-11-28)

1. Option **OMIT_TTRAIL** for **MAP** folder tab: syntax changed. See job_map_element on page 204.

5.0.2 (2019-08-13)

1. Vertical **SEPARATOR** in **FIELDGROUP**s. See ui_fieldgroup_line on page 71.
2. **VOID** size relative to **FORM** size. See ui_field_void_size on page 74.

5.0.0 (2019-06-05)

1. Option **OMIT_TTRAIL** for **MAP** folder tab added. See job_map_element on page 204.
2. Options to define container **SCROLLBARS** policy. See ui_form_container_options on page 141.

2 Introduction

2.1 INTENS – an integrated engineering environment

INTENS is an integrated engineering environment with a graphical user interface to combine calculation programs with relational database systems.

Existing batch programs (or MATLAB functions) can be integrated without source code modifications.

Changes and modifications in the engineering processes are fully documented.

INTENS is dynamically configurable and therefore easily adaptable to a variety of engineering processes.

INTENS is currently supported in the following environment:

- the operating systems: Linux, Windows (Qt User Interface)
- Docker, Kubernetes (Web Browser User Interface)

2.2 Purpose of this manual

This manual provides information for configuring INTENS to specific needs. It is written for administrators who have some familiarity with structured programming languages such as C/C++, Python and the Linux operating system.

3 System Architecture

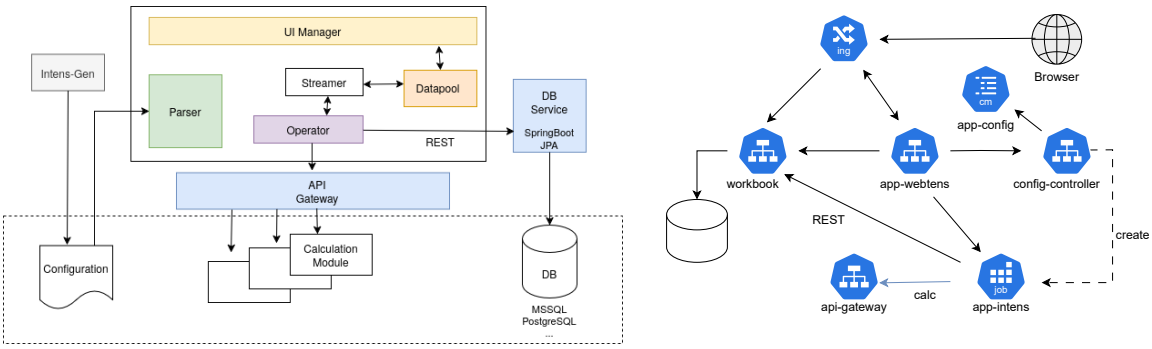


Figure 1: Architecture in Desktop and Cloud Operation Mode

Each INTENS application is created through a simple script which contains the configuration of the applications data and the interfaces of DB Service, calculation programs and user interface. The operation of INTENS applications is possible in a desktop or in a cloud enviroment.

As shown in the above figure, INTENS consists of the following components:

The Parser interpretes the configuration data at program startup and sends the appropriate messages to all involved objects to create the application with all configured interfaces.

The Datapool dynamically manages an arbitrary number of variables with an arbitrary number of calculation variants, called cycles. The only limitation is the size of the machine's virtual memory. The variables have unique names, data types (INTEGER, REAL or STRING) and modification attributes (OPTIONAL, EDITABLE, LOCKABLE) but no static dimension. They can be accessed as lists (vectors) or matrices with dynamic dimension, which will be expanded or reduced according to the requested size.

The UI Manager creates the necessary dialog objects, manages the user input and displays the results to provide a standardized and uniform look-and-feel. Each application has one main window, a configurable number of form windows and several dialog windows like file selection dialogs and message dialogs. The main window contains a title bar, a menu bar, a work area and an action button bar. The values of the data items are displayed in textfields which are part of the form windows. Each form consists of an configurable number of fieldgroups, text and plot windows, which can be arranged horizontally and vertically within one form. Plot graphs can be printed and/or saved as PostScript or HPGL files. Calculation programs can be started and stopped by activating push buttons or menu buttons.

The Streamer manages all input and output formats. A stream consists of a sequence of data items and string constants. Together with the datapool and the operator, the streamer also controls the dependencies between the datapool items. In order to keep the data consistent, all data items that belong to an output stream are set invalid if one of the items of a corresponding input stream is modified.

The Operator communicates with the operating system and the external calculation programs using the pipe mechanism or the MathLink protocol. Several (one to many) calculation programs are combined to a process group that can be started and stopped by the user activating a push button. Data can also be read from and saved to files.

4 Configuration Syntax

4.1 Conventions

The parser reads and interpretes the configuration data file to configure the **INTENS** application at each program startup. This section describes the syntax of the **INTENS** script (configuration) language.

The following conventions are used in the syntax diagrams:

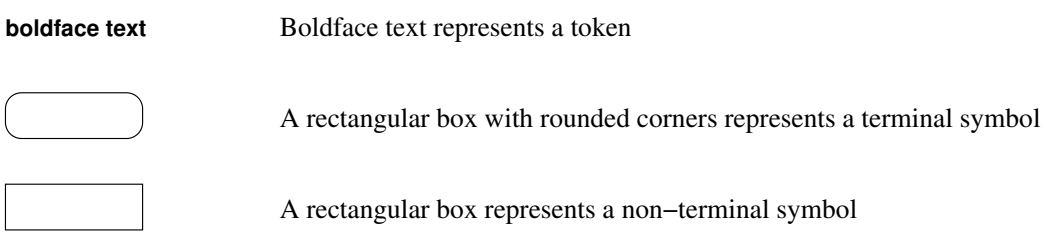


Figure 2: Syntax conventions

- Any text is considered to be case sensitive.
- Characters after `//` are ignored. They may be used as comment marks. `#`-characters can be used for macro definitions in combination with a pre-compiler.
- Floating-point constants contain a decimal point (`3.14`) or an exponent (`1e-2`) or both.
- A string constant is a sequence of zero or more characters surrounded by double quotes. Two or more following string constants can be concatenated with an & operator:

  ```
  "Im Innern der Erde " & "wütet das Nichts."
  ```

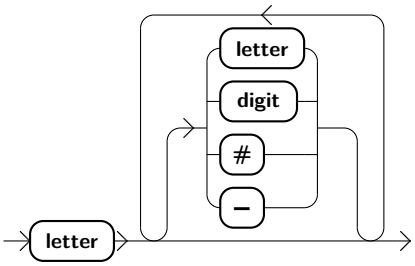
- A string can be composed at parse time (i.E. needed for translation):

  ```
  COMPOSE_STRING("Create new %1.", LABEL(motor))
  ```

4.1.1 Identifier

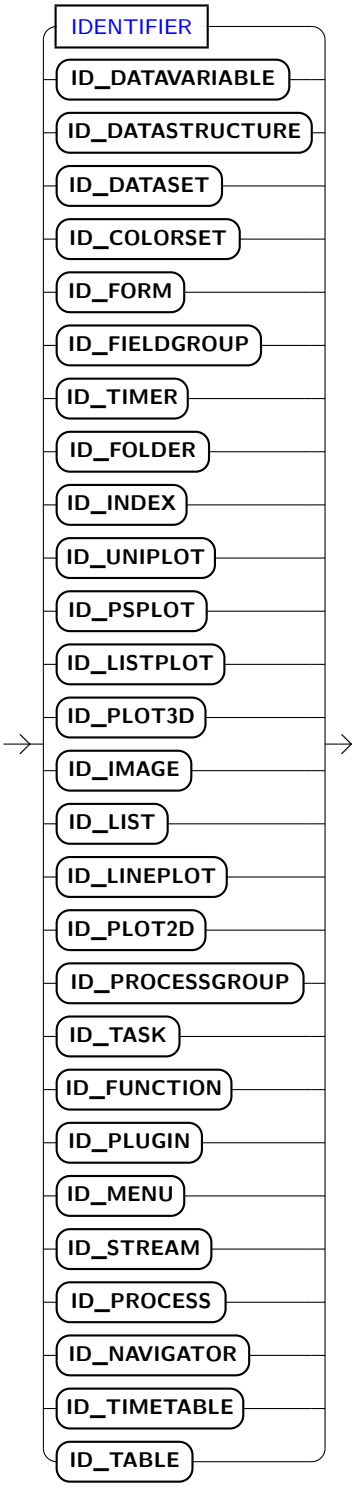
Identifiers are used in almost every section of this manual. Their first character must be a letter:

IDENTIFIER

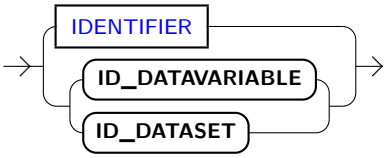


Identifier	Description
letter	alphabetic characters (a..z, A..Z)
digit	digits (0..9)
#	# character (not a wildcard!) may be used as abbreviation of number
_	underline character

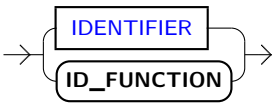
identifier


IDENTIFIER
ID_DATAVARIABLE
ID_DATASTRUCTURE
ID_DATASET
ID_COLORSET
ID_FORM
ID_FIELDGROUP
ID_TIMER
ID_FOLDER
ID_INDEX
ID_UNIPLOT
ID_PSPLOT
ID_LISTPLOT
ID_PLOT3D
ID_IMAGE
ID_LIST
ID_LINEPLOT
ID_PLOT2D
ID_PROCESSGROUP
ID_TASK
ID_FUNCTION
ID_PLUGIN
ID_MENU
ID_STREAM
ID_PROCESS
ID_NAVIGATOR
ID_TIMETABLE
ID_TABLE

data_identifier



job_function_pointer



Example:

```
data_item_identifier
speed
fieldgroup_1
main_form
Superusers
StreamTitle
Phone_#
```

4.1.2 Scale Factors

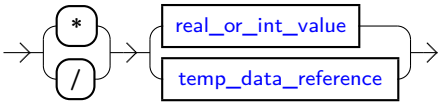
Scale factors are used in conjunction with datapool items to display their value divided or multiplied by a factor.

The default value of a scale factor is NaN (not a number). It is used when a datapool item is declared to be the scale factor, and that item has no value.

You can change the default value of all scale factors to be 1.0 by starting Intens with the commandline option --defaultScaleFactor1. Invalid scale factors then behave as if they were not declared.

NOTE: starting with Rel 5.3.2 a unit manager can be used as an alternative (see section Unit Manager page 242).

scale_factor



scale	Description
`real or int value`	any numeric value or expression as shown in the example below.
`temp data reference`	references a numeric data item declared in the datapool (section Data Reference page 51).

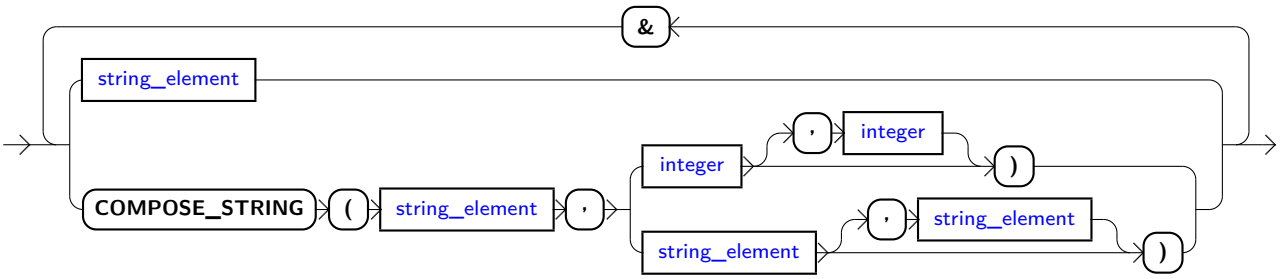
Example:

```
DATAPOOL
  REAL data, factor;
END DATAPOOL;

UI_MANAGER
  FIELDGROUP fg_identifier (
    "This fieldgroup shows following values:",
    " data: " data,
    " data * 1e3: " data * 1e3,
    " data / 2: " data / 2,
    " data * factor: " data * factor,
    " data / factor: " data / factor);
END UI_MANAGER

FUNCTIONS
  FUNC INIT {
    data = 20; factor = 2;
  };
END FUNCTIONS;
```

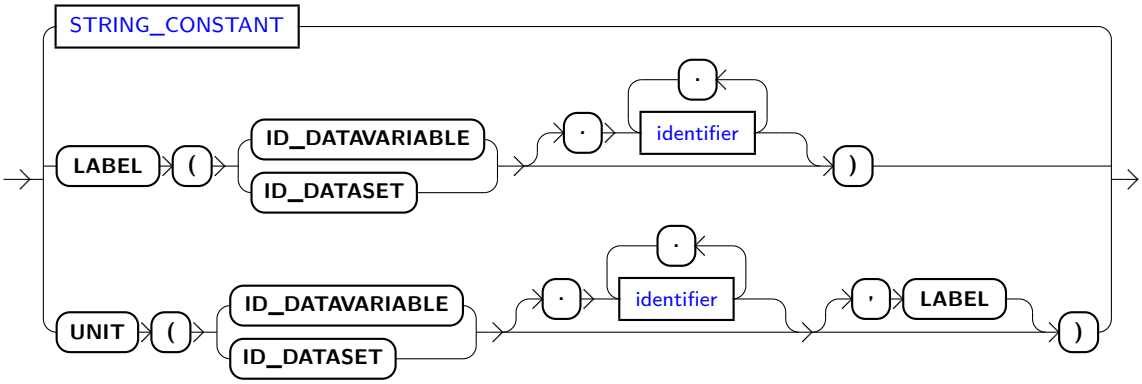
4.1.3 String

Strings are used in almost every section of this manual. They must be delimited by a double quote character (”):

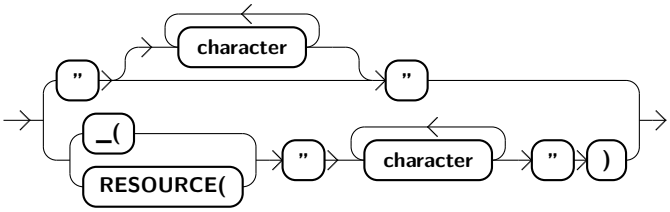
string



string_element



STRING_CONSTANT



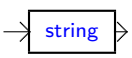
string	Description
character	any charachter sign, except double quote (”), at (@) and backslash (\)
	double quotes are used to delimit the string
	at’s are used to switch between fonts (see section multifont strings Multifont Strings page 25)
&	concatenate string with strings, LABEL’s or UNIT’s
`_("...")`	a string that can be translated (using GNU gettext)

RESOURCE	see section Resource on page 28
COMPOSE_STRING	Compose string: write values into string. Needed for translatable strings or to use a **RESOURCE** value in a string. The format-string (first argument) contains %1, %2, ... and the remaining value(s) (integer or string) are put in these places. One format-string with up to 3 integers or 15 strings are implemented. The composition is done at parse time.
LABEL	returns the label defined in the datapool (section data_item_options page 40).
UNIT	returns the unit description defined in datapool (section data_item_options page 40).
`ID_DATAVARIABLE`	references a data item declared in datapool (section Data Item page 36). For options like **LABEL** and **UNIT** use data items without indexes.
`ID_DATASET`	references a data set declared in datapool (section Data Set page 44).
.	structure item separator Structures are explained in section Structure definition page 49
`identifier`	references a data item declared in datapool (section Data Item page 36).

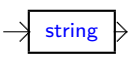
Example:

```
"This is a string delimited by double quotes"
"This is a " & "String"

"This string is concatenated with " & LABEL(data_item_identifier)
```

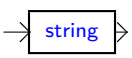
filename_string



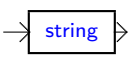
document_string



title_string



label_string



bg_color_string

fg_color_string

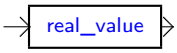
char_const

A CHAR_CONSTANT is a single character. It must be delimited by a single quote sign (').

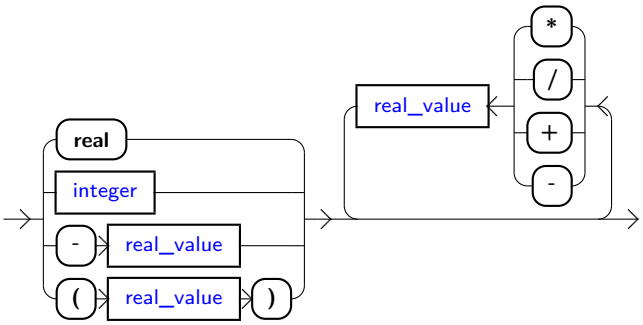
4.1.4 Numbers

Real or integer values are used in almost every section of this manual. They are number constants written directly in the description file:

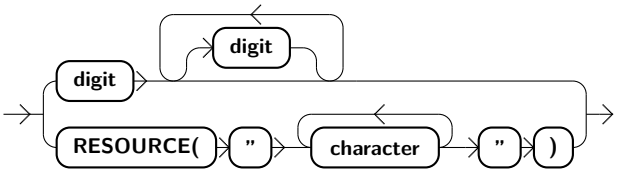
real_or_int_value



real_value

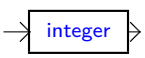


integer



string	Description
`real`	a floating point number
`integer`	an unsigned integer constant
digit	number sign (0..9)
RESOURCE	see section Resource on page 28

width



precision

4.1.5 Multifont Strings

Any string can be displayed with multiple fonts using the standard HTML elements.

Options	Description
`<tt>`	default monospace font face
`<em>`	default emphasis font face
`<b>`	default bold font face
`<strong>`	similar to `<b>`
`<small>`	use one font-size smaller
`<big>`	use one font-size larger
`<sub>`	subscripted text
`<sup>`	superscripted text
`<h1>`	section heading (h1 - h6)

Note: use of CSS is encouraged.

Example:

```
DESCRIPTION "Multifont strings";
UI_MANAGER
  FIELDGROUP
    fieldgroup_identifier (
      "<h1>Examples of multifont strings</h1>",
      "",
      "<b style='font-size:20px; color:#1c87c9;'>Bold Blue</b>"
          "<strong>STRONG</strong> <small>small</small>",
      "<em>Emphasized</em> <big>Big</big>"
          " normal text<sub>subcripted</sub>"
      "text<sup>sup</sup> <tt>monospaced</tt>"
    );
  FORM
    form_identifier {MAIN} ((fieldgroup_identifier));
END UI_MANAGER;
END.
```

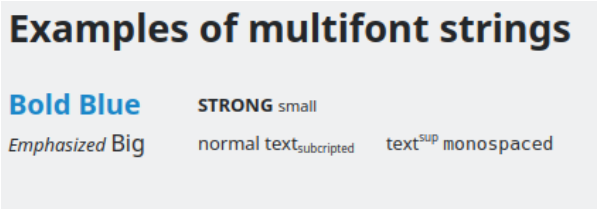


Figure 3: Multifont string

4.1.6 Setting locale

If the description files have been prepared to support multiple languages the active language is selected by setting an environment variable at intens startup:

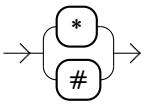
Example:

- Linux: LC_MESSAGES=de_CH.utf8
- Windows: LANG=de_CH.utf8

4.1.7 Wildcards

Wildcards are used as a place holder and are represented by the following two characters.

wildcard



Both of them have the same meaning. There is no difference between the interpretation of them. However, it is suggested to use only the asterisk.

Example:

```
Data_item_identifier[*]
Data_item_identifier[#]

Index_identifier[*]
Index_identifier[#]
```

4.1.8 Resource

INTENS can use the value of an environment variable or of a resource property set in a resource file (.ini, provided with --resfile, section [Resource]).

INTENS reads the **RESOURCE** at parse time. If neither environment variable nor resource property is found, a parser error is thrown and the application does not start.

The type of a **RESOURCE** is **INTEGER**, **REAL** or **STRING**. The type is based on the value of the resource:

- "123" is an **INTEGER**
- "3.14" is a **REAL**
- "123abc" is a **STRING**

Make sure the type matches the context of the **RESOURCE**: a parser error is thrown when **RESOURCE** is i.E. an **INTEGER** in a context where a **STRING** is needed.

Example:

```
MESSAGE_QUEUE mq_request{
  REQUEST
, HOST=RESOURCE("REPLY_HOST")
, PORT_REQUEST=RESOURCE("REPLY_PORT")
};
```

With a start script that includes the lines

```
export REPLY_HOST="localhost"
export REPLY_PORT=12345
```

INTENS replaces `RESOURCE("REPLY_HOST")` with the **STRING** "localhost" and `RESOURCE("REPLY_PORT")` with the **INTEGER** 12345.

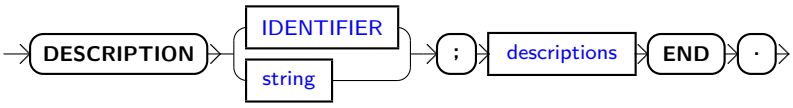
4.2 DESCRIPTION

The application title following the **DESCRIPTION** may be given as string or single word with no spaces. It will be displayed as title string in all windows of that application.

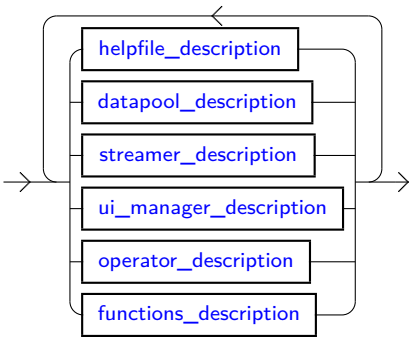
Each of the **INTENS** objects has its own description block. All blocks (except `language_description`) can be repeated in any sequence as long as the referenced identifiers within one block are declared in one of the previous blocks.

Description blocks **DATAPOOL** and **UI_MANAGER** are mandatory.

description



descriptions



description block	description
IDENTIFIER / string	application title (see above).
HELPFILE	section HELPFILE page 31.
DATAPOOL	section DATAPOOL page 35. This block is **mandatory**.
STREAMER	section STREAMER page 52.
UI_MANAGER	section UI_MANAGER page 67. This block is **mandatory**.
OPERATOR	section OPERATOR page 162.
FUNCTIONS	section FUNCTIONS page 189.

The following example shows a minimal configuration to give a first practical view of **INTENS**:

```
// SEMAFOR Informatik & Energie AG
// Basel, Switzerland
//

DESCRIPTION "Minimal Example";
DATAPOOL
  REAL{EDITABLE} speed;
END DATAPOOL;

UI_MANAGER
   FIELDGROUP
      g1( "speed:" speed "km/h" );
   FORM
      f1{MAIN}( (g1) );
END UIMANAGER;
END.
```

After reading the above configuration data **INTENS** has created a main window containing two text windows and a fieldgroup as shown in figure below. You will later see (section UI_MANAGER on page 67) how to configure additional forms and how to move the text windows to another form.

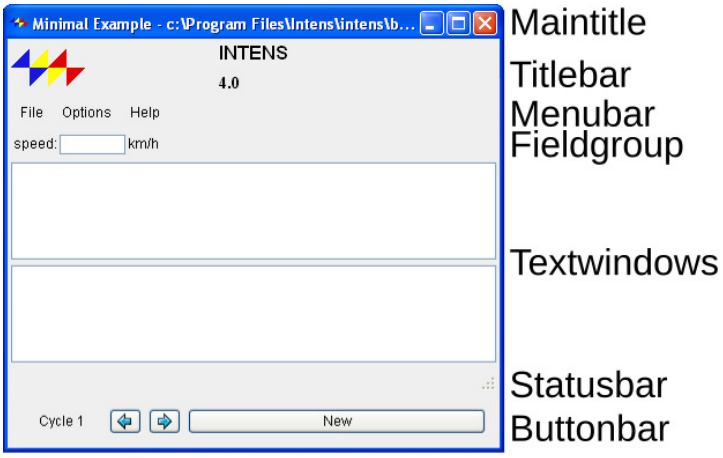


Figure 4: Minimal example

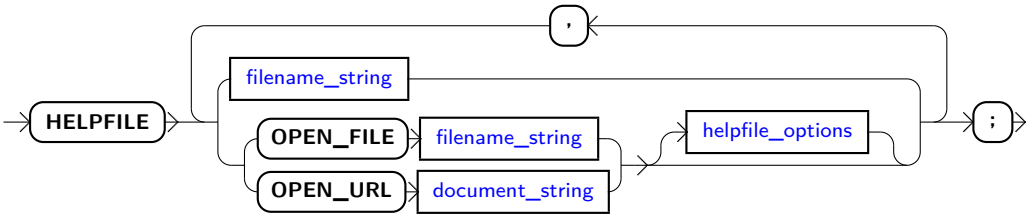
4.3 HELPFILE

4.3.1 Description

You can give any number of help text files to be made available to the user via the help menu button. **INTENS** provides two methods for displaying these files:

- display a simply structured text file in a special help window (see chapter **INTENS** Helpfiles).
- display a HTML[1] file in the web browser which will be started if not already running (see chapter HTML files).

helpfile_description



help_description	description
`filename_string`	string: Name of a helpfile (see chapter **INTENS** Helpfiles).
`document_string`	string: Name (address) of a document
OPEN_FILE	connect to the web browser and open the specified file (see chapter HTML files).
OPEN_URL	connect to the web browser and open the specified document by loading it from a WWW server (see chapter HTML files).

4.3.2 INTENS Helpfiles

These text files contain unformatted ASCII texts combined with some control information:

- The first line must begin with `#`. The characters immediately following until the end of that line will be used as title string.
- Lines beginning with a period define a new chapter. The following text on the same line will be used as chapter head.
- Lines beginning with a colon just after a chapter line define help keys. These help keys can be connected with forms (see chapter ui_form_option_list on page 145).

[1] Hypertext Markup Language

```
#INTENS
.Overview
Each INTENS application is created through a simple script which
contains the configuration commands for the applications data
exchange interfaces between RDBMS, calculation programs, user
interface, printer and file system.
.Parser
:PARSE
The Parser interpretes at program startup the configuration data
and sends the appropriate messages to all involved objects to
create the application with all configured interfaces.
.Datapool
  ...
```

help_file	description
#	begin of file and title string
.	begin of chapter and chapter label
:	help key (ui_form_option_list page 145)

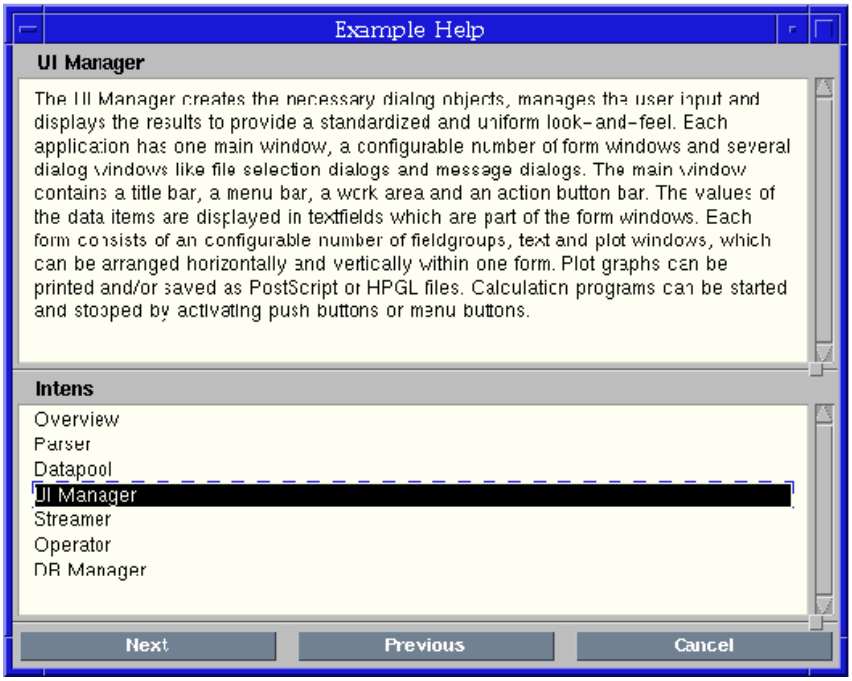
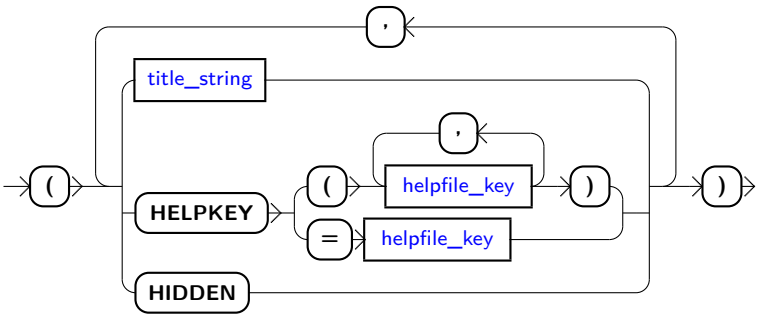


Figure 5: Example of a Help Window

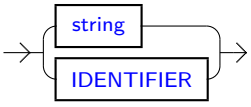
4.3.3 HTML files

For displaying HTML files with the web browser either use **OPEN_FILE** or **OPEN_URL**. The specified file must have a browser readable format such as HTML. The only thing **INTENS** does is connecting to the web browser and invoking its open command (with the `-remote` command).

helpfile_options



helpfile_key



help_options	description
`title_string`	string: Defines the label in the help menu. The filename is used if there is no title defined.
HIDDEN	No menu entry will be created for this helpfile.
HELPKEY	specifies the helpkeys (anchors) that are defined in the HTML file (see chapter ui_form_option_list on page 145).

If you want to reference specified anchors in your document, then you have to list them as **HELPKEY**s. The following example shows an anchor named ”goHere” in a HTML document:

```
...
<A NAME="goHere"<\A>
...
```

```
HELPFILE
  "helpfile.hlp",
  "helpfile_2.hlp",
  OPEN_FILE "helpfile.html"  HELPKEY("goHere"),
  OPEN_URL  "www.semafor.ch"
;
```

4.4 DATAPOOL

4.4.1 Description

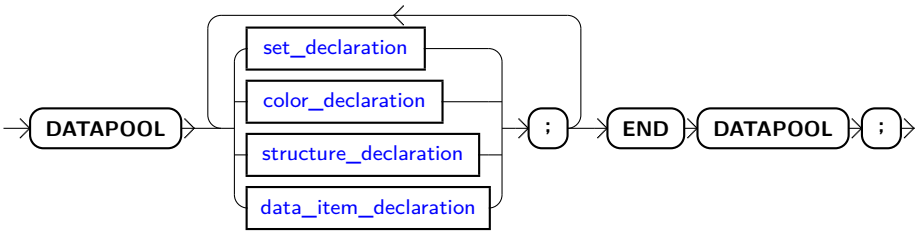
All **Data Items** used in an **INTENS**-application are defined in the **DATAPOOL** section. One of the most important functions of the datapool is the dynamic allocation of memory for these data items.

To define **STRUCTURES** you have to give an identifier, which will be used as a data type to define data items later. (see section Structure definition on page 49).

You can define **SET**s, which are used by the **UI_MANAGER** to show values of data items in a Combobox or Option menu. (see section Data Set on page 44).

You can also define **COLOR**s, which are used by the **UI_MANAGER** to show items with different colors corresponding to their values. (see section Data ColorSet on page 47).

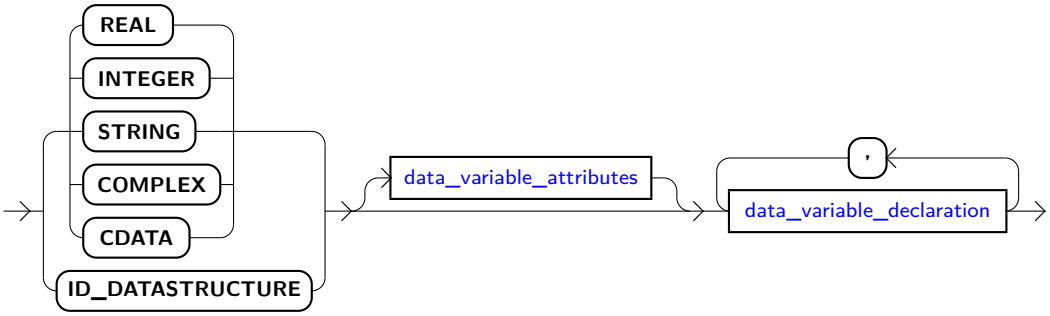
datapool_description



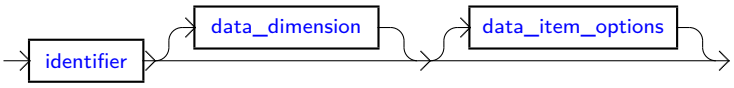
4.4.2 Data Item

Since the **DATAPOOL** allocates memory dynamically, it is not necessary to declare the dimension of the data items (unsigned integers between braces). Nevertheless the declaration is recommended, if the dimension is known and if it will not change during the calculation process. Their index range always begins at 0. See data_dimension on page 37 for the syntax.

data_item_declaration



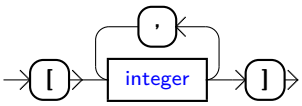
data_variable_declaration



data item	Description
`identifier`	data item identifier. It is needed for referencing this data item.
REAL	defines data items for storing real number values.
INTEGER	defines data items for storing integer number values.
STRING	defines data items for storing characters.
COMPLEX	defines data items for storing **complex** quantities.
CDATA	defines data items for storing character or binary data. (database: mapped to BLOB, no length restrictions while strings are limited to 255 characters)
ID_DATASTRUCTURE	defines data items for storing structured data.
	must be a previously defined structure (section Structure definition on page 49)

```
DATAPOOL
  REAL
    data_item_identifier_A[10],         // 1 dimension
    data_item_identifier_B[3,20],       // 2 dimensions
    data_item_identifier_C[2,4,10];     // 3 dimensions

END DATAPOOL;
```

data_dimension



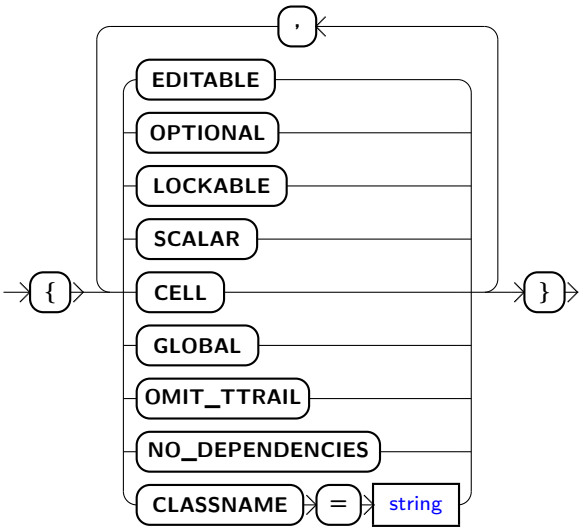
Declare the dimension of the data item. See first paragraph in section Data Item on page 36 for more information.

The following data items are **predefined** by the system at startup and must not be redefined:

Itemname	Description
DATE	contains the current date (yyyy-mm-dd, **STRING**)
USER	contains the user name (**STRING**)
HOST	hostname of the working machine (use VAR("HOST") to access it, **STRING**)
IPADDR	ip address of the working machine (**STRING**)
INTENS_VERSION	**INTENS** version (i.E. 5.3.1/5.3.2dev, **STRING**)
INTENS_VERSION_MAJOR	**INTENS** MAJOR version (i.E. 5, **INTEGER**)
INTENS_VERSION_MINOR	**INTENS** MINOR version (i.E. 3, **INTEGER**)
INTENS_VERSION_PATCH	**INTENS** PATCH version (i.E. 1/2dev, **STRING**)
INTENS_REVISION	**INTENS** revision (i.E. -/67-g0df41633, **STRING**)
RESTUSERNAME	username used to login to the RESTful web service (**STRING**)

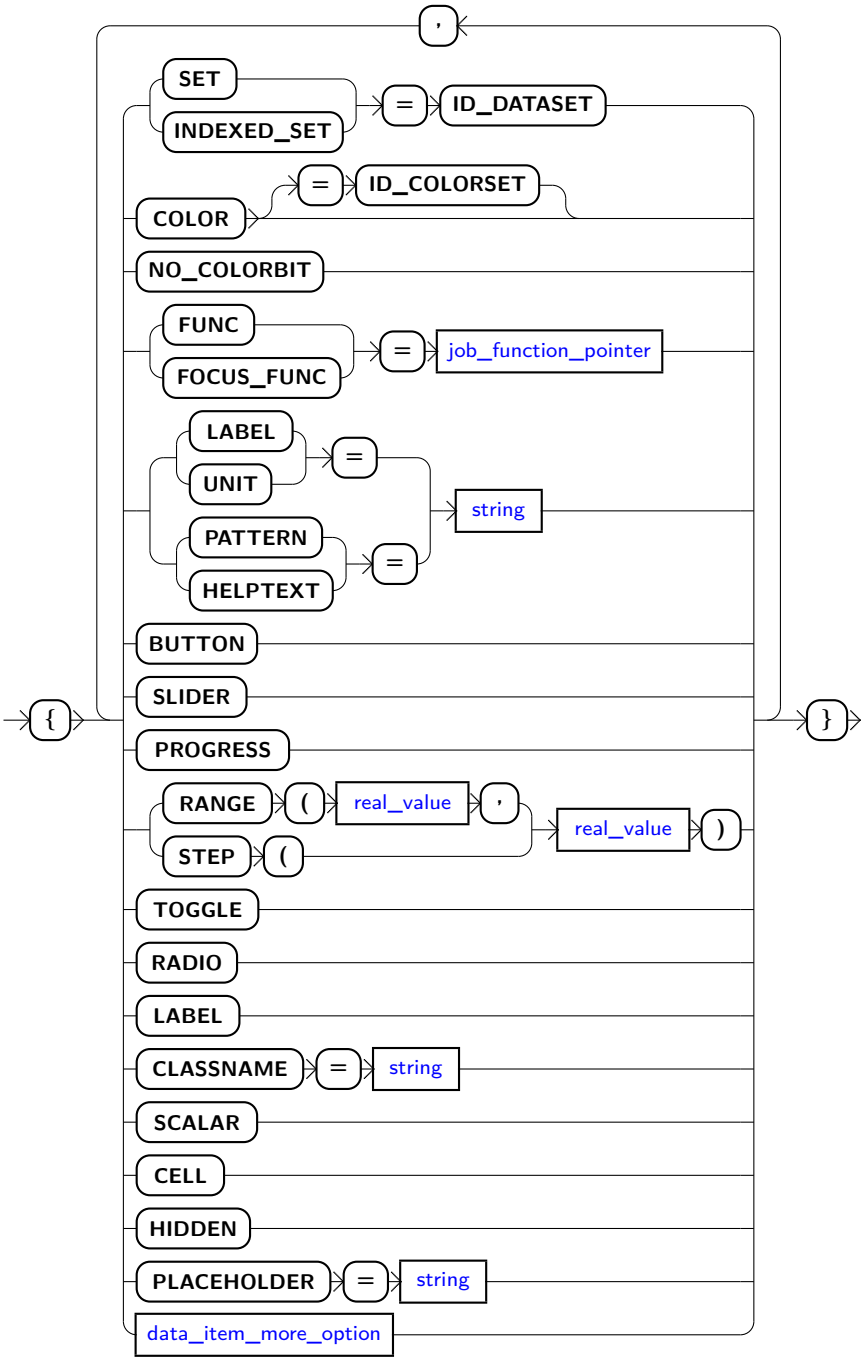
RESTUSERNAMELIST	list of usernames to show in login dialog for RESTful web service (**STRING**)
RESTBASE	base url of the RESTful web service (**STRING**)
REST_SERVICE.APP_VERSION_MAJOR	see paragraph Rest Service: Version control on page 222 (**INTEGER**)
REST_SERVICE.APP_VERSION_MINOR	see paragraph Rest Service: Version control on page 222 (**INTEGER**)
REST_SERVICE.APP_VERSION_PATCH	see paragraph Rest Service: Version control on page 222 (**INTEGER**)
REST_SERVICE.DB_VERSION_MAJOR	see paragraph Rest Service: Version control on page 222 (**INTEGER**)
REST_SERVICE.DB_VERSION_MINOR	see paragraph Rest Service: Version control on page 222 (**INTEGER**)
REST_SERVICE.DB_VERSION_PATCH	see paragraph Rest Service: Version control on page 222 (**INTEGER**)
REST_SERVICE.DB_VERSION_IGNORE	see paragraph Rest Service: Version control on page 222 (**INTEGER**)
PLOT2D_UIMODE	see paragraph UI Mode in section Plot2d on page 108 (**STRING**)
PLOT2D_SYMBOLSIZE	see paragraph Global Symbolsize in section Plot2d on page 109 (**INTEGER**)
Global_Point	structure object with data items X, Y and Y2 Used in UI Mode “Select Point”. See paragraph UI Mode in section Plot2d on page 108
Global_Rect	structure object with data items X1, Y1, X2 and Y2 Used in UI Mode “Select Rectangle”. See paragraph UI Mode in section Plot2d on page 108

data_variable_attributes

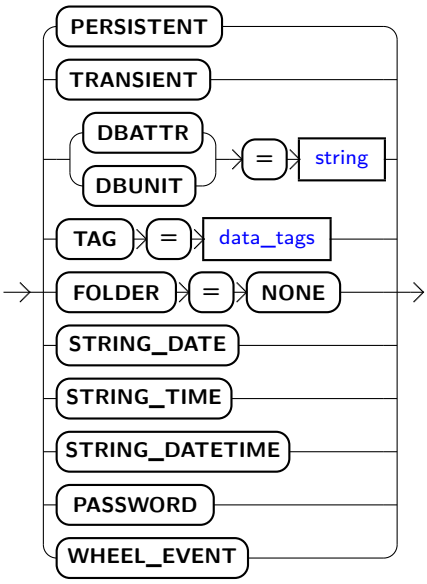


Attributes	Description
EDITABLE	The value can be changed (edited) by the user (see also section UI_MANAGER on page 67)
OPTIONAL	Same as **EDITABLE**. The corresponding text fields may be given different foreground/background color, font etc.
LOCKABLE	The data item can be protected against being overwritten by stream operations. (see also section UI_MANAGER on page 67)
SCALAR	Data items with this attribute are transferred as scalars instead of matrices to any Matlab function (or instread of vectors to and from the database). They are not written as a list in a **JSON STREAM**.
CELL	The data item is transferred to the Matlab workspace as cell instead of array.
GLOBAL	The data item is not changed by cycle operations.
OMIT_TTRAIL	The data item is not handled by transactions (ABORT, UNDO, ...).
NO_DEPENDENCIES	No dependencies between this input and its outputs (results) are added. (see paragraph Dependency on page 52)
CLASSNAME	Data items having a classname are transferred as objects to any Matlab function. The **CLASSNAME** can also be used in **FUNCTIONS**: **CLASSNAME**(item).

data_item_options



data_item_more_option

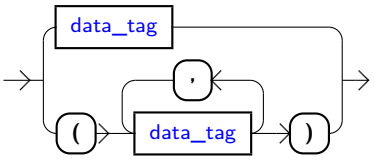


Options	Description
SET	The items values are part of a **SET**. A set is a limited number of values of the same type. The item is displayed as Combobox on the user interface. `ID_DATASET` must be previously defined (see section Data Set page 44)
INDEXED_SET	Like **SET**. When the data item is used as a vector, each element can have a different list of values to select from.
COLOR = ...	The field is shown in the colors that corresponds to its items value. `ID_COLORSET` must be previously defined (see section Data ColorSet page 47)
COLOR	String data item to define a color. The item is displayed as a color picker: a button that shows the color. Pressing the button opens a color dialog to choose a color. The chosen color is stored as the value of the variable.
NO_COLORBIT	The eight available color bits (see section set_statement on page 200) are interpreted as a 8-bit number. This gives the possibility to use up to 255 different colors.
FUNC	Defines the function that will be called after an interactive modification of the items value. (see also section FUNCTIONS on page 189)
LABEL = ...	Set the item's **LABEL** which can be used by the **UI_MANAGER** and **STREAMER**. The **LABEL** is set to the item's name by default if no **LABEL** is provided. (see also section Fieldgroup on page 71)
UNIT	The item has a unit which can be used by the **UI_MANAGER** and **STREAMER**. (see also section Fieldgroup on page 71)
PATTERN	A regular expression that defines the possible input values.

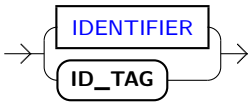
HELPTEXT	For each item a helptext may be defined. This text appears as soon as the mouse-pointer crosses the field, which contains the data-item.
BUTTON	The item is displayed as button on the user interface. Activating the button executes the associated function (see above **FUNC**).
SLIDER	The item is displayed as a slider.
PROGRESS	The item is displayed as a progress bar. This makes it possible to show a progress bar inside a **FIELDGROUP**. Assign the desired value (from 0 to 100) to the (integer) variable and the item shows the progress.
RANGE	Minimal and maximal values for a **SLIDER**.
STEP	Difference between the labels of **SLIDER**.
TOGGLE	The item is displayed as Toggle on the user interface. Activating the toggle changes the value from 0 to 1, deactivating changes the value from 1 to 0. A variable that has no value (invalid) is shown as a deactivated toggle, just as a variable with the value 0.
RADIO	The item is displayed as ragio button on the user interface. Activating the radio button changes the value from 0 to 1, deactivating changes the value from 1 to 0. A variable that has no value (invalid) is shown as a deactivated radio button, just as a variable with the value 0.
LABEL	The item is displayed as a label (not editable).
CLASSNAME	Data items having a classname are transferred as objects to the Matlab workspace. The **CLASSNAME** can also be used in **FUNCTIONS**: **CLASSNAME**(item)
SCALAR	Data items with this attribute are transferred as scalars instead of matrices to the Matlab workspace (or instread of vectors to and from the database). They are not written as a list in a **JSON STREAM**.
CELL	The data item is transferred to Matlab workspace as cell instead of array.
HIDDEN	The data item is not transferred to Matlab. It is, by default, not written in **JSON STREAM**s (see st_json_options on page 59). It is however sent to the rest service (see restService_statement on page 220).
PLACEHOLDER	For each item a placeholder text may be defined. This text appears in the field which contains the data-item when it is **INVALID**.
PERSISTENT	This data item can be stored to and retrieved from the database.
TRANSIENT	This data item is not stored to and retrieved from the database. It is useful in a **PERSISTENT STRUCT**.
DBATTR	defines the corresponding db attribute name.

DBUNIT	defines the corresponding db attribute unit.
TAG	The TAG-identifier is referenced by the navigators COL-definition. (see section ui_navigator_option_list on page 123)
STRING_DATE	String data item to show and enter a date (popup calendar may be used).
STRING_TIME	String data item to show and enter a time.
STRING_DATETIME	String data item to show and enter a date and time.
PASSWORD	String data item is displayed in password mode.
WHEEL_EVENT	The mouse wheel can be used to increment and decrement the value, just as the up and down keys can be used for every numeric data item. This does not work when commandline option --withoutArrowKeys is given. This option enables the mouse wheel event for one numeric data item, whereas the commandline option --withWheelEvent enables it for every numeric data item.

data_tags



data_tag

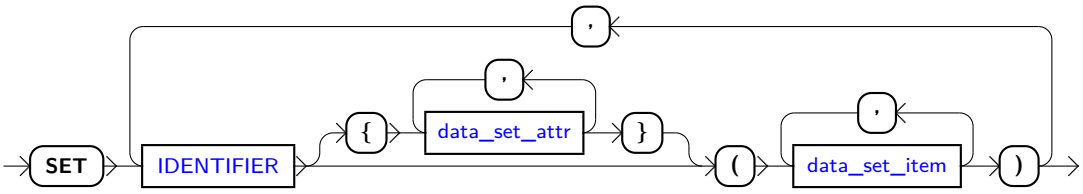


tag	Description
IDENTIFIER	Defines a new tag.
ID_TAG	References an existing tag.

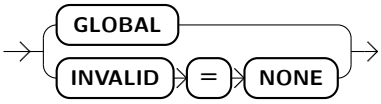
4.4.3 Data Set

Data sets are used for assigning labels to data items. Data sets may be assigned to data items using the option SET=data-set_identifier of a previously declared data set. (section data_item_options on page 40)
By editing such a data item the associated label-strings are shown, while the corresponding values are transferred to the external process.

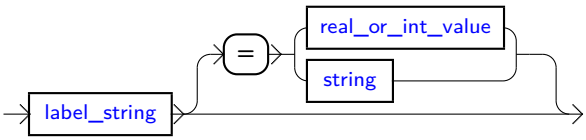
set_declaration



data_set_attr



data_set_item



Option	Description
GLOBAL	The data set is not changed by cycle operations.
INVALID=NONE	No empty (invalid) entry is created.

The label-strings of a set define the option menu labels. The value types can be **REAL**, **INTEGER** or **STRING**. The values correspond to the text of the label. If the values are missing, **INTENS** assumes increasing integer numbers starting at 0 for real and integer items. String items contain the string itself.

```
DATAPOOL
  SET
    set_identifier ( "circle" = 0, "triangle" = 1, "square"= 2 );

  INTEGER {EDITABLE}
    data_item_identifier {SET = set_identifier};

END DATAPOOL;
```

4.4.4 Dynamic Combobox

Sometimes it is not possible to use a **SET** to build a combobox. Then, a **STRING** item can be changed to a combobox by assigning a special value to it:

```
{"value": "a", "input":["A","B","C",null], "output":["a","b","c",null]}
```

It must be a JSON object with the members:

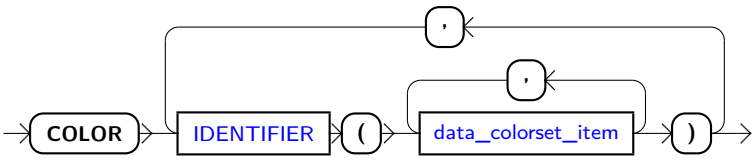
- *value*: the selected value. One of the values of *output*.
- *input*: list of strings. The elements presented to the user in a combobox.
- *output*: list of values (normally strings). The values corresponding to the *input* values.

When the first element of *input* is selected, *value* is set to the first element of *output*.
To extract the *value* from the total string in a function, use **STRING_VALUE**(item).

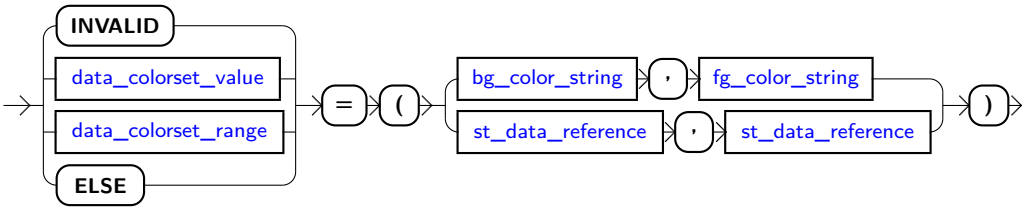
4.4.5 Data ColorSet

Data colorsets are used for assigning colors to values. Data colorsets may be assigned to data items using the option COLOR=data-colorset_identifier of a previously declared data colorset. (section data_item_options on page 40)
The field of such a data item will change to the associated colors that correspond to its value.

color_declaration



data_colorset_item



value	Description
INVALID	invalid value.
`data colorset value`	less than, more than or exaclty one specific value (see paragraph Data ColorSet Value and Range on page 48).
`data colorset range`	data range (see paragraph Data ColorSet Value and Range on page 48).
ELSE	all other values (not matched by any other colorset item values).

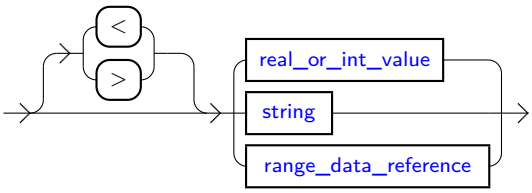
The pair of bg_color_string and fg_color_string of a colorset define the background and foreground color used for value defined on the left.
The color-string may be in one of these formats:

- #RGB (each of R, G and B is a single hex digit)
- #RRGGBB
- A name from the list of colors defined in the list of SVG color keyword names provided by the World Wide Web Consortium; for example, "steelblue" or "gainsboro". These color names work on all platforms.

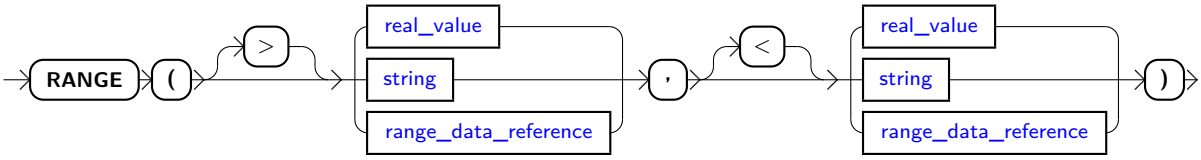
Instead of providing static color-strings, the colors can also be a data reference. If the data item has a valid color string value, that value is used. This can be used i.E. to change the color of plot curves (see section Plot2d on page 108).

Data ColorSet Value and Range

data_colorset_value



data_colorset_range



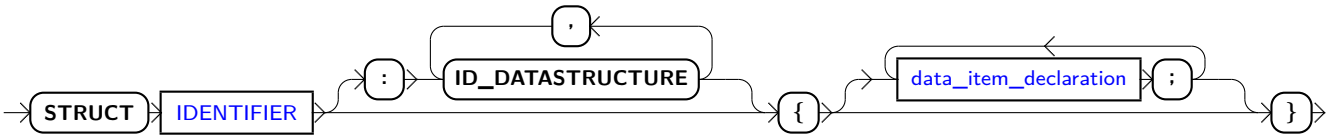
value	Description
`real or int value`	any value or mathematical 'combination' as shown in the example in section Scale Factors on page 20.
`string`	any string constant or 'combination'. (see section String on page 21)
`range data reference`	references a data item declared in the datapool (section temp_data_reference on page 51).

```
DATAPOOL
  COLOR
    red_green_blue_color (
      INVALID = ( "red", "black" ),
      < 0     = ( "#fff", "#f00" ),
      RANGE ( 0, <2 ) = ( "#ffffff", "#00ff00" ),
      2 = ( "#ffffff", "#0000ff" ),
      ELSE = ( "#ffffff", "#0000ff" )
    )
  ;

  INTEGER {EDITABLE}
    data_item_identifier {COLOR = red_green_blue_color};

END DATAPOOL;
```

4.4.6 Structure definition

Data items may be grouped in structures. This enhances the possibilities of storing and transmitting data.
A structure holds several data items of any valid data type described in section Data Item on page 36.

structure_declaration



Structure definition	Description
ID_DATASTRUCTURE	identifier of previously defined structure.
	inherit definition of this struct.
`data item declaration`	see section Data Item on page 36.

```
DATAPOOL
  STRUCT structure_def
  {
     INTEGER {EDITABLE}
        data_item_identifier_1,
        data_item_identifier_2;
     REAL {OPTIONAL}
        data_item_identifier_3;
  };

  structure_def
    structure_identifier_1,
    structure_identifier_2;

END DATAPOOL;
```

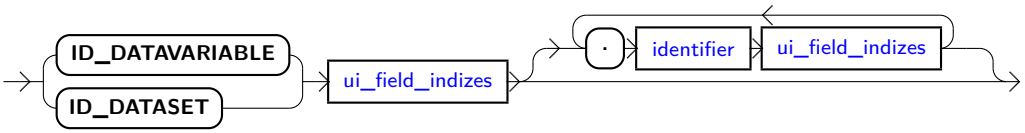
4.4.7 Examples

```
DESCRIPTION "Example of data defining and referring"

DATAPOOL
  SET shape_set ( "circle" = 0, "triangle" = 1, "square"= 2 );

  STRUCT Object {
     INTEGER {EDITABLE}
        shape {SET = shape_set,COMBOBOX
              ,LABEL = "The Shape"};
     REAL {OPTIONAL}
        size  {LABEL = "The Size", UNIT = "cm"};
  };

  Object obj[20];

  STRING {EDITABLE, SCALAR}
     text;

END DATAPOOL;

UIMANAGER
  FIELDGROUP
    fieldgroup_identifier
    (
      "Select first shape:" obj[0].shape,
      "Enter first size:"   obj[0].size,

      "Select last shape:"  obj[19].shape,
      "Enter last size:"    obj[19].size,

      "Enter a remark:"     text
    )
  ;

  FORM
    form_identifier {MAIN}
    (
      (fieldgroup_identifier)
    )
  ;

END UIMANAGER;
```

4.4.8 Data Reference

Temp Data Reference references a data item declared in the datapool (section Data Item page 36):

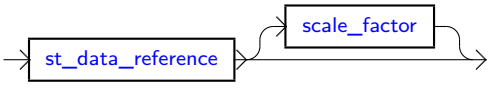
temp_data_reference



option	Description
`ui field indizes`	optional indizes (see page 70).
`.`	structure item separator Structures are explained in section Structure definition page 49
`identifier`	references a data item declared in datapool (section Data Item page 36). For options like **LABEL** and **UNIT** use data items without indexes.

Range Data Reference references a data item declared in the datapool (section Data Item page 36):

range_data_reference



option	Description
`scale factor`	a scale factor (see section Scale Factors page 20).
`st data reference`	references a data item declared in datapool (section Referencing data variables (Data item) on page 61).

4.5 STREAMER

4.5.1 Description

The Streamer manages all input and output formats. A stream consists of a sequence of data items and string constants.

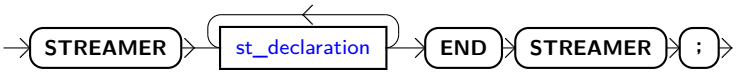
Dependency Together with the datapool and the operator, the streamer controls the dependencies between input and output variables (datapool items). In order to keep the data consistent, all data items that belong to an output stream are set invalid when one of the items of a corresponding input stream is modified.

Dependencies between input and output **STREAM**s are defined through **PROCESSGROUP**s and **MESSAGE_QUEUE REQUEST**s. They are activated when the **PROCESSGROUP** or the **MESSAGE_QUEUE REQUEST** is run.

The user is asked for confirmation before dependencies are cleared - unless the option **AUTOCLEAR_DEPEND** is used.

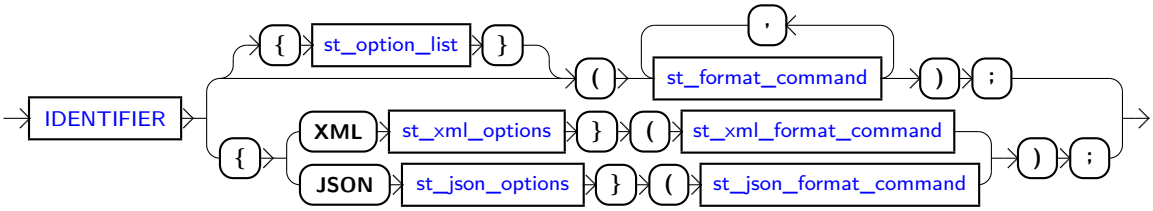
STREAMs can be excluded from dependencies using the option **NO_DEPENDENCIES** (see job_message_queue_option on page 214 or job_plugin_option on page 214) or by not giving a dependency option and using the **INTENS**command line argument --defaultMessageQueueDependencies false.

streamer_description



4.5.2 IO Stream

st_declaration



A stream is used to read or write to or from data items, referencing them by their identifier.

A stream is a sequence of format commands that has a unique identifier. An identifier within a format command references a data item. If an identifier is preceeded by an exclamation mark it will be checked to have a valid value before the corresponding calculation program is called. Item values that are not valid will not be transferred.

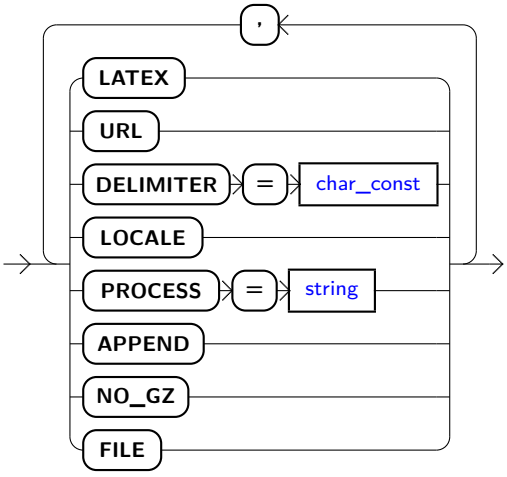
The streamer adds one blank character (see **DELIMITER** on page 53) after data items to separate them. This blank character is added only after data items and indices without a fixed width (see **field conversion** on page 56 and **field length** on page 57).

The line containing the **XML** token defines an xml stream (see section XML format command on page 57).

The line containing the **JSON** token defines an json stream (see section JSON format command on page 58).

Stream identifiers are registered datapool items and contain the filename (if any) of their last read or write operation.

st_option_list

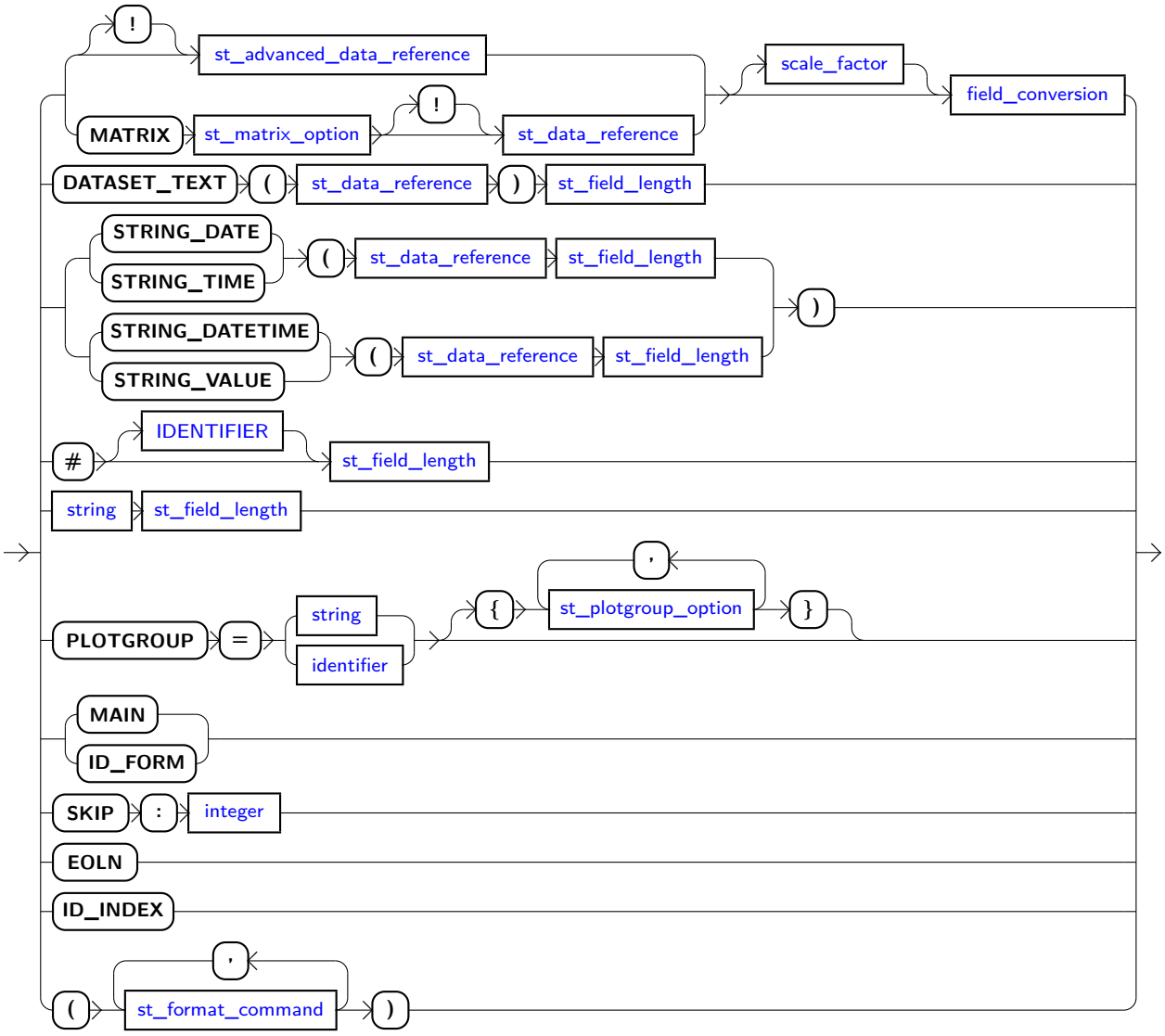


option	Description
LATEX	LaTeX special characters (i.E. \ ^ | or {) are replaced with their LaTeX command.
URL	reserved characters are percent-encoded and no **DELIMITER**s are added.
DELIMITER	defines the character used to separete stream items. default is the blank character.
LOCALE	stream uses locale decimal separator.
PROCESS	defines a filter process. The streams data is processed by the filter process.
APPEND	append received data to the **CDATA** element of the **STREAM**.
NO_GZ	Stream content is not gunzipped when reading a file stream, although the filename ends in '.gz'.
FILE	write to or read from a temporary file instead of the **DATAPOOL**. This reduces memory needed by **INTENS**, but values are not in the **DATAPOOL**.

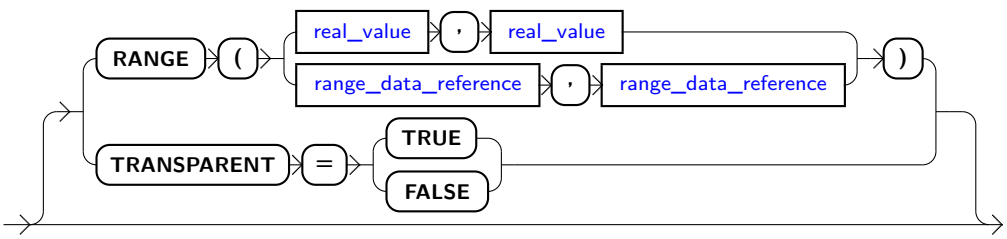
Not all combinations are allowed:

- **LATEX** or **URL**
  you can only choose one filter
- **URL** : no **DELIMITER**
  No **DELIMITER**s are added with **URL** option

st_format_command



st_plotgroup_option

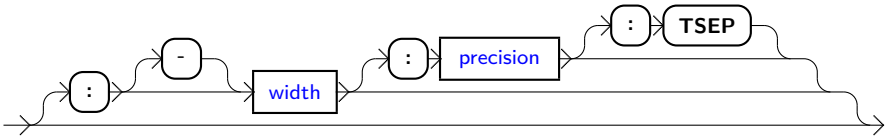


format command	description
!	perform a validation check
`data reference`	the possibly indexed data item (see section Referencing data variables (Data item) page 61)
MATRIX	The data item will be transferred as a matrix (see section Matrix page 64)
`scale factor`	see section Scale Factors page 20.
	The value of the data item is multiplied by the scale factor on an output operation and divided by the scale factor on an input operation.
`field conversion`	see page 56.
DATASET_TEXT	includes the string of the **SET** associated with the item (can only be used if the item has a **SET**)
STRING_DATE	prints the date in the local format (i.E. 24.01.08 instead of 2008-01-24) (see section data_item_options on page 40)
STRING_TIME	prints the time in the local format (i.E. 15:08:00:000 instead of 15:08:00) (see section data_item_options on page 40)
STRING_DATETIME	prints the date-time in the local format (i.E. 24.01.08 15:08 instead of 2008-01-24T15:08:00) (see section data_item_options on page 40)
STRING_VALUE	extracts the value of a dynamic **COMBOBOX** (see section Dynamic Combobox on page 46)
`#`	represents the explicit index of **pre-indexed** arrays.
	On input-operations data will not be filled into the array in order of incoming from the stream, but exactly to the position where this value points to.
	On output-operations it represents the index value of the data-item in the array.
`# IDENTIFIER`	`#` may be followed by any number or letter sign.
	This is used as an identifier when having more than one array-index.
	(section Referencing data variables (Data item) page 61), each array may have its own index.
`string`	a string constant (section String page 21)
`st field length`	see page 57.
PLOTGROUP	the following identifier specifies a plot group which will be included as PostSript file (Should only be used with LaTeX)
MAIN ID_FORM	**XML** tree of **MAINFORM** or **ID_FORM**.
SKIP`:n`	ignore the following `n` characters

EOLN	a new line
( )	Format commands within a second pair of parentheses build a repetition group. Each repetition group must have at least one array item.
RANGE (X-val1, X-val2)	Shows plot beginning at x-axis position X-val1 through X-val2
TRANSPARENT	Shows the plot with (**TRUE**, default) or without (**FALSE**) a transparent background.

```
STREAMER
  streamer_identifier_1 (data_item_identifier_1);
  streamer_identifier_2 (#, data_item_identifier_2[#]);
  streamer_identifier_3 (#1, (#2, data_id_1[#1], data_id_2[#2]));
END STREAMER;
```

field_conversion



Field conversion is optional.
The first : is followed by **width**, the second by **precision** and the third by **TSEP**.

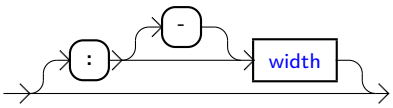
field conversion	description
-	Alignment: alignes the data item to the left (default is right)
width	defines the length of the field
precision	a) defines the number of digits after the decimal point for **REAL** items, b) has no meaning for **INTEGER** and **STRING** items
TSEP	a) Thousand separator (12'345.67) for **REAL** items, b) has no meaning for **INTEGER** and **STRING** items

```
STREAMER
  frequency_stream (frequency:10:2:TSEP);
END STREAMER;
```

gives the following stream:

123’456.78

st_field_length

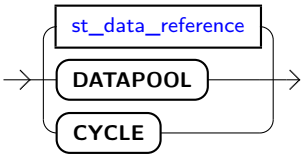


field length	description
`-`	Alignment: alignes the data item to the left (default is right)
`width`	defines the length of the field

4.5.3 XML format command

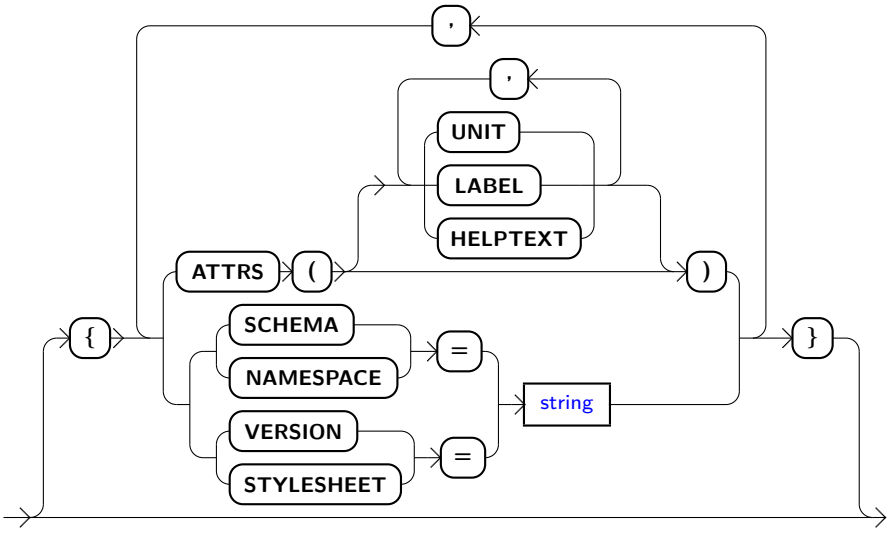
The **XML** token is used to export datapool variables to xml-files or import values from such files into datapool variables.

st_xml_format_command



xml format command	description
`st data reference`	see section Referencing data variables (Data item) on page 61
DATAPOOL	all data items defined in datapool are in- or exported by filestream actions.
CYCLE	all data items defined in cycle are in- or exported by filestream actions.

st_xml_options



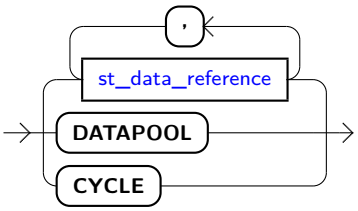
xml format option	description
UNIT	include unit attribute (see chapter data_item_options on page 40).
LABEL	include label attribute (see chapter data_item_options on page 40).
HELPTEXT	include helptext attribute (see chapter data_item_options on page 40).
SCHEMA	set the xml schema.
NAMESPACE	set the xml namespace.
VERSION	add a version attribute to the xml-file.
STYLESHEET	set the xml stylesheet.

```
STREAMER
  xml_stream_1 {XML { ATTRS ( UNIT ) } } (data_item_identifier);
  xml_stream_2 {XML} (DATAPOOL);
END STREAMER;

OPERATOR
  FILESTREAM
    FStream_1 = xml_stream_1;
    FStream_2 = xml_stream_2;
END OPERATOR;
```

4.5.4 JSON format command

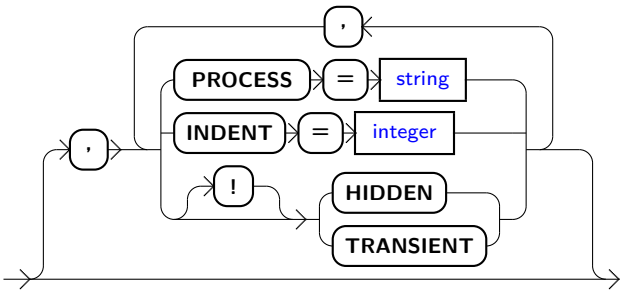
The **JSON** token is used to export datapool variables to json-files or import values from such files into datapool variables.

st_json_format_command



json format command	description
`st data reference`	see section Referencing data variables (Data item) on page 61
DATAPOOL	all data items defined in datapool are in- or exported by filestream actions.
CYCLE	all data items defined in cycle are in- or exported by filestream actions.

st_json_options



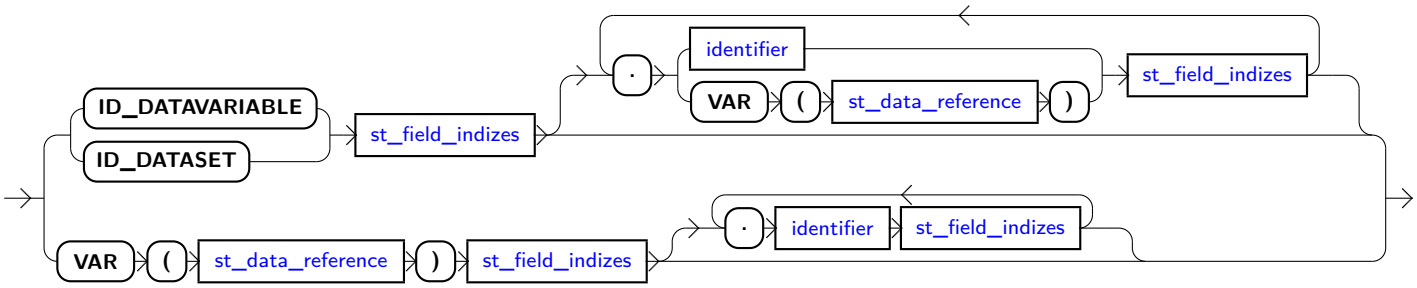
json format option	description
PROCESS	defines a filter process. The streams data is processed by the filter process.
INDENT	defines the indent level of a **JSON STREAM** (pretty print). 0 (default): no indentation.
HIDDEN	write values of hidden variables (see data_item_options on page 40)
! HIDDEN	don't write values of hidden variables (default)
TRANSIENT	write values of transient variables (default, see data_item_more_option on page 41)
! TRANSIENT	don't write values of transient variables

```
STREAMER
  json_stream {
    JSON
  , INDENT = 4
  , HIDDEN
  , !TRANSIENT
  } (data_item_identifier);
END STREAMER;
```

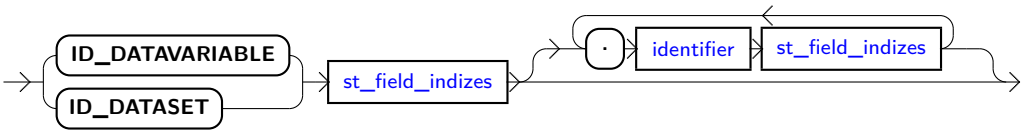
4.5.5 Referencing data variables (Data item)

Data items are declared in section DATAPOOL on page 35.
A stream is used to read or write to or from data items, referencing them by their identifier.

st_advanced_data_reference

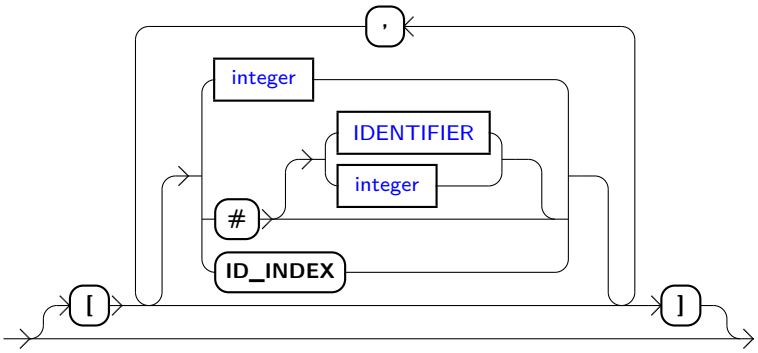


st_data_reference



data item	description
ID_DATAVARIABLE	data item identifier previously defined in section Data Item page 36
ID_DATASET	data set identifier previously defined in section Data Set page 44
`identifier`	data item identifier previously defined in section Data Item page 36
`st field indizes`	index (list) (see page 62)
VAR(string-item)	String dataitem contains the identifier of a data item. By changing the contents of string-item at runtime, specified datapool item is referenced.
.	structure item separator Structures are explained in section Structure definition page 49

st_field_indizes



index	description
INT_CONSTANT	a constant index
# (or empty)	the index of the array items (this is called pre-indexed)
#aZ9	# may be followed by any number or letter sign. This is used as an identifier when having more then one array-index. Must be previously declared in stream-declaration format-command (section st_format_command page 54).
ID_INDEX	an index identifier (see section Index-Object on page 95

If a repetition group has no index as token (is not pre-indexed), the column index of the matrix items is assumed to be continuous and is beginning with 0.

Pre-indexed repetition groups ignore newline characters on input. Their index must be the first element in the group.

On input operations a string constant within the stream description is considered as a search token.

4.5.6 Examples

```
STREAMER
  file_in(
    asm_type:-20,asm_descriptor:-35,doc_id:-12, \n,
    pz, rS, rR, lSigmaS*1e3, lSigmaR*1e3, lh*1e3, \n, \n,
    (psiMag[], iMag[], \n), \n, un, fn, \n,    klSamples
  );

  in(
    klSamples, un, fn, pz, rS, rR, lSigmaS, lSigmaR, lh, \n
  );

  out(
     ( speed[], i1[], i2[], torque[], cos[], \n )
  );

  report_out(
    \n, DATE, \n,
    \n, "Induction Motor Calculation Report", \n,
        "---------------------------------", \n,
    \n, "Machine Parameters:", \n,
    "N/(1/min): ", (speed[]), \n  );
END STREAMER;
```

Files structured like the following example can be read with the above declared stream `file_in`

```
1LA5 163-2CA        3000 1/min, 380V/50Hz              CHAX12345
 4 0.094 0.0847  1.05 0.78 28.4

       348.     18.17
       435.     22.71
       522.     27.35
      1305.    179.28
      1392.    215.73
      1479.    249.70
      1566.    292.42
      1653.    333.05
      1740.    372.93
      1827.    410.90

380 50
10
```

4.5.7 Matrix

The **MATRIX** token is used mainly with MATLAB interfaces as shown in the following example:

```
matlab_out
  ( Luser,ueuser
  , MATRIX Lin
  , Linmin ,Linmax
  , MATRIX ue[1,101]
  , U2effmax ,U2effnenn ,U2effmin
  );
```

The matrix is a multi-dimensional structure who's values are transferred in row-major order:

```
<dimension> <dim1> <dim2> <dim3> ...
<M001> <M002> <M003> ... <M010> <M011> <M012> ... <M020> <M021> ...
```

The first value indicates the number of dimensions, followed by the number of items in each dimension.

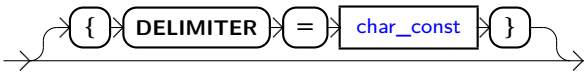
Example of a 3x3 matrix:

```
2 3 3
0 1 2 1 0 1 2 2 0
```

gives the following matrix:

0	1	2
1	0	1
2	2	0

st_matrix_option



matrix option	description
DELIMITER	character used to separate strings. (Default: '\0') (can only be used if the item is a string).

4.5.8 How to send data through a stream

First you need to define a process which generates data. This may be a unix-command like cat that reads from standard input and writes to standard output channel (see section Process page 162).
Each stream may be declared as input or as output by the processgroup definition (section Process Group page 164).

- Input-stream
  Read from datapool (data-item) write to standard output.

- Processgroup
  Pipe standard output to standard input of process.

- Process
  Execute process (eg. cat MyFile.dat).

- Processgroup
  Pipe standard output of process to standard input.

- Output-stream
  Read from standard input write to datapool (data-item).

```
DATAPOOL
  STRING
    data_item_output[15]
  ;
END DATAPOOL;

STREAMER
  streamer_id_out (#, data_item_output[#], EOLN) ;
END STREAMER;

OPERATOR
  PROCESS
    myfile_p : BATCH {
      "cat MyFile.dat"
    }
  ;
  PROCESSGROUP
    load_myfile { "Load" } (
      streamer_id_out = myfile_p (  );
    )
  ;
END OPERATOR;
```

The unix-command ”cat MyFile.dat” reads the file MyFile.dat and sends its contents to standard output.

Contents of ”MyFile.dat”:

```
0 Each
1 word
2 of
3 this
4 file
5 will
6 be
7 stored
8 in
9 data_item_output
10 after
11 the
12 processgroup
13 is
14 executed
```

Executing the process group load_myfile by pressing the button ”Load” sets the values as follows:

```
data_item_output[0]  <-- "Each"
data_item_output[1]  <-- "word"
data_item_output[2]  <-- "of"
data_item_output[3]  <-- "this"
data_item_output[4]  <-- "file"
data_item_output[5]  <-- "will"
data_item_output[6]  <-- "be"
data_item_output[7]  <-- "stored"
data_item_output[8]  <-- "in"
data_item_output[9]  <-- "data_item_output"
data_item_output[10] <-- "after"
data_item_output[11] <-- "the"
data_item_output[12] <-- "processgroup"
data_item_output[13] <-- "is"
data_item_output[14] <-- "executed"
```

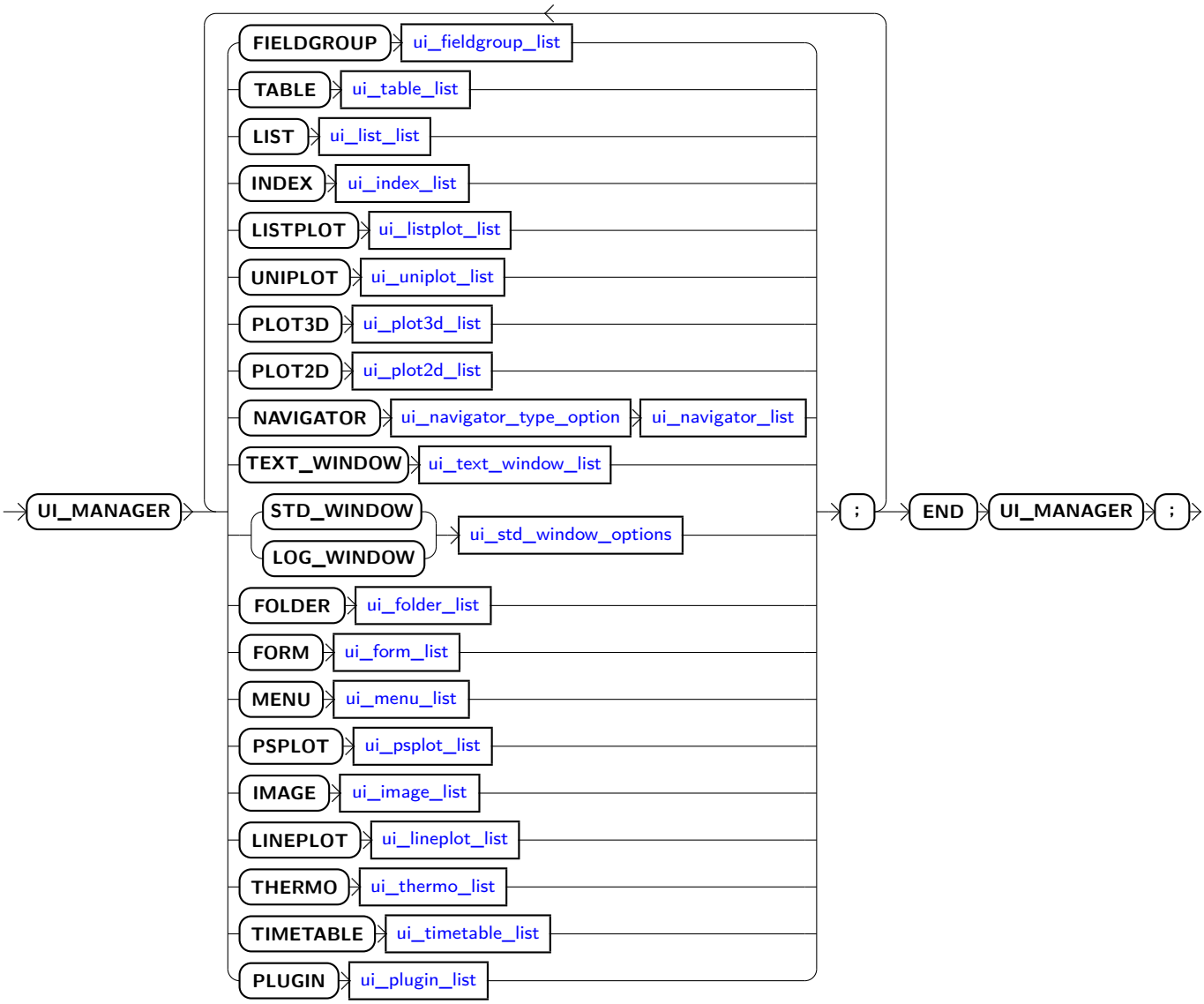
4.6 UI_MANAGER

4.6.1 Description

The **UI_MANAGER** block defines the appearance of the user interface without having the system administrator to place the interface objects geometrically. **INTENS** places the text fields, plots, tables, buttons etc. automatically after calculating their sizes.

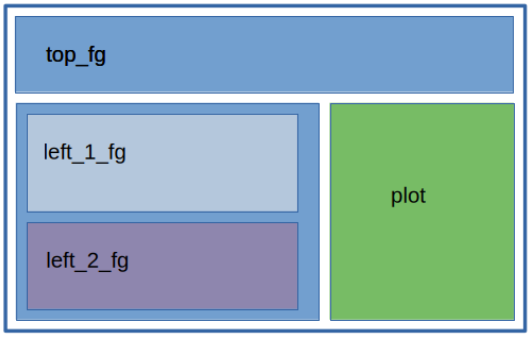
NOTE: Due to this feature, the layout of the windows may look slightly different after changing the language.

ui_manager_description



An **INTENS**-application usually has a main window with a menubar and several user defined forms. Each form may contain any number of horizontally and vertically arranged fieldgroups, text and plot windows and folder groups. Its layout is defined within a form declaration. Each pair of paranthesis toggles the orientation starting with a vertical orientation.

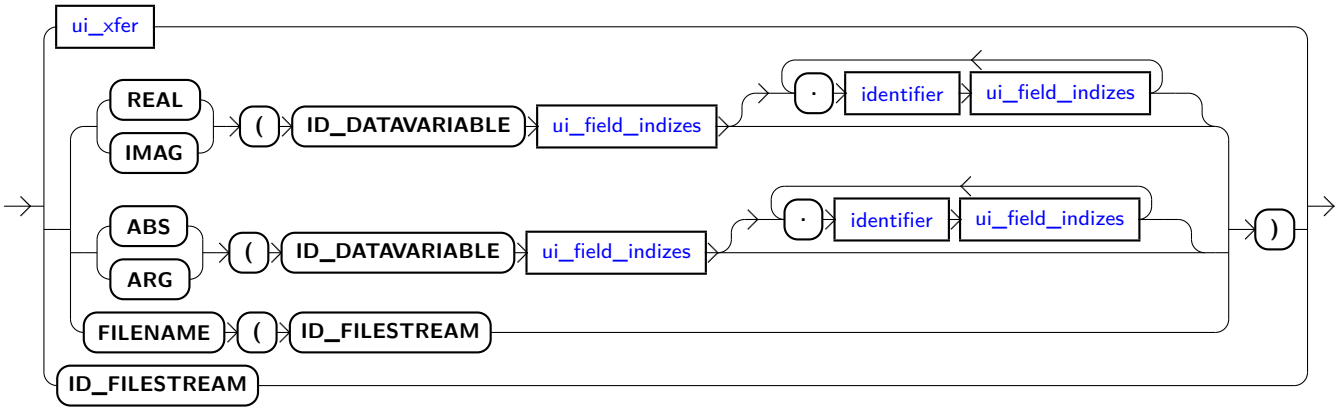
```
FIELDGROUP
  top_fg ( ... ),
  left_1_fg ( ... ),
  left_2_fg ( ... );

PLOT2D plot ( ... );

FORM main_form {MAIN} (
  top_fg,
  ( (
  left_1_fg,
  left_2_fg ), plot )
);
```



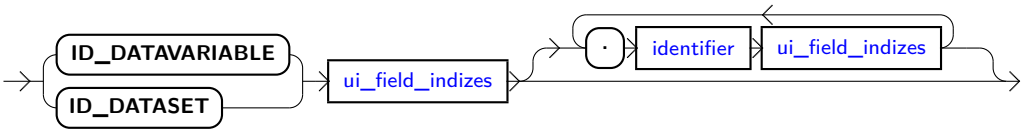
4.6.2 Data Variables

Most of the GUI-objects defined in the **UI_MANAGER** are directly linked with data items of the datapool. These objects are used to enter or display values.

ui_field_data_reference

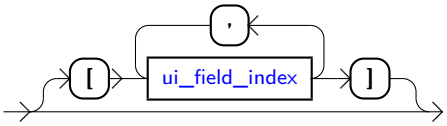


ui_xfer

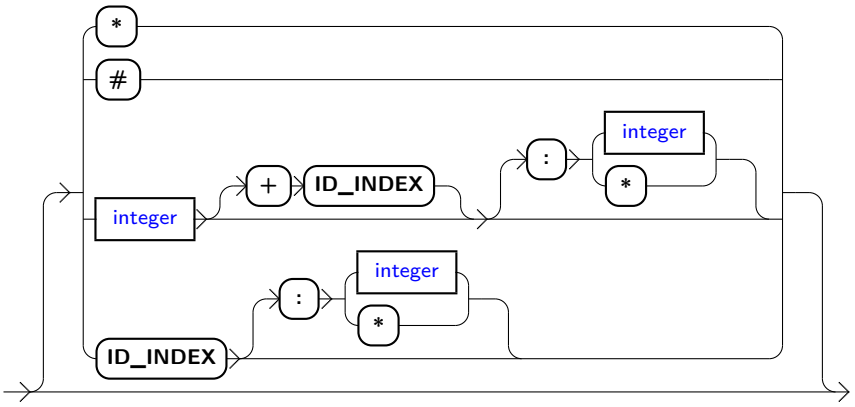


function	description
REAL	Real value of **COMPLEX** data item
IMAG	Imaginary quantity of **COMPLEX** data item
ABS	Absolute value
ARG	Angle of **COMPLEX** data item
FILENAME	Name of the file saved or opened through a filestream.
ID_DATAVARIABLE	references a data item declared in datapool (section Data Item page 36).
ID_DATASET	references a data set declared in datapool (section Data Set page 44).
.	structure item separator

ui_field_indizes



ui_field_index



By using indexes you can refer to each value of a data item.

index	description
ID_INDEX	refers a index object defined in UI_MANAGER (section Index-Object page 95)
+	add constant offset to an array-index
wildcard	see section Wildcards page 27
:	specifies a range (eg. first 5 [0:4])

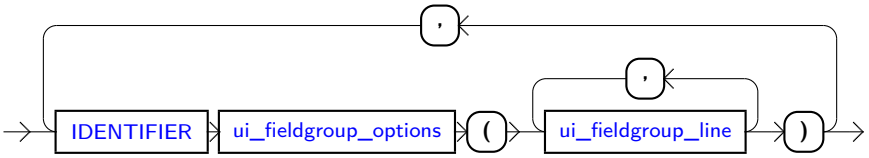
Examples:

```
data_item_id      // a scalar item with index 0,0 (or matrix item)
data_item_id[ ]   // an array item
data_item_id[n]   // a scalar item with index 0,n
data_item_id[*]   // an array item with running index on row
data_item_id[m,*] // an array item with running index on column
data_item_id[m,n] // a scalar item with index m,n

data_item_id[index-object_identifier]
                        // displayed as an index-object,
                        // which enables you to scroll through
                        // the data item using the mouse pointer.


data_item_id[ 5 + index-object_identifier]
                        // adds an offset of 5 to the index-object.
```

For options like **LABEL** and **UNIT** use data items without indexes.

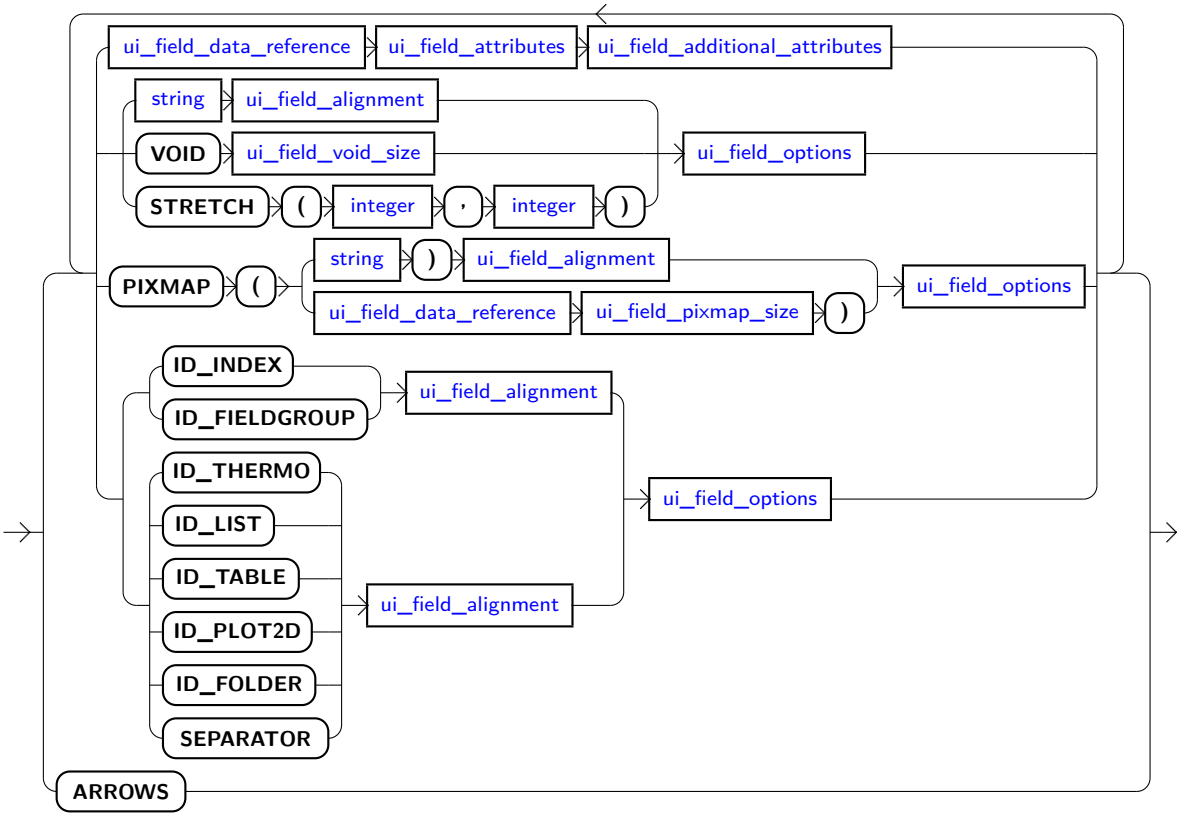
4.6.3 Fieldgroup

ui_fieldgroup_list



Fieldgroups have a unique identifier and consist of vertically aligned field lines separated by commas. A field line has strings, text fields, option menus or a pair of arrows to scroll the data items within a table. It can also have many other gui elements (INDEX, LIST, TABLE, ...). All fieldgroup items are placed left justified except items that are followed by a `'>'` or a `'|'`.

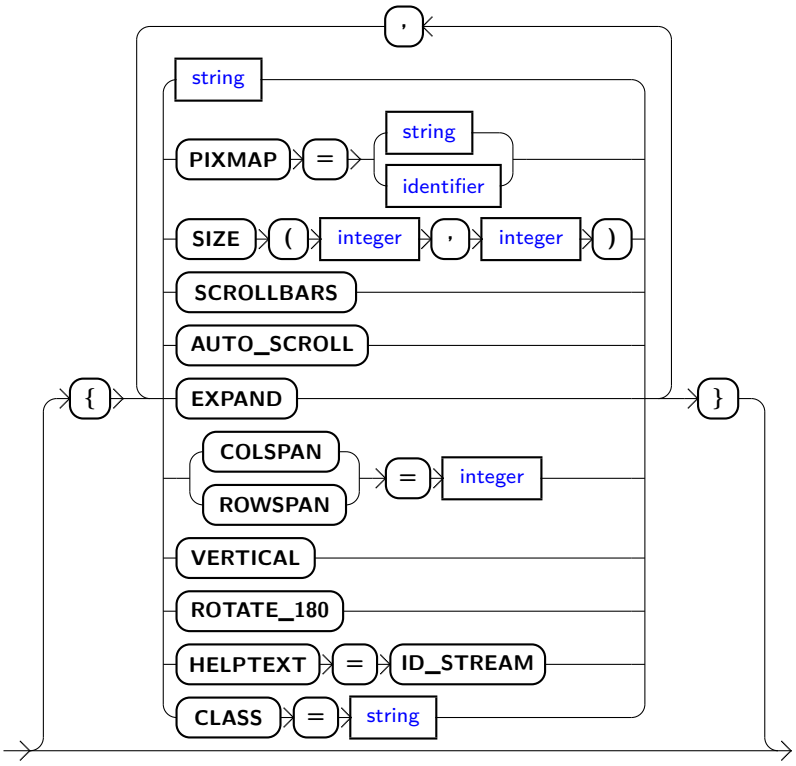
ui_fieldgroup_line



line element	description
`data reference`	display a data item (see section ui_field_data_reference on page 69)
`string`	creates a text label.
VOID	creates an empty space which can be used for alignment.

STRETCH	define stretch factors for the column and row. They make the column / row expand if additional space is available. A higher stretch factor expands more.
PIXMAP(string)	Filename (with or without extension). Displays a image. File is searched in following directories: $BITMAP_PATH : colon (linux) or semicolon (windows) separated list of directories $APPHOME/bitmaps IntensHome/bitmaps : IntensHome is the parent directory of the **INTENS** executable ./bitmaps .
PIXMAP(data_reference)	Datapool variable (**STRING** or **CDATA**). The value of the variable can be a filename (with or without extension) of the image to display or the image itself (image file content). If it is a filename, the file is searched in the directories as described above (**PIXMAP**(string)).
`ID_INDEX`	display an index object (see section Index-Object on page 95)
`ID_FIELDGROUP`	display a fieldgroup object (see section Fieldgroup on page 71)
`ID_THERMO`	display a thermo object (see section Thermo on page 157)
`ID_LIST`	display a list object (see section List on page 92)
`ID_TABLE`	display a table object (see section Table on page 85)
`ID_PLOT2D`	display a plot2d object (see section Plot2d on page 108)
`ID_FOLDER`	display a folder (see section Folder on page 136)
SEPARATOR	creates a horizontal (default) or vertical separator To create a vertical separator, add the option **VERTICAL**. To create a vertical separator over multiple rows, add the option **ROWSPAN**= n. The vertical separator can be left (default), center or right aligned (within the column). Example: centered vertical separator, two rows high: **SEPARATOR**| {**VERTICAL**, **ROWSPAN**=2}
ARROWS	inserts a row/column with an additional index (to scroll the table elements) and row/column header. Only allowed in tables. (Fieldgroup must have **TABLESIZE** set.)

ui_field_additional_attributes



attribute	description

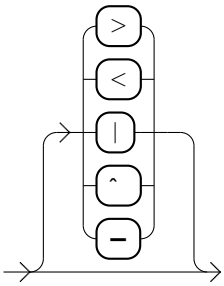
`string`	With a **BUTTON**:
	show this string on the **BUTTON** instead of its **LABEL**.
	Combine this with **PIXMAP** to show a **PIXMAP** and a string on a **BUTTON**.
	Examples:
	`open { _("Open") }`
	`open { PIXMAP="open", _("Open") }`
	`open { PIXMAP="open", LABEL(open) }`

PIXMAP	display button-items picture faced instead of text faced.
SIZE(w, h)	scale the **PIXMAP** to this size (in pixel)
SCROLLBARS	attach a scrollbar to the multiline textfield
AUTO_SCROLL	automatically scroll to the end of the multiline textfield
EXPAND	make field expandable
COLSPAN = n	ui field uses n columns
ROWSPAN = n	ui field uses n rows
VERTICAL	draw ui field vertically
ROTATE_180	draw ui field rotated 180 degrees
HELPTEXT	use the given **STREAM** as the **HELPTEXT** of the field
CLASS	set **CLASS** property of the field (used in qt style sheets)

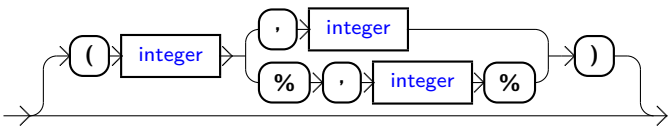
Example:

```
DESCRIPTION "Example PIXMAP";
DATAPOOL
  SET
    pixmap_set ("plot2d", "semafor-logo");
  STRING {EDITABLE}
    filename {SET = pixmap_set};
END DATAPOOL;

UI_MANAGER
  FIELDGROUP
    Group_1 (
        filename PIXMAP( filename, 100, 50 )
      )
   ;
  FORM
    Form_1
      ( ( Group_1 )
      )
   ;
END UI_MANAGER;
END.
```

ui_field_alignment



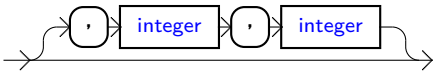
ui_field_void_size



alignment	description
<	the field is right aligned (within the column)
>	the field is left aligned (within the column)
|	the field is centered (within the column)

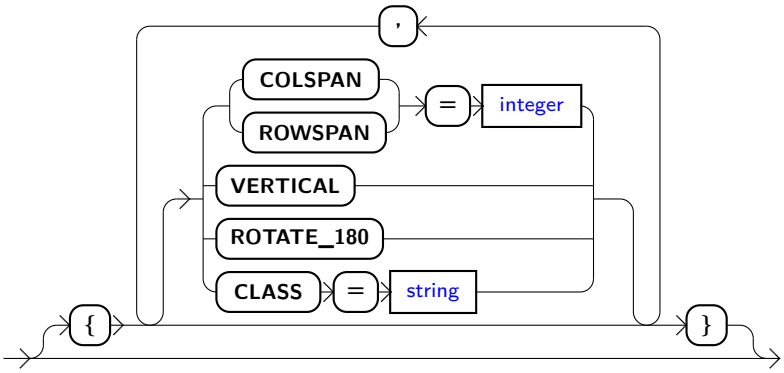
^	the field is top aligned (within the row)
_	the field is bottom alligned (within the row)
void size	minimal width, height of the field w, h in pixels or relative to form size (%)

ui_field_pixmap_size



pixmap field size	description
empty	size of the image is used
, x, y	size in pixels the image is scaled to

ui_field_options



option	description
COLSPAN = n	ui field uses n columns
ROWSPAN = n	ui field uses n rows
VERTICAL	draw ui field vertically
ROTATE_180	draw ui field rotated 180 degrees
HELPTEXT	use the given **STREAM** as the **HELPTEXT** of the field
CLASS	set **CLASS** property of the field (used in qt style sheets)

The value displayed in the textfield is the value of the data item (optionally multiplied by a scale factor) referenced by the identifier.

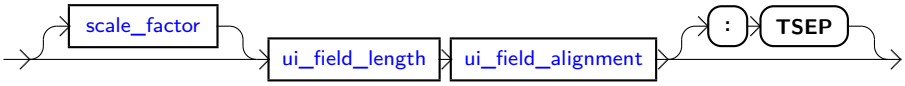
Text fields have a default width of 8 characters. It can be changed providing a ui_field_length. The first integer (after the :) is the width, the optional second integer is the precision. If the width is a negative integer value, the alignment of the value within the field is inverted: strings are left aligned (default is right), numbers are right aligned (default is left). The precision

specifies the number of lines for a string or the number of digits after the decimal point for a real number.

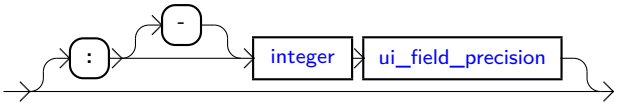
Examples:

- `s:50:3`
  STRING s in a text area 50 characters wide and 3 high
- `r*1e3:-12:3:TSEP` with r = 12.3456789
  |12'345.679 |
  value of **REAL** r multiplied by 1000, value is left aligned, field width is 12 characters, precision 3, thousand separator is shown

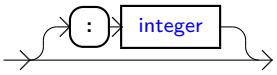
ui_field_attributes



ui_field_length

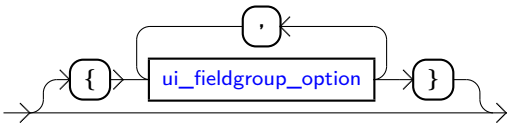


ui_field_precision

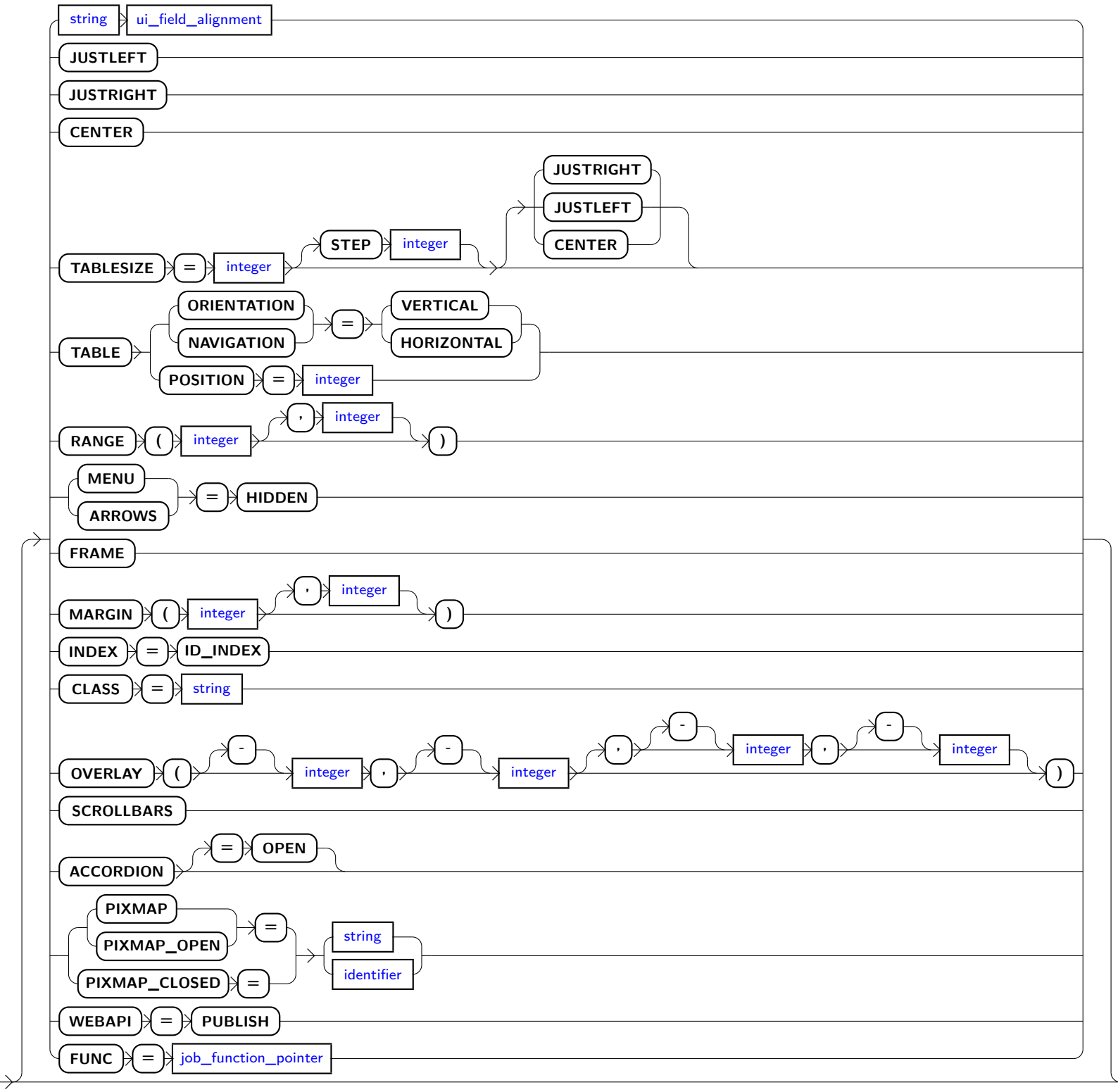


data item format	description
`scale`	see section Scale Factors page 20
	defines the factor that multiplies the items value before displaying and the divisor that divides the input value before saving in the datapool. (has no meaning for STRING items)
`-`	Alignment: inverses the alignment
	a) alignes the **STRING** data item to the right (default is left)
	b) alignes the numeric data item to the left (default is right)
`width`	defines the length of the field
	a) only that many characters can be entered for numeric items,
	b) more characters can be entered for **STRING** items
`precision`	a) defines the number of digits after the decimal point for **REAL** items,
	b) defines the number of lines for **STRING** items
TSEP	Thousand separator (12'345.67) (for **REAL** items only).

ui_fieldgroup_options



ui_fieldgroup_option



The FIELDGROUP options define the behaviour and appearance of the fieldgroup.

option	description
`string`	The title of the fieldgroup

JUSTLEFT	
JUSTRIGHT	
CENTER	Without one of these options, **INTENS** places the table fields left justified within each column.
TABLESIZE	creates a table with arrow buttons to scroll through the items. There is only one `data_item` per line allowed.
STEP	defines the step width that is scrolled with one arrow click.
STEP = 0	no scrolling is possible (the arrow buttons are not created)
ORIENTATION	The tables default orientation is **HORIZONTAL**.
NAVIGATION	determines the direction in which the tables text fields are to be traversed during keyboard navigation. **HORIZONTAL** is the default value.
POSITION= n	defines which element is shown multiple times. n = 2 would show the first two elements (0 and 1) once and the third element (index 2) multiple times. **1** is the default value.
RANGE	determines the index range that can be used for the associated data items. The default starting index is 0.
MENU=**HIDDEN**	Do not display table row/column menu, when right mouse button is pressed at row/column header. Is used in conjunction with option **TABLESIZE**.
ARROWS=**HIDDEN**	Do not display table row/column header. Is used in conjunction with option **TABLESIZE**.
FRAME	Display a frame around the fieldgroup
MARGIN	Set margin (and optionally spacing) in pixels
INDEX=ID_INDEX	Use an **INDEX** object (see section Index-Object on page 95) as the **INDEX** of the **FIELDGROUP**. This makes it possible to set and read the value of the **FIELDGROUP INDEX**. Is used in conjunction with option **TABLESIZE**.
CLASS	sets **CLASS** property of the **FIELDGROUP** (used in qt style sheets)
ACCORDION	displays the **FIELDGROUP** with a header and a body. It has a toggle button, that expands or collapses the body. The button label uses the title of the **FIELDGROUP**.
ACCORDION = **OPEN**	The body part of the **FIELDGROUP** is shown at start
PIXMAP	sets the individual image on the left of the **ACCORDION** button.
PIXMAP_OPEN	sets the individual image for an expanded **FIELDGROUP**.
PIXMAP_CLOSE	sets the individual image for a collapsed **FIELDGROUP**.

FUNC	Identifier of previously defined function, which will be executed by selecting the **FIELDGROUP**.
OVERLAY	allows placing a **FIELDGROUP** anywhere over its parent **FIELDGROUP**, similar to CSS `position: absolute`. The element does not occupy layout space. The first two numbers define pixel coordinates relative to the parent's top-left corner; negative values are measured from the opposite edges (right/bottom). The last two numbers define width and height. One must call the **MAP** function for a **FIELDGROUP** with **OVERLAY**; otherwise, it will not be visible.

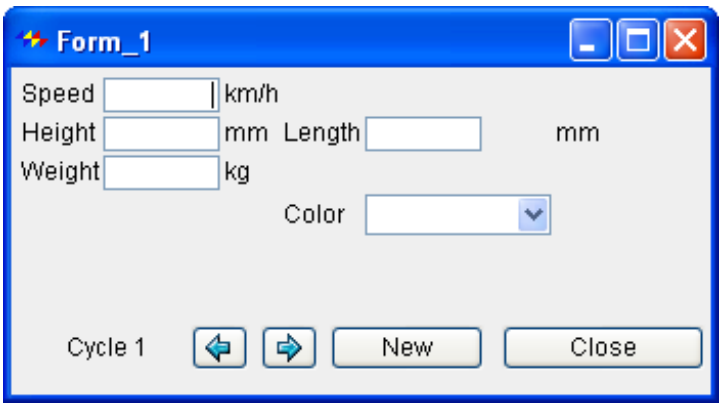


Figure 6: Example without FIELDGROUP options

Fieldgroup Examples The following examples show various configuration options of the fieldgroup.

The configuration 'Example 1' uses no fieldgroup options. **INTENS** places all fields left justified (see page 81).

```
DESCRIPTION "Example 1";
DATAPOOL
  SET
     Colors ("red" = 1,"green" = 2,"blue" = 3)
    ;
  REAL {EDITABLE}
    speed ,height ,length ,weight
  ;
  INTEGER {EDITABLE}
    color {SET = Colors}
  ;
END DATAPOOL;

UI_MANAGER
  FIELDGROUP
    Group_1
      ( "Speed"    speed   "km/h"
      , "Height"   height  "mm"   "Length"   length  "mm"
      , "Weight"   weight  "kg"
      , VOID       VOID    VOID    "Color"    color
      )
   ;
  FORM
    Form_1
      ( ( Group_1 )
      )
   ;
END UI_MANAGER;
END.
```

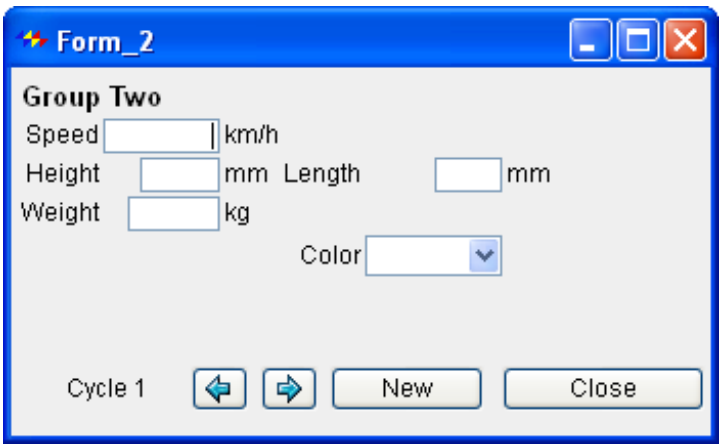


Figure 7: Example with option **JUSTRIGHT** and fieldgroup title

The second example uses the option **JUSTRIGHT** and sets the fieldgroup title “Group Two”. All fields except those with a following ’<’ character are right justified within the fieldgroup. (see page 82).

```
DESCRIPTION "Example 2";
DATAPOOL
  SET
    color_set ("red" = 1, "green" = 2, "blue" = 3 )
  ;
  REAL {EDITABLE}
    speed, height, length, weight
  ;
  INTEGER {EDITABLE}
    color { LABEL="Color:", SET=color_set }
  ;
END DATAPOOL;

UI_MANAGER
  FIELDGROUP
    Group_2 {"Group Two", JUSTRIGHT} (
      "Speed:">  speed*3.6>       "km/h"<
    , "Height:"> height:5:1>      "m"<    "Length:"> length:4> "m"<
    , "Weight:"> weight*1e-3:6:2> "kg"<   VOID       VOID      VOID
    , VOID       VOID             VOID   "Color:">  color:5>  VOID
    )
  ;
  FORM
    Form_2
      ( ( Group_2 )
      )
   ;
END UI_MANAGER;
END.
```

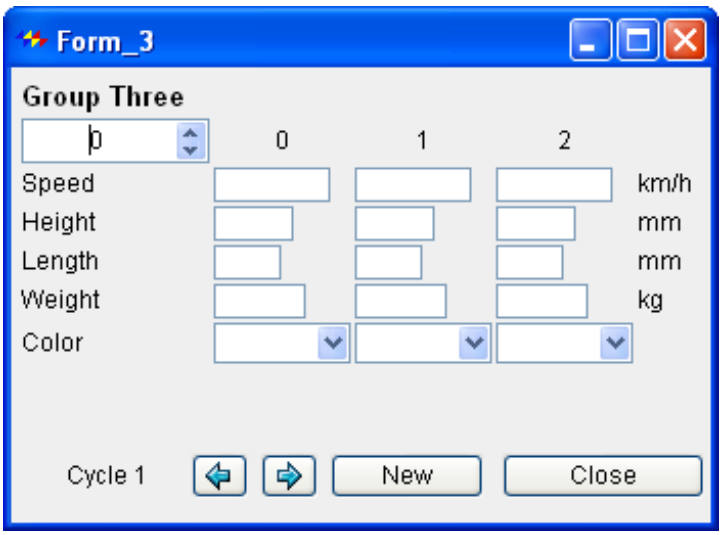


Figure 8: example with the option TABLESIZE=3

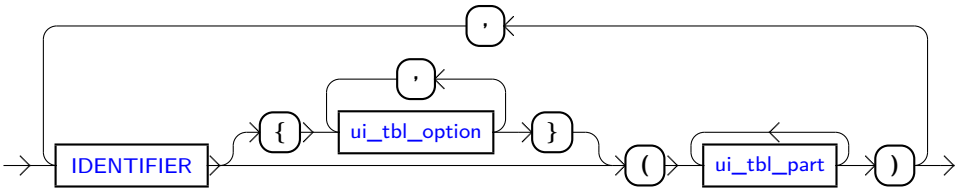
The third example shows data items arranged as a table. This is done with the option **TABLESIZE**.

```
DESCRIPTION "Example 3";
DATAPOOL
  SET
     Colors ("red" = 1,"green" = 2,"blue" = 3)
    ;
  REAL {EDITABLE}
    speed ,height ,length ,weight
  ;
  INTEGER {EDITABLE}
    color {SET = Colors}
  ;
END DATAPOOL;

UI_MANAGER
  FIELDGROUP
    Group_3 {"Group Three", TABLESIZE = 3, STEP 2}
      ( "Speed"    speed       "km/h"<
      , "Height"   height:5:1  "mm"<
      , "Length"   length:4    "mm"<
      , "Weight"   weight:6:2  "kg"<
      , "Color"    color:5
      )
   ;
  FORM
    Form_3
      ( ( Group_3 )
      )
   ;
END UI_MANAGER;
END.
```

4.6.4 Table

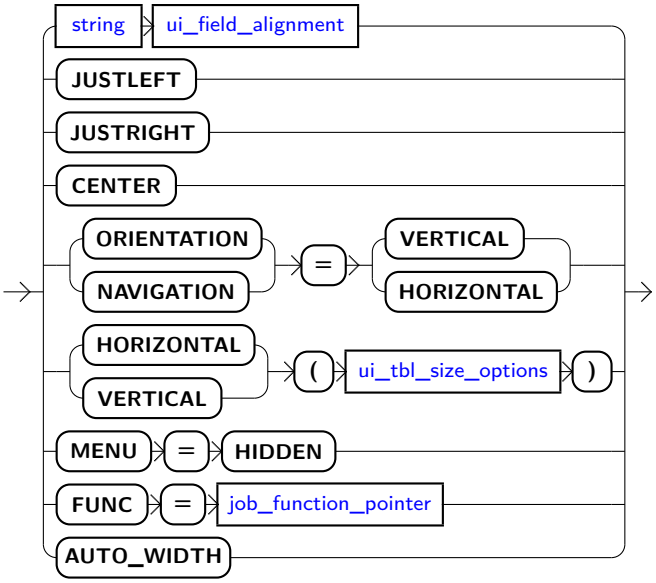
The user interface element **TABLE** is a multi-purpose widget for displaying and editing matrices and lists.

ui_table_list



The table options define the behaviour and appearance of the table.

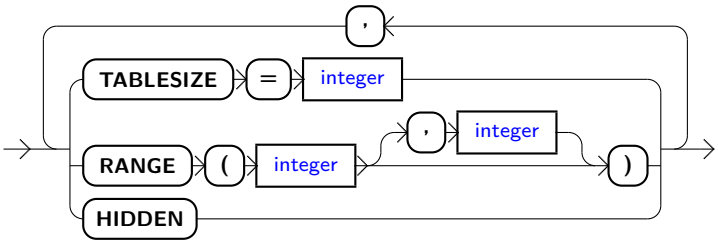
ui_tbl_option



option	description
`string`	The title of the table
JUSTLEFT	TODO
JUSTRIGHT	TODO
CENTER	Without one of these options, **INTENS** places the table fields left justified within each column.
ORIENTATION	The tables default orientation is horizontal.
NAVIGATION	determines the direction in which the tables text fields are to be traversed during keyboard navigation. **HORIZONTAL** is the default value.
HORIZONTAL	horizontal size and range

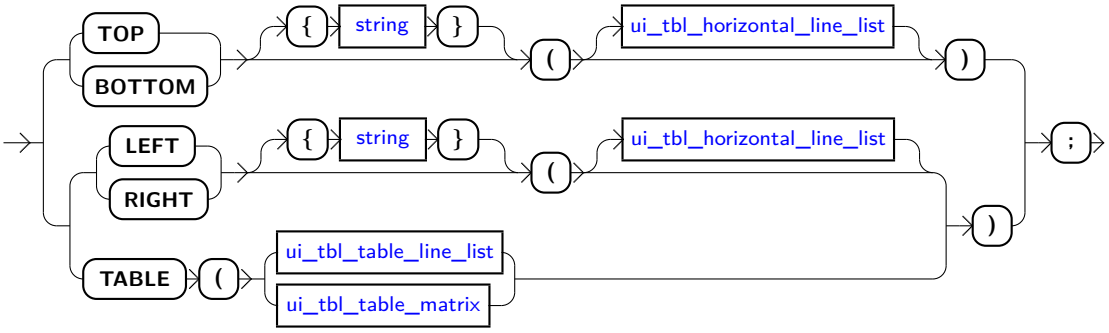
VERTICAL	vertical size and range
MENU = **HIDDEN**	Do not display table row/column menu, when right mouse button is pressed at row/column header. (See section row/column menu on page 90.)
FUNC	function to call by mouse click or enter key The function can use a Reason (see page 234) to distinguish select, unselect, ...
AUTO_WIDTH	tell webtens to determine the column widths automatically, based on the content

ui_tbl_size_options



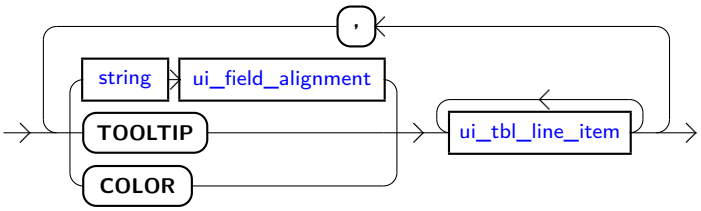
size options	description
TABLESIZE	defines the minimal number of rows (**VERTICAL**) or columns (**HORIZONTAL**) shown in the **TABLE** part. **TOP** and **BOTTOM** rows are shown in addition. **LEFT** and **RIGHT** columns are shown in addition.
RANGE	determines the index range that can be used for the associated data items. The default starting index is 0. The first number defines the number shown for index 0. It is the offset of the number shown on the gui. The optional second index is the maximal number shown. The second index also defines the maximal number of data items shown. When that number is less than or equal **TABLESIZE**, the **SCROLLBAR** is not shown.
HIDDEN	**HORIZONTAL** or **VERTICAL** headers are not shown

ui_tbl_part

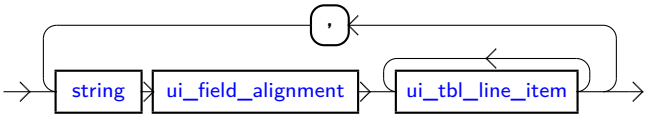


option	description
TOP	positions following lines at the top of the table.
BOTTOM	positions following lines at the bottom of the table.
LEFT	positions following lines to the left of the table.
RIGHT	positions following lines to the right of the table.
TABLE	specifies the table entries (data displayed in the table)
`title string`	The title of the specified side of the table

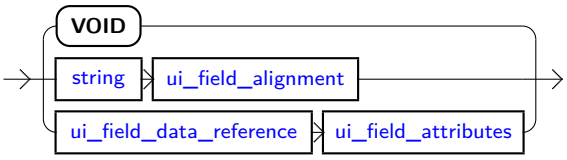
ui_tbl_table_line_list



ui_tbl_horizontal_line_list



ui_tbl_line_item



ui_tbl_table_matrix

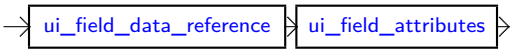


table items	description
VOID	creates an empty space.
TOOLTIP	The values of this row/column are not shown but used as the tooltips.
COLOR	The values of this row/column are not shown. The colors corresponding to their values (see section Data ColorSet on page 47) are used for the column/row.
`string`	A label string. Defines the colum/row header.
`alignment`	(see section ui_field_alignment on page 74)
`data reference`	(see section ui_field_data_reference on page 69)
`field attributes`	(see section ui_field_attributes on page 76) To hide a column, provide a width of 0 Hiding a column is useful when table row/column menu (right mouse button) are used. The menu functions delete, insert, clear etc. all fields in the table. This includes hidden columns.

The following example shows the configuration of two tables and how they will be displayed by **INTENS** (see page 90)

```
TABLE
  Table1 { "Table 1"
         , HORIZONTAL ( TABLESIZE = 5  )
         , VERTICAL   ( TABLESIZE = 23 )
         }
  ( TOP   ( "Length"  length[0:*,0]:10
          , "Weight"  weight[0:*,1]:10
          );
    LEFT  ( "Speed"   speed[0:*]:10     );
    TABLE ( rslt[*,*]:10 );
  )
, Table2 { "Table 2"
         , HORIZONTAL ( TABLESIZE = 3 )
         , ORIENTATION = HORIZONTAL
         , NAVIGATION  = HORIZONTAL
         }
  ( TOP ( "Color"   color[0,*]:-20     );
    TABLE ( "first"   first [0,*]:-10
          , "second"  second[1,*]:10
          , "third"   third [2,*]:10   );
  )
;
```

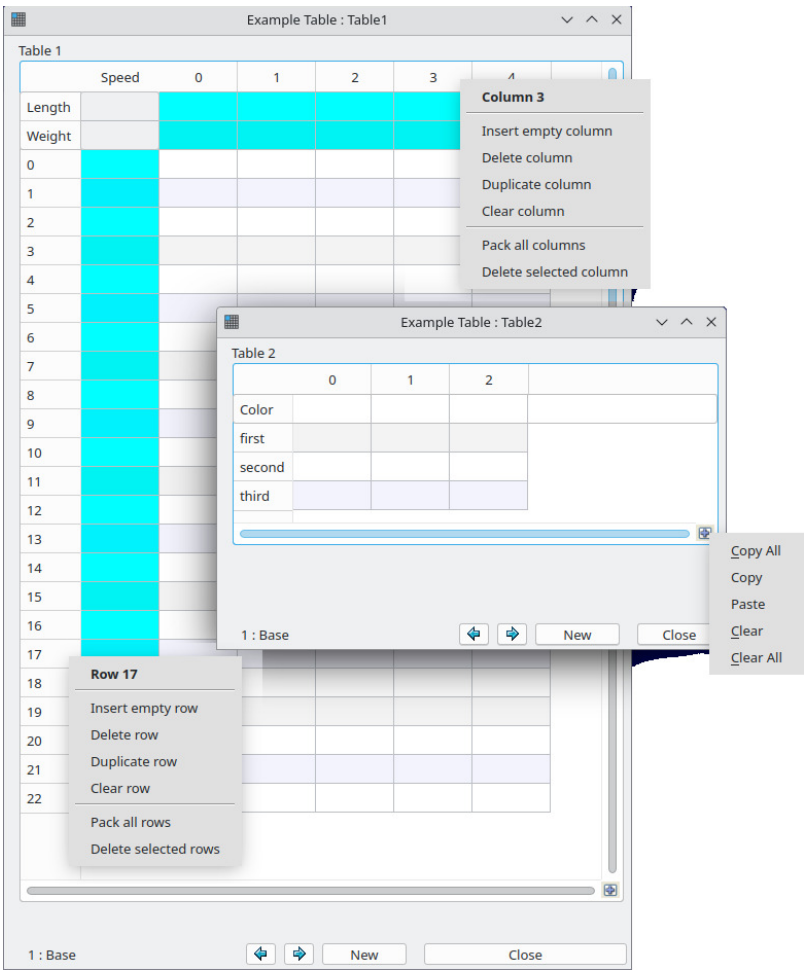


Figure 9: example of **TABLE**

TABLE menus **INTENS TABLE**s have two kind of right click menus:

row/column menu These menus are shown when the right mouse button is pressed on a row or column header. The menu header shows the type of menu (Row or Column) and number of the row/column the menu was opened for. The first four entries depend on that row/column number.

row menu entries	description
Insert empty row	An empty row is inserted. This shifts down the values of this and all following rows.
Delete row	The row is deleted. This shifts up the values of all following rows.
Duplicate row	A new row with the values of this row is inserted. This shifts down the values of this and all following rows.
Clear row	The values of this row are deleted, but the row is kept.
Pack rows	Values are shifted up to fill empty cells.

Delete selected rows	This is independent of the row the menu is shown for. It deletes the rows of all selected table cells. Following rows are shifted up.

column menu entries	description
Insert empty column	An empty column is inserted. This shifts right the values of this and all following columns.
Delete column	The column is deleted. This shifts left the values of all following columns.
Duplicate column	A new column with the values of this column is inserted. This shifts right the values of this and all following columns.
Clear column	The values of this column are deleted, but the column is kept.
Pack columns	Values are shifted left to fill empty cells.
Delete selected columns	This is independent of the column the menu is shown for. It deletes the columns of all selected table cells. Following columns are shifted left.

bottom right menu This menu is shown when the right mouse button is pressed on the cross at bottom right corner of the table.

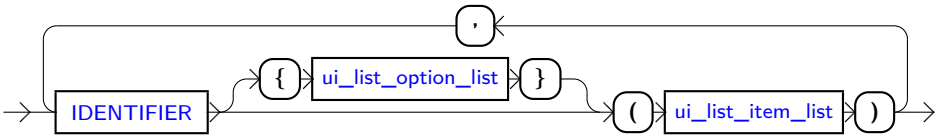
corner menu entries	description
Copy All	All values are copied to the clipboard, i.E. to be pasted into a spreadsheet or another **INTENSTABLE**.
Copy	The selected values are copied, i.E. to be pasted into a spreadsheet or another **INTENSTABLE**.
Paste	Previously copied values are pasted to the table, starting at the selected cell.
Clear	The selected values are deleted.
Clear All	All values are deleted.

Paste and Clear operations are done cell by cell, as if the user would change the values. Therefore, the variables functions (if any) is called with **REASON_INPUT**, **INPUT**, **OLDVALUE**, **INDEX**, **NODE**, **THIS**, etc. available inside the functions.

4.6.5 List

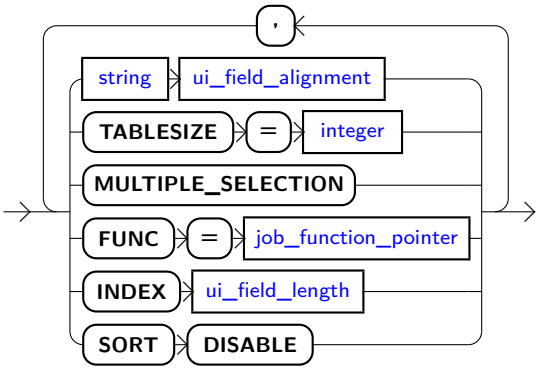
The user interface element **LIST** is a multi-purpose widget for displaying lists.

ui_list_list



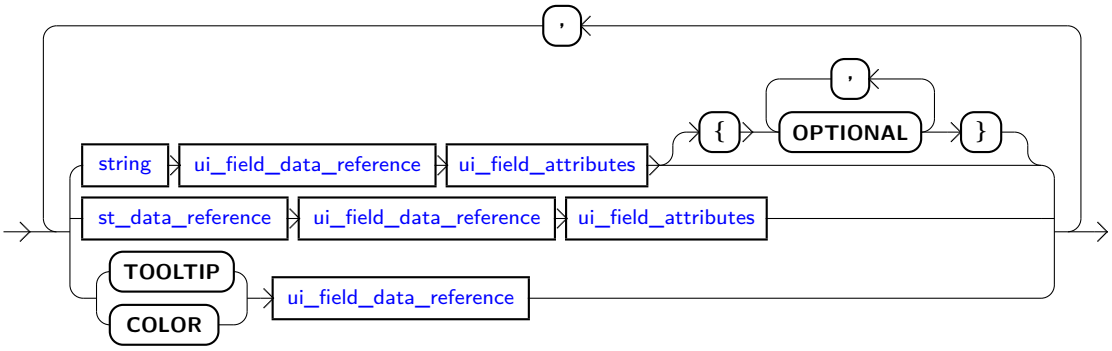
The list options define the behaviour and appearance of the list.

ui_list_option_list



list options	description
`string`	the title of the list
`alignment`	(see section ui_field_alignment on page 74)
TABLESIZE	defines size of displayed rows
MULTIPLE_SELECTION	enable the selection of multiple rows (see function statement **GET_SELECTION** on page 205).
FUNC	function to call by mouse click or enter key The function can use a Reason (see page 234) to distinguish select, unselect, activate.
INDEX	displays row numbers
SORT DISABLE	disable the possibility to click on a column header to sort the list by this column

ui_list_item_list



list items	description
`string`	A label string. Defines the column header.
TOOLTIP	The values of this column are not shown but used as the tooltips.
COLOR	The values of this column are not shown. The colors corresponding to them (see section Data ColorSet on page 47) are used for the rows.
OPTIONAL	The column is hidden, but it can be shown using the 'Column Configuration', which can be opened in the right click menu.
`data reference`	(see section ui_field_data_reference on page 69)
`field attributes`	(see section ui_field_attributes on page 76)

The following example shows the configuration of a list and how it will be displayed by **INTENS** (see page 94)

```
LIST
  list_identifier { "List"|, TABLESIZE = 10, INDEX } (
    "Label 1" data_item_1[*] :30,
    "Label 2" data_item_2[*] :10
  )
;
```

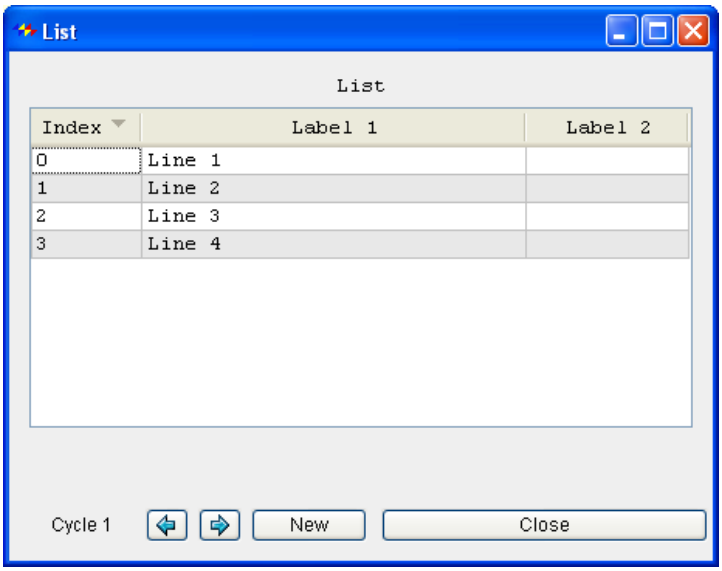


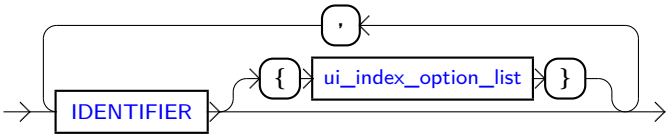
Figure 10: example of List

4.6.6 Index-Object

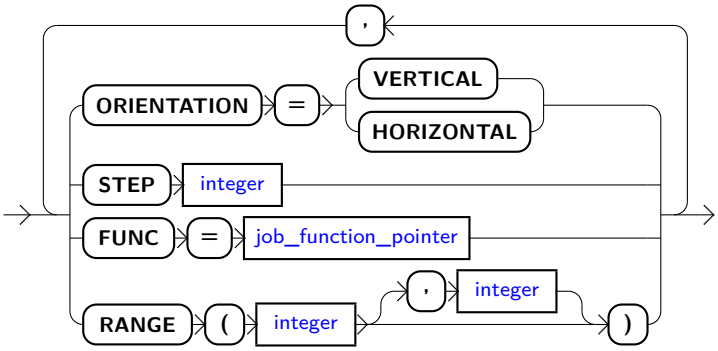
Figure 11: Index-Object

An index-object is a GUI-Object which may be invisible or placed in a fieldgroup. It can be used as a variable index. (see section Data Variables on page 69).
The index-object displays like a scrollbar. Its arrows let you navigate through the associated data items by mouse click.

ui_index_list



ui_index_option_list



index options	description
ORIENTATION	The index-object-arrows are shown:
	VERTICAL(△ ▽)
	HORIZONTAL(◁ ▷)
STEP	defines the step for each arrow-pressing (back and forward)
FUNC	calls a function defined in section FUNCTIONS page 189.
RANGE	determines the index range that can be used for the associated data items. The default starting index is 0.

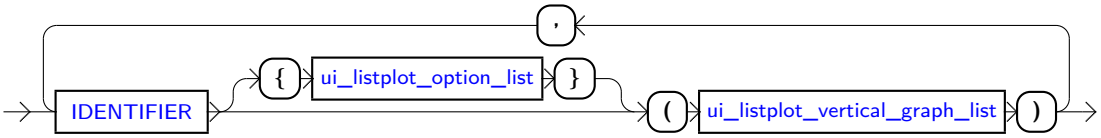
Examples:

```
DESCRIPTION "Example INDEX";
DATAPOOL
  REAL {EDITABLE}
    a
   ;
END DATAPOOL;
  ...
UI_MANAGER
  INDEX
    index_a { RANGE(1,10) }
  ;
  FIELDGROUP
    fieldgroup_a (
      "Variable a"  a[index_a]:10
     ,"Index of a"  index_a
    );
END UI_MANAGER;
END.
```

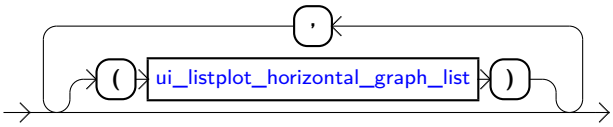
4.6.7 Listplot

List plots can be used to display line curves with up to 8 differently scaled axes in a common plot window.

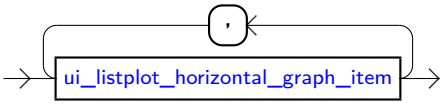
ui_listplot_list



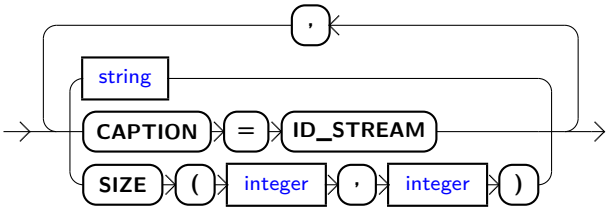
ui_listplot_vertical_graph_list



ui_listplot_horizontal_graph_list

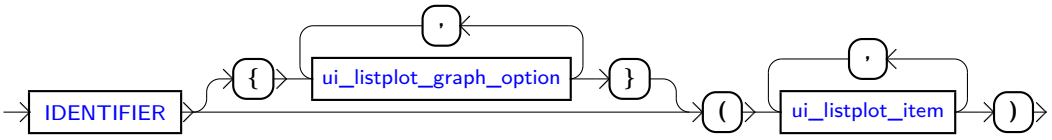


ui_listplot_option_list

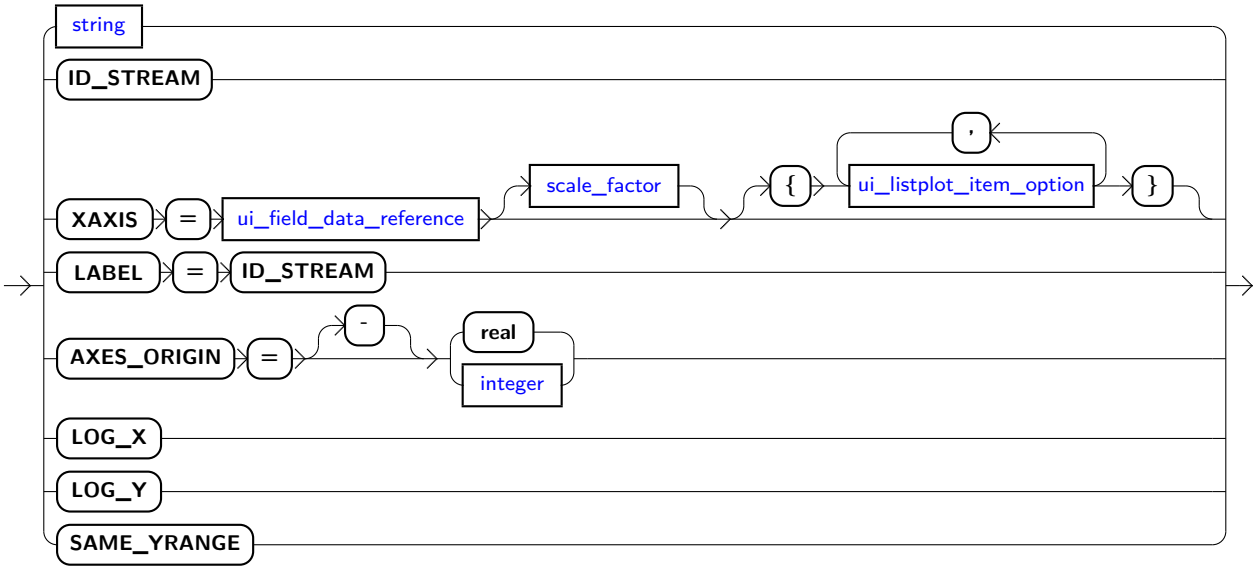


options	description
`string`	defines the label string for the print menu button.
CAPTION	defines the caption text that will be printed in the right most column of the plot. The identifier must be a previously declared stream. (see section STREAMER on page 52)
SIZE	x, y in pixel

ui_listplot_horizontal_graph_item

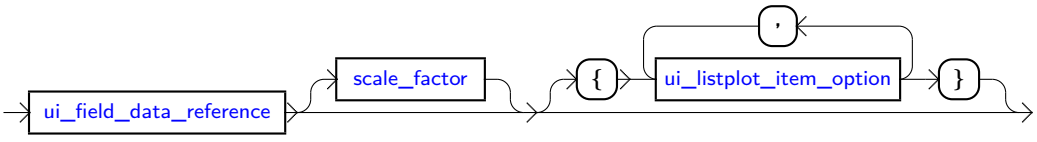


ui_listplot_graph_option



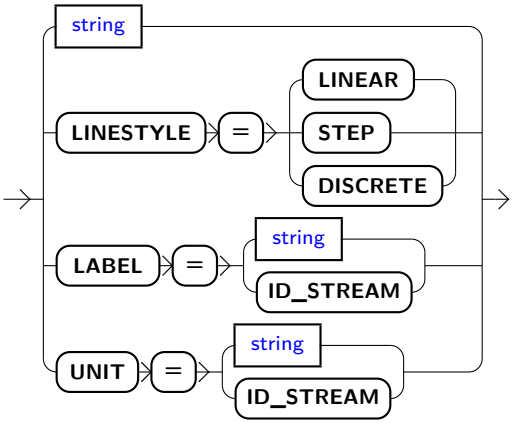
graph options	description
`string`	defines the graph title.
`identifier`	defines the graph title that will be printed at the top of the plot. The identifier must be a previously declared stream. (see section STREAMER on page 52)
XAXIS	defines the data item whose values give the x-coordinates
AXES_ORIGIN	defines the origin for all y-values
LOG_X	the x-axis is scaled logarithmically
LOG_Y	the y-axis is scaled logarithmically
SAME_YRANGE	sets all y-axes to the the same range
LABEL	TODO

ui_listplot_item



plot item	description
`identifier`	references the data to be plotted (data item, stream-id).
`scale`	a scale factor (see section Scale Factors page 20).

ui_listplot_item_option



plot item options	description
`string`	defines the unit string.
`identifier`	streamer-identifier.
LINESTYLE	defines the line style
LINEAR	the points are connected by straight lines
STEP	the points are connected with horizontal and vertical lines
DISCRETE	the points are not connected.
LABEL	string or stream identifier
UNIT	string or stream identifier

The following example shows the configuration of a 2D-plot and how it will be displayed by **INTENS** (see page 100)

```
LISTPLOT
  curve_plot{ "Plot" } (
     ( magn_curve {"Magnetizing curve", XAXIS=iMag}
       ( psiMag{"[Vs]"} ),
       x_i2  {"Rotor current", XAXIS=speed*60{"1/min"} }
       (i2{"[A]"} )
     ),
     ( x_torque{"Induction Motor", XAXIS=speed*60{"1/min"} }
       ( torque*1e-3{"[kNm]"}, i1{"[A]"}, cos )
     )
  );
FORM
  curve_form (
     ( curve_plot )
  );
```

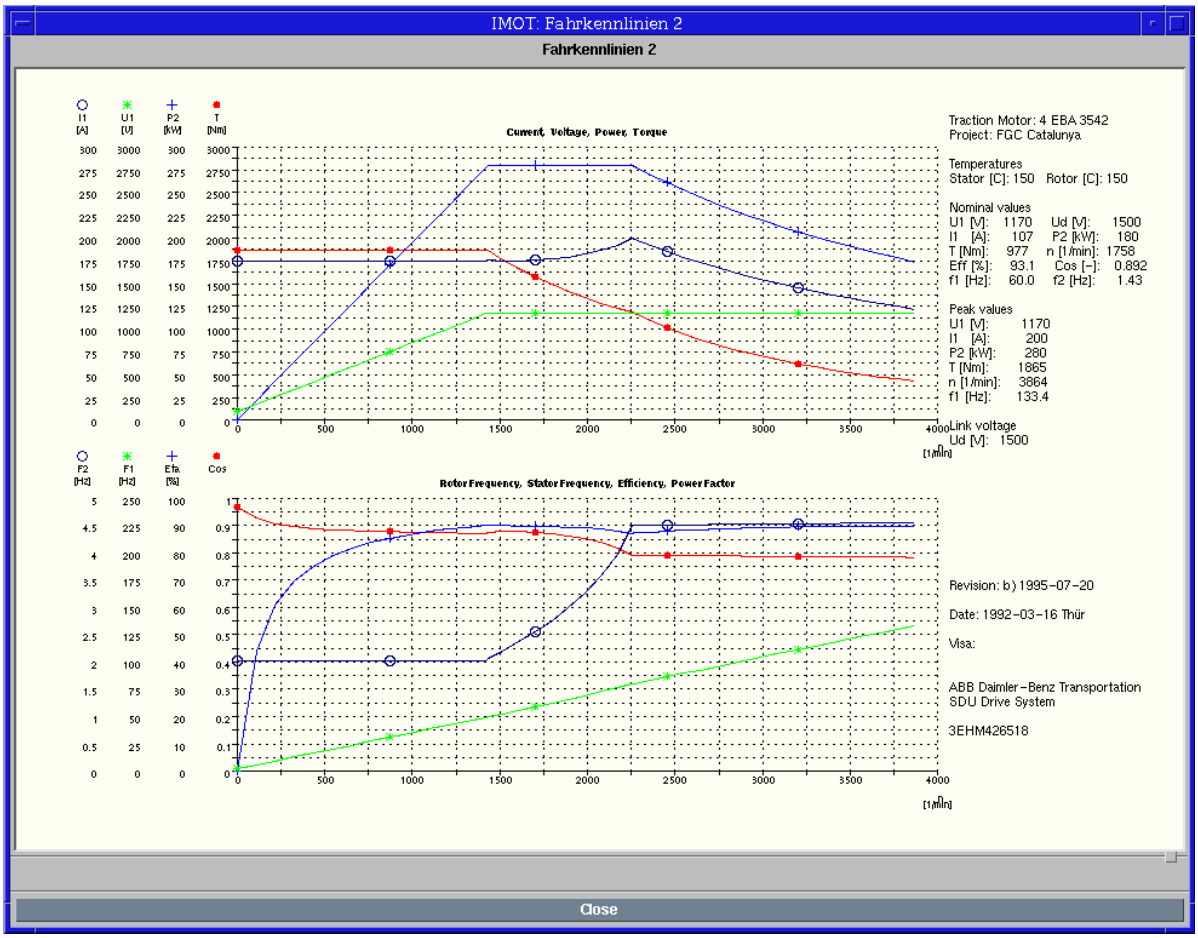
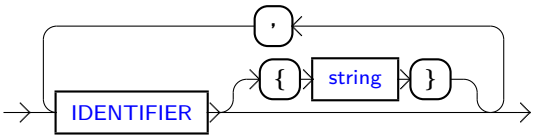


Figure 12: **LISTPLOT** diagram with 2 plots

4.6.8 Uniplot

UNIPLOT is a ABB standard plot format with a special set of plot vectors. It is mainly used within existing Fortran programs. An example is given in figure example of a UNIPLOT diagram.

ui_uniplot_list




```
DESCRIPTION "Example UNIPLOT";
  ... STUFF HERE ...
UI_MANAGER
  UNIPLOT
    mech_plot{ "Plot Mech" };
  FORM
    Form_Uniplot {"Plot Mech", HELPKEY "Mech_Plot", HIDECYCLE}
      ( ( mech_plot ) );
END UI_MANAGER;

OPERATOR
  PROCESS  plot_mech_proc : BATCH {"plotproc"};
  PROCESSGROUP
    mech_prog {"Mech"}(output_stream {DISPLAY=NONE},
                       mech_plot[mechuni] = plot_mech_proc( input_stream );
                      );
END OPERATOR;
  ... STUFF HERE ...
END.
```

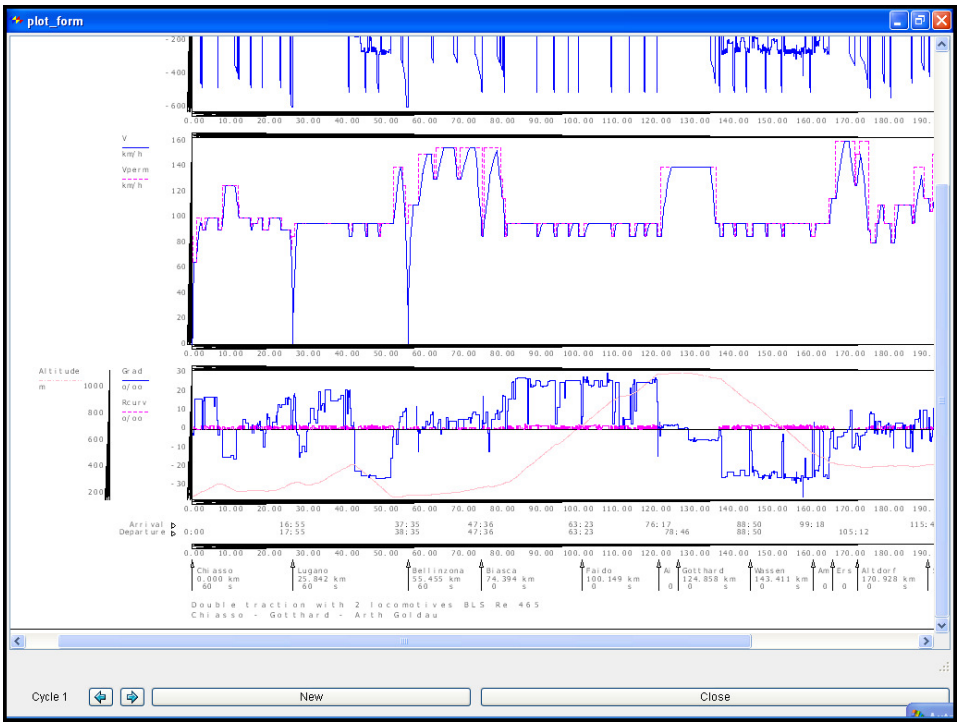
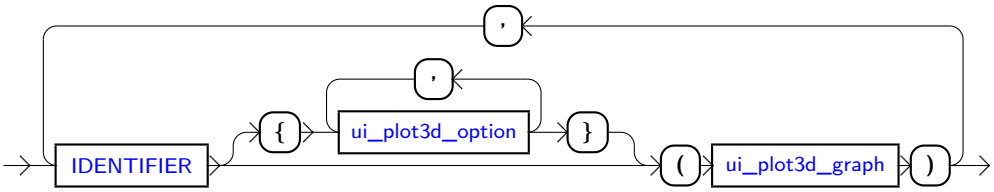


Figure 13: example of a UNIPLOT diagram

4.6.9 Plot3D

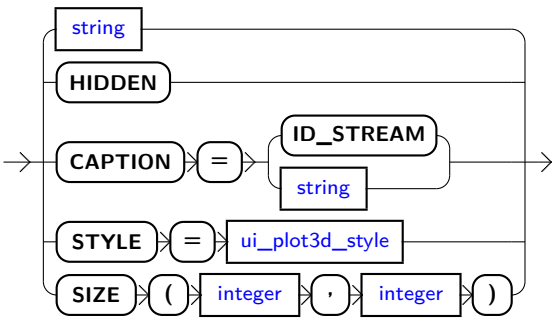
Plots of type **PLOT3D** can be used to produce 3-dimensional graphs of datapool values which have 2 dimensions.

ui_plot3d_list



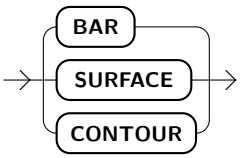
Pressing the right mouse button within such a diagram pops up a menu which provides additional configuration functions.

ui_plot3d_option



options	description
`string`	defines the label string for the print menu button.
CAPTION	defines the caption text that will be printed at the bottom of the plot. The identifier must be a previously declared stream. (see section STREAMER on page 52)
STYLE	defines the initial style of the plot
HIDDEN	don't add the plot to the **File Print** menu
SIZE	initial size in pixel

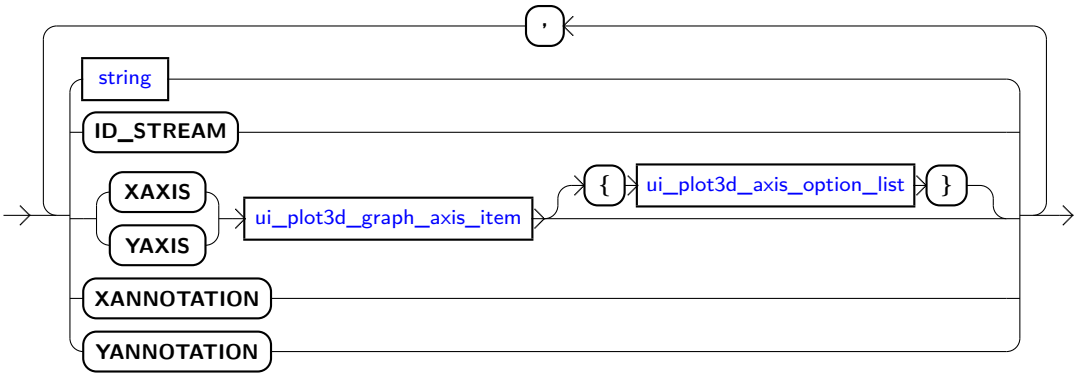
ui_plot3d_style



style	description
BAR	Draws bars
SURFACE	Fills in a color for each value.
CONTOUR	Does not fill in any color.

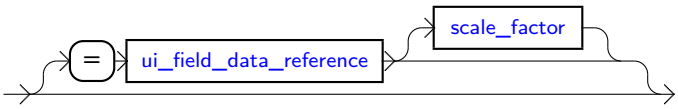
ui_plot3d_graph

ui_plot3d_graph_option_list

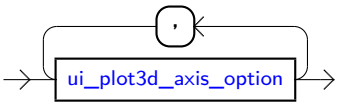


graph options	description
`string`	defines the plot3d graph title that will be printed at the top of the plot.
`identifier`	defines the plot3d graph title that will be printed at the top of the plot. The identifier must be a previously declared stream. (see section STREAMER on page 52)
XAXIS	defines the x-coordinates
YAXIS	defines the y-coordinates
XANNOTATION	Shows x-axis as defined in plot3d-axis option **ANNOTATION** on page 105.
YANNOTATION	Shows y-axis as defined in plot3d-axis option **ANNOTATION** on page 105.

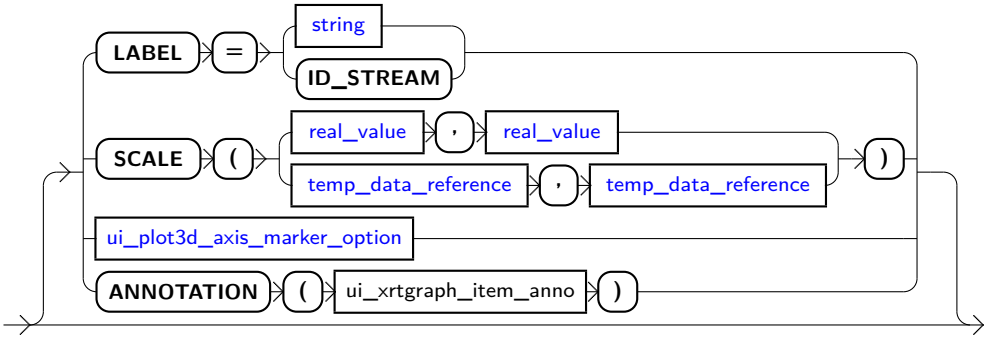
ui_plot3d_graph_axis_item



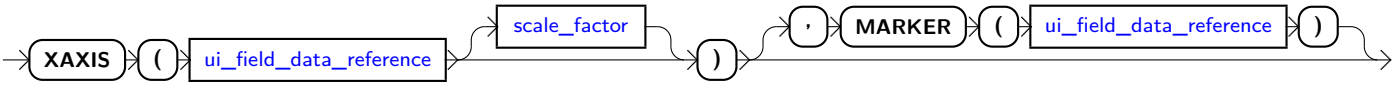
ui_plot3d_axis_option_list



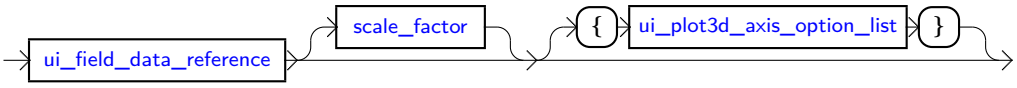
ui_plot3d_axis_option



ui_plot3d_axis_marker_option



ui_plot3d_item



plot item option	description
`scale`	see section Scale Factors page 20.
LABEL	defines the label that will be printed at the top of the axis
SCALE	axis scale (min, max)
ANNOTATION	defines extra labels for the axis (see page **??**).
MARKER	show markers with optional labels

```
DESCRIPTION "Example 3D PLOT";
DATAPOOL
  REAL {EDITABLE}
    matrix
   ,xValues
   ,yValues
   ;
END DATAPOOL;

UI_MANAGER
  PLOT3D
    plot_3d
      { CAPTION = "streamCaption" }
      ( plot
           { "SIN(x) * COS(y)"
           , XAXIS = xValues { LABEL = "X-Axis" }
           , YAXIS = yValues { LABEL = "Y-Axis" }
           }
         ( matrix * 0.01 { LABEL = "Matrix Values" }
         )
      );
  FORM
    Form_Plot3D {"3D Plot", HIDE_CYCLE}
      ( ( plot_3d )
      );
END UI_MANAGER;
END.
```

PLOT3D Plot styles

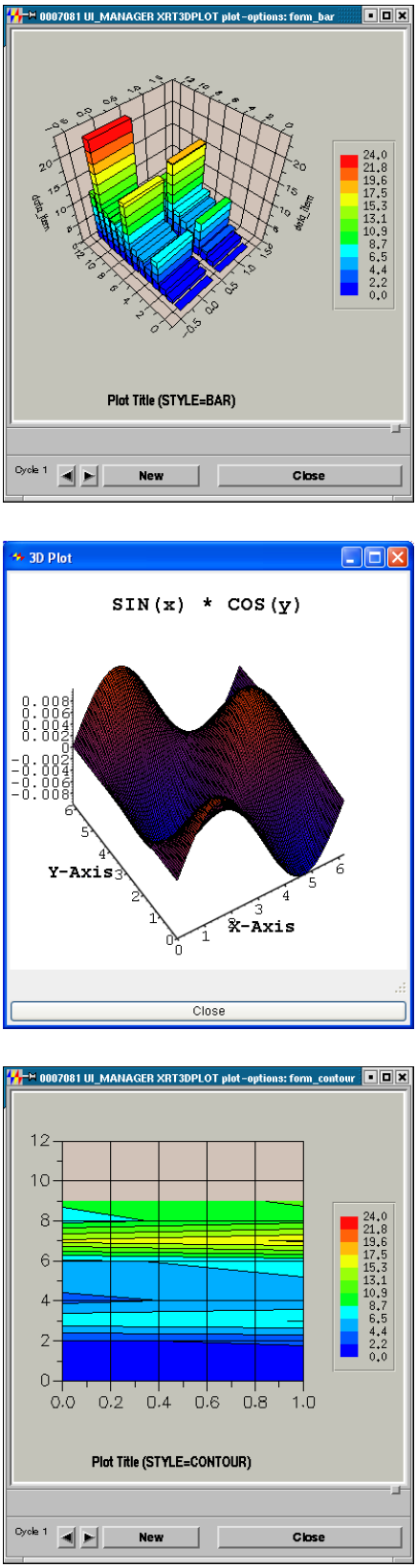
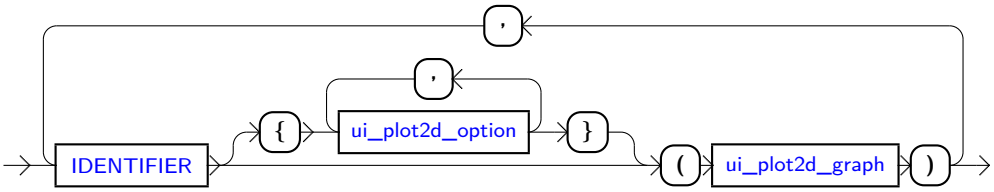


Figure 14: PLOT3D Plot styles

4.6.10 Plot2d

Plots of type **PLOT2D** can be used to produce 2-dimensional graphs of datapool values.

ui_plot2d_list



Pressing the right mouse button within such a diagram pops up a menu which provides additional configuration functions. See ui_plot2d_menu_entry_list on page 151.

UI Mode One option in that menu is the UI Mode. The UI Mode defines the behaviour of the left mouse button inside the plot2ds.

The mode can be changed using

- that menu
- keyboard shortcuts (when focus is on a **PLOT2D**)
- **DATAPOOL** variable **PLOT2D_UIMODE**

The first two ways automatically change the value of **PLOT2D_UIMODE**.

All **PLOT2D**s within one application share one global UI Mode. The following modes exist:

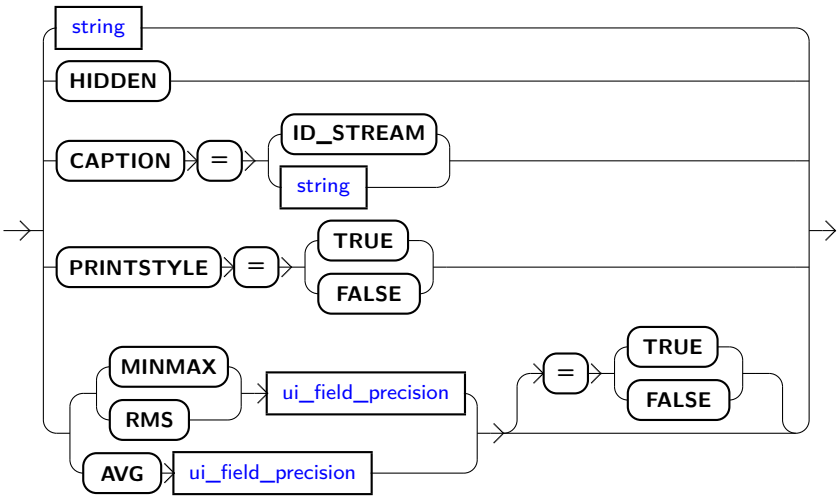
UI Mode	Shortcut (hint) **PLOT2D_UIMODE** value description
Zoom	Ctrl-S (scale) “ZOOM” Select a rectangle to zoom in.
Select Point	Ctrl-D (dot) “SELECT_POINT” Select a point (not a line) of a plot curve. If the plot2d has the **FUNC** option, that function is called and **REASON_SELECT_POINT** is set. **INDEX** is set to the index of the selected point. The function can access the coordinates of the point using the datapool variables Global_Point.X and Global_Point.Y. Without the **FUNC** option, you can select multiple points. Select the same point again to unselect it. Use **GET_SELECTION** in a function to get the information about the selected points. See gui_more_statement on page 205.

Select Rectangle	Ctrl-A (area) “SELECT_RECTANGLE” This mode is only available when the **FUNC** option is set. That function is called after selecting a rectangle and **REASON_SELECT_RECTANGLE** is set. The function can access the coordinates of the rectangle using the datapool variables Global_Rect.X1, Global_Rect.X2, Global_Rect.Y1 and Global_Rect.Y1.

Global Symbolsize The symbol sizes of **PLOT2D** curves are normally defined in the resource file. The user can change them using the option Configuration... in the right mouse button menu of a **PLOT2D**.

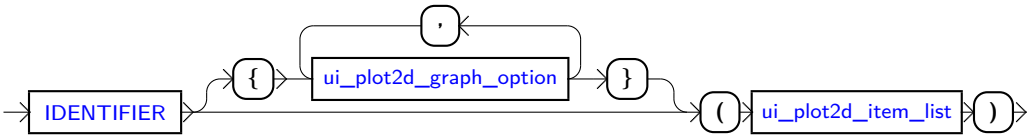
And the symbol size of all **PLOT2D** can be changed to the same value using the **DATAPOOL** variable **PLOT2D_SYMBOLSIZE**. When that variable has a value, it is used. When the value is cleared, the **PLOT2D**s use the configured symbol sizes again.

ui_plot2d_option

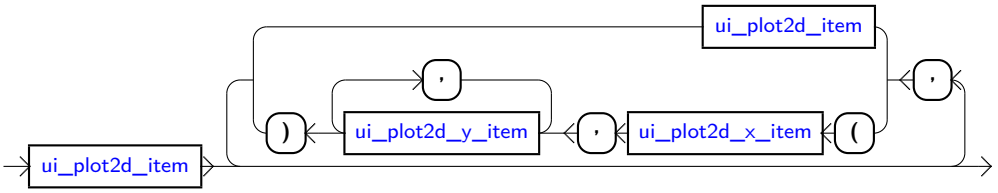


options	description
string	Menutext
CAPTION	Footertext: Title displayed in the graphics-form. (Using special-characters such as space, you need to use string-representation.) identifier may also refere to a previously declared stream.
PRINTSTYLE	= **TRUE** prints out graph descriptions. Use extra resources for printing (fonts, line styles, background etc.).
HIDDEN	don’t add the plot to the **File Print** menu
MINMAX	show(**TRUE**, default) or hide (**FALSE**) min and max of a selected curve.
RMS	show(**TRUE**, default) or hide (**FALSE**) rms of a selected curve.
AVG	show(**TRUE**, default) or hide (**FALSE**) avg and max of a selected curve.

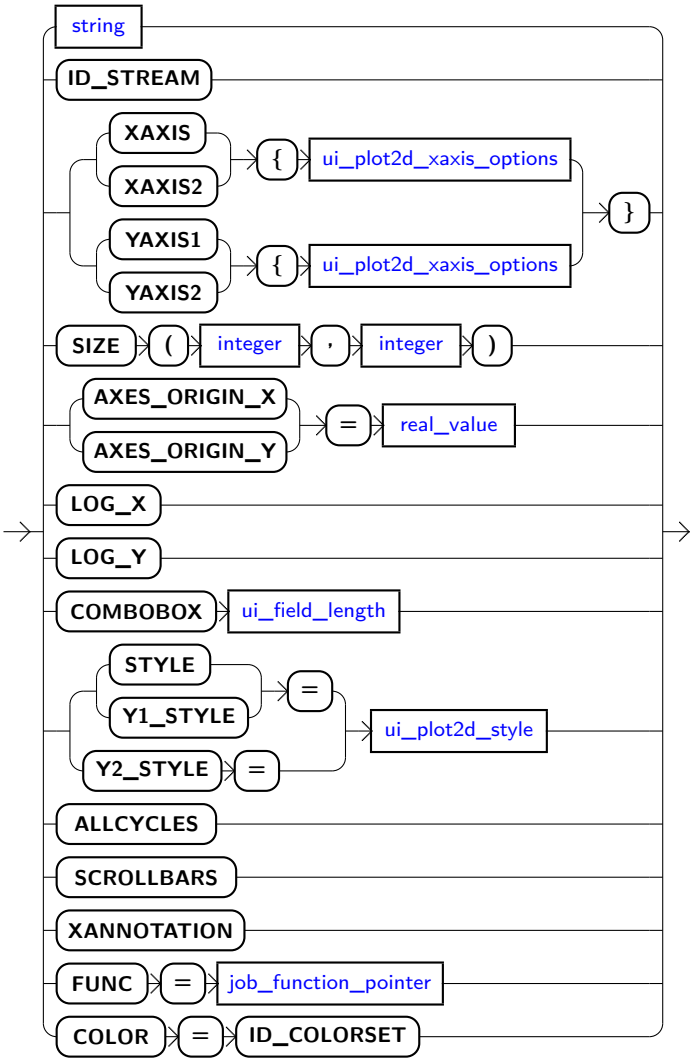
ui_plot2d_graph



ui_plot2d_item_list

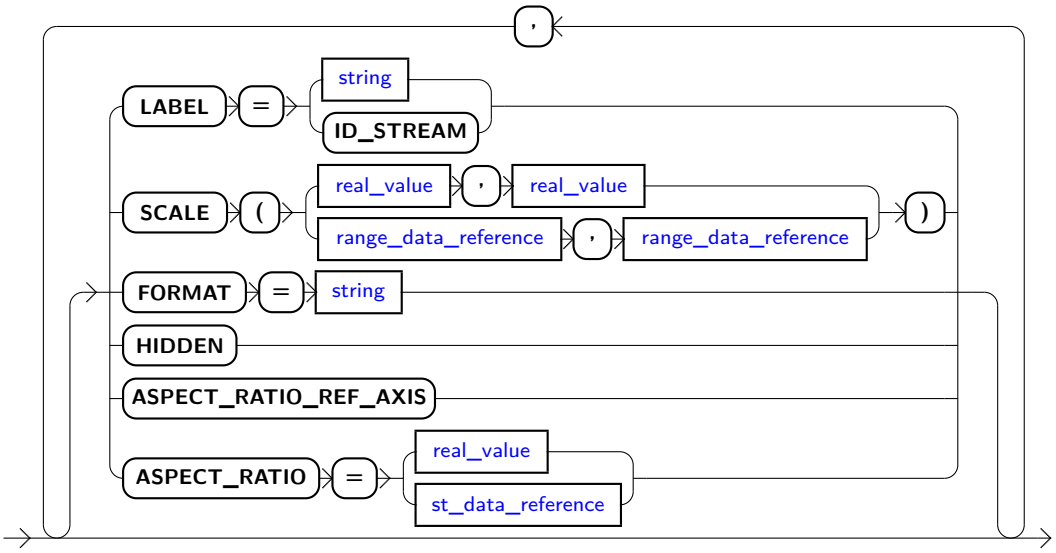


ui_plot2d_graph_option



2d graph options	description
`string`	defines the graph title that will be printed at the top of the plot.
ID_STREAM	defines the graph title that will be printed at the top of the plot. The identifier must be a previously declared stream. (see section STREAMER on page 52)
XAXIS{..}	describes the x-axis.
XAXIS2{..}	describes the x2-axis. This only draws an axis on top of the plot with the given **SCALE**. Without **SCALE**, this option does not make sense.
YAXIS1{..}	describes the y1-axis.
YAXIS2{..}	describes the y2-axis.
AXES_ORIGIN_X	defines the origin for the x-axis.
AXES_ORIGIN_Y	defines the origin for all y-axis.
LOG_X	the x-axis is scaled logarithmically.
LOG_Y	the y-axis is scaled logarithmically.
COMBOBOX	displays the item names as a combobox when the config menu is opened pressing the right mouse button over the plot.
STYLE	style of all curves.
Y1_STYLE	style of y1 curves.
Y2_STYLE	style of y2 curves.
ALLCYCLES	Shows data of each cylcle in same graph.
SCROLLBARS	Shows scroll-bars in 'zoomed zone' mode.
XANNOTATION	Shows x-axis as defined in x-graph-item option **XANNOTATION** on page 115.
FUNC	Defines the function that will be called when a point or rectangle is selected. See paragraph UI Mode in section Plot2d on page 108.
COLOR = ...	Defines the color set that will be used for the colors of the plot curves. See section Data ColorSet on page 47.

ui_plot2d_xaxis_options

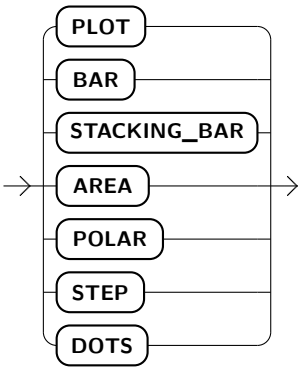


2d axis options	description
LABEL	axis label
SCALE	axis scale (min, max)
FORMAT	axis label number format: width[:precision] "10" : width of the number "10:2" : width and decimals of the number
HIDDEN	hide the axis labels
ASPECT_RATIO_REF_AXIS	This axis is the reference axis for calculating the unit scaling of the plot. The length (in pixel) and axis range of this axis is the reference for the other axis. The length of the other axis is adapted to get the desired unit scaling. Define **ASPECT_RATIO** for both axes. See example on page 119 and hardcopy example on page 121.

ASPECT_RATIO

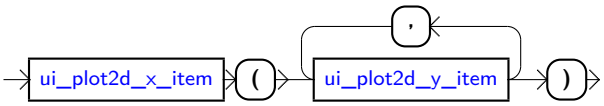
The meaning of this option differs for the reference axis (with **ASPECT_RATIO_REF_AXIS**) and the other axis:
Reference axis: Range of the reference axis relative to the other axis. As this axis is the reference axis, the range of the other axis is calculated as (range reference axis / this factor).
Example: Reference axis range: 0 .. 250 = 250.
2: range of the other axis is 250 / 2 = 125.
0.5: range of the other axis is 250 / 0.5 = 500.
Other axis: Defines the unit scaling (pixel per unit) of this axis relative to the reference axis.
Examples:
1: same unit scaling as the reference axis.
2: 1 unit on this axis needs twice the pixels as on the reference axis.
0.5: 1 unit on this axis needs half the pixels as on the reference axis.
See example on page 119 and hardcopy example on page 121.

ui_plot2d_style

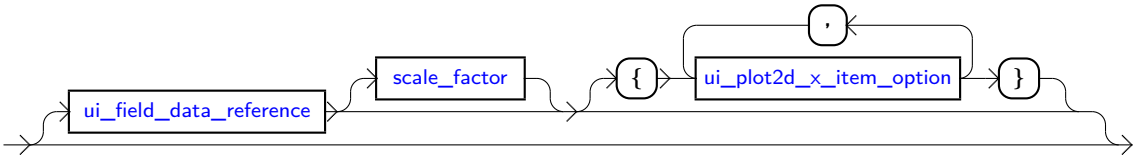


plot style	description
PLOT	see hardcopy example on page 118
BAR	see hardcopy example on page 118
STACKING_BAR	see hardcopy example on page 118
AREA	see hardcopy example on page 118
POLAR	see hardcopy example on page 118
STEP	see hardcopy example on page 118
DOTS	show a symbol at every data point

ui_plot2d_item

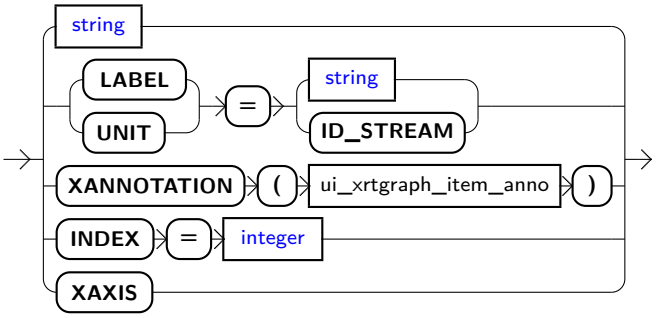


ui_plot2d_x_item



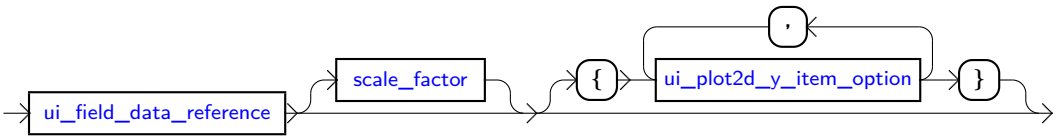
2d item list	description
`scale`	see section Scale Factors page 20.

ui_plot2d_x_item_option



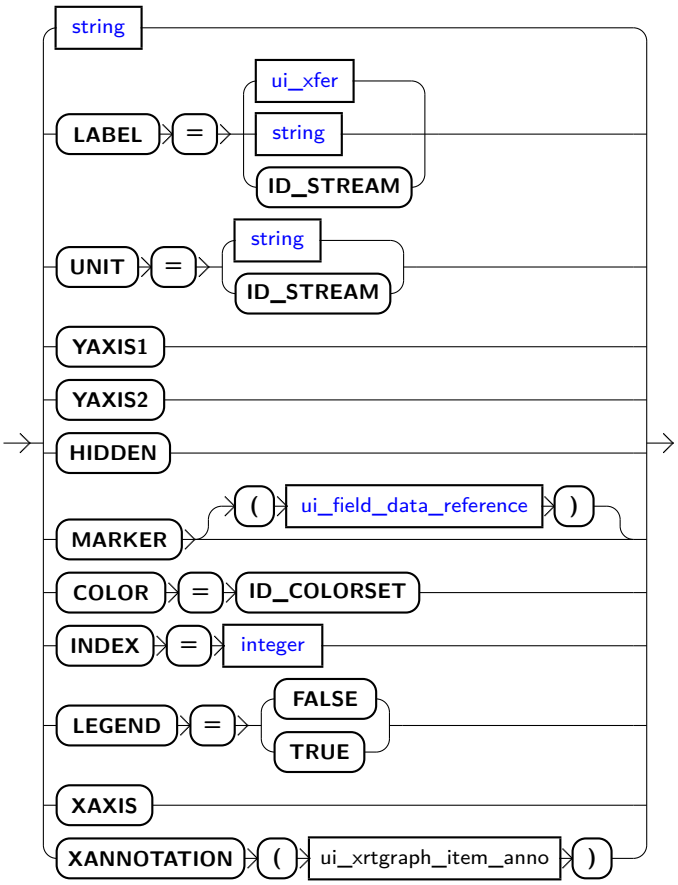
X-graph item option	description
`string`	title which will be printed to the right of x-axis description.
LABEL	defines the item name which will be printed as the x-axis description. use a **STREAM** when the name needs to be variable.
UNIT	defines the unit description which will be printed to the right of x-axis description.
XANNOTATION	defines extra labels for x axis (see page **??**).
INDEX	Data reference has two wildcards. One (defined here) is used to create many curves.
XAXIS	defines the x-coordinates of the x-axis.

ui_plot2d_y_item



y item list	description
`scale`	see section Scale Factors page 20.

ui_plot2d_y_item_option



Y-graph item option	description
`string`	title which will be printed at the top of the plot.
YAXIS1	defines the y-coordinates of the left y-axis. If this option is omited, the axis is not displayed at startup.
YAXIS2	defines the y-coordinates of the right y-axis. If this option is omited, the axis is not displayed at startup.
LABEL	defines the item name which is used in the plot legend. use a **STREAM** when the name needs to be variable.
UNIT	defines the unit description which will be printed at the top of the axis.
HIDDEN	does not display the axis.
MARKER	show markers with optional labels
COLOR = ...	use a color set (see section Data ColorSet on page 47) to show the markers in different colors. The provided marker labels are the values used to get a color from the color set.
INDEX	Data reference has two wildcards. One (defined here) is used to create many curves.

LEGEND	show (**TRUE**, default) or hide (**FALSE**) this curve in the legend
XAXIS	defines the x-coordinates of the x-axis.
XANNOTATION	defines extra labels for x axis (see page **??**).

```
DESCRIPTION "Example PLOT-2D";
DATAPOOL
  REAL {EDITABLE}
    xValues,
    y1Values,
    y2Values
  ;
END DATAPOOL;

STREAMER
  streamer_identifier ("Plot Title");
END STREAMER;

UI_MANAGER
  PLOT2D
    plot2d_identifier
    (
      graph_identifier {streamer_identifier}
      (
        xValues {LABEL="x-axis"}
        (
          y1Values {YAXIS1, LABEL="y1-axis"},
          y2Values {YAXIS2, LABEL="y2-axis"}
        )
      )
    )
  ;
  FORM
    form_identifier {"2D Plot", HIDE_CYCLE}
    (
      ( plot2d_identifier )
    )
  ;
END UI_MANAGER;
END.
```

Plot styles

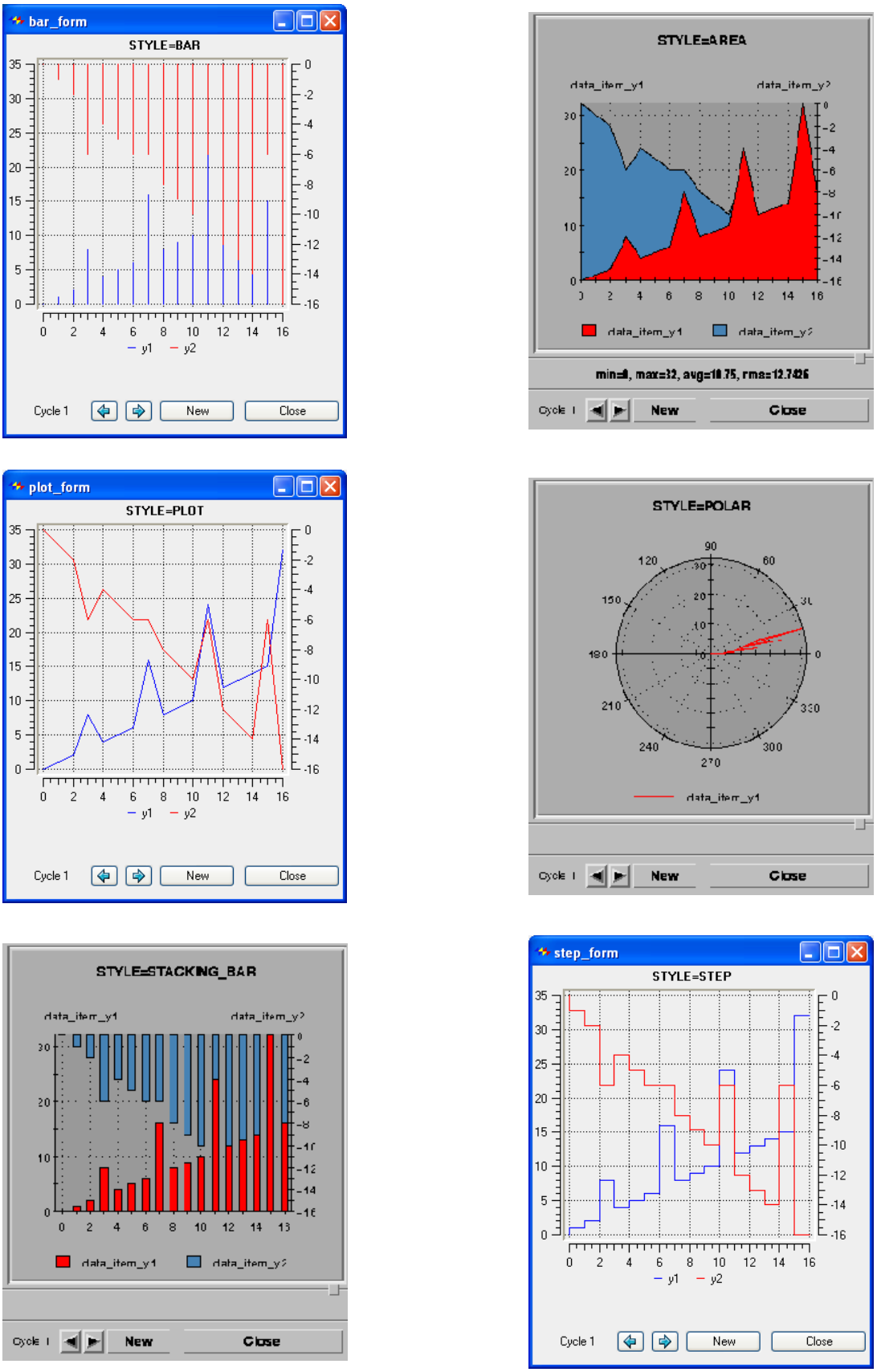


Figure 15: Plot2d Plot styles

Aspect Ratio

```
DESCRIPTION "PLOT2D: ASPECT_RATIO";

DATAPOOL
  STRUCT Axis {
    REAL {EDITABLE}
      value
    , min {SCALAR}
    , max {SCALAR}
    , aspect_ratio {SCALAR}
    ;
  };
  Axis {SCALAR}
    x
  , y
  ;
END DATAPOOL;

UI_MANAGER
  TABLE xy_table {
    ORIENTATION=VERTICAL
  , HORIZONTAL(TABLESIZE=15)
  } (
    TABLE (
      LABEL(x)
      x.value[*]

    , LABEL(y)
      y.value[*]
    );
  );

  FIELDGROUP scale_fg (
    VOID
    LABEL(x.min)
    LABEL(x.max)
    LABEL(x.aspect_ratio)

  , LABEL(x)
    x.min
    x.max
    x.aspect_ratio

  , LABEL(y)
    y.min
    y.max
    y.aspect_ratio
  );
```

```
  PLOT2D xy_plot (
    plot {
      XAXIS {
        LABEL=LABEL(x)
      , SCALE(x.min, x.max)
      , ASPECT_RATIO=x.aspect_ratio
      }
    , YAXIS1 {
        LABEL=LABEL(y)
      , SCALE(y.min, y.max)
      , ASPECT_RATIO_REF_AXIS
      , ASPECT_RATIO=y.aspect_ratio
      }
    } (
      ( x.value[*] {
          XAXIS
        , LABEL=LABEL(x)
        }
      , y.value[*] {
          YAXIS1
        , LABEL=LABEL(x)
        }
      )
    )
  );

  FORM main_form {MAIN} (
    ( ( xy_table
      , scale_fg
      )
    , xy_plot
    )
  );
...
END.
```

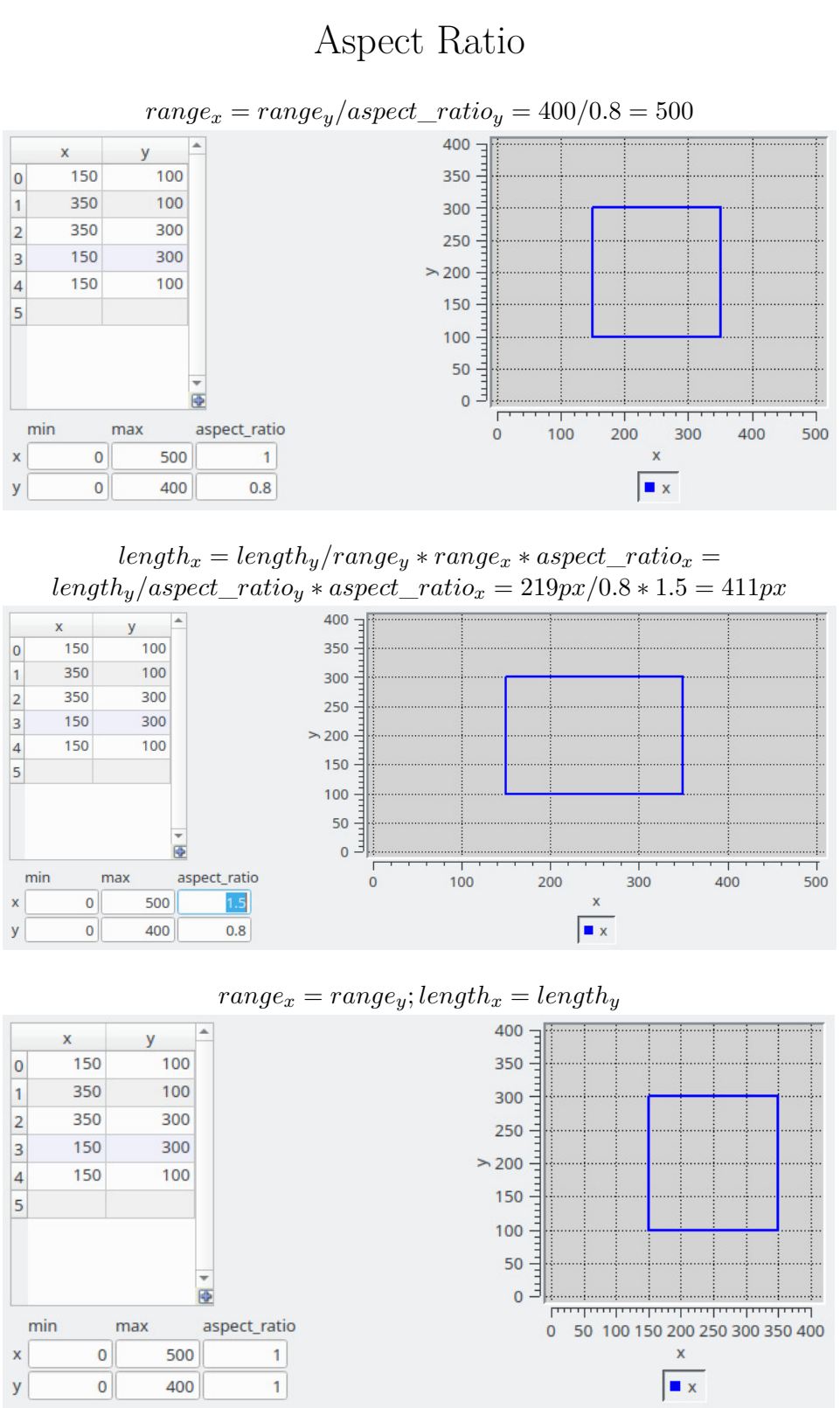


Figure 16: Plot2d: Aspect Ratio

4.6.11 Navigator

The navigator is used to display datapool-structures as a tree using folders and subfolders.

The routine used to find the pictures to use is described in section Navigator Pixmaps page 132.

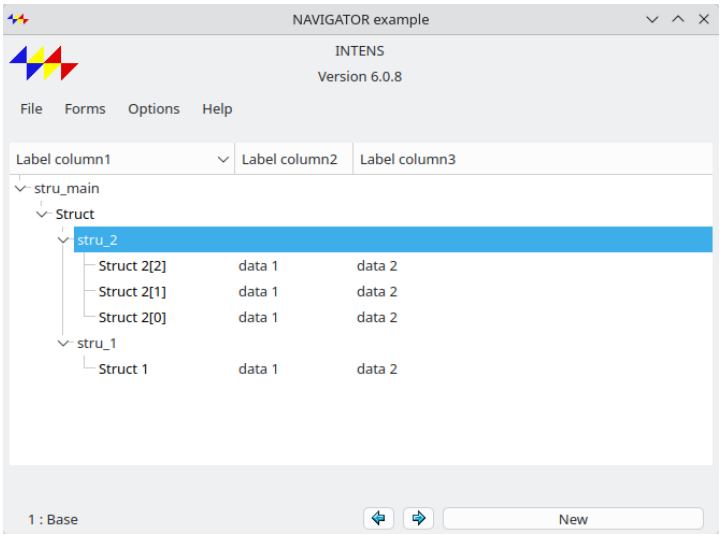
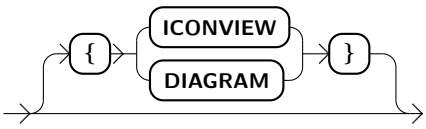


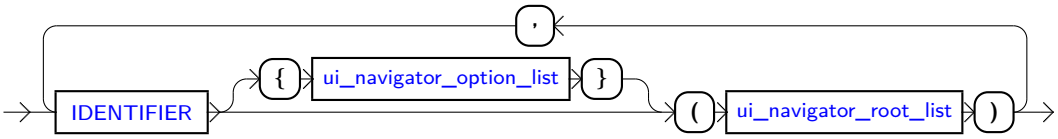
Figure 17: example of a Navigator tree

ui_navigator_type_option



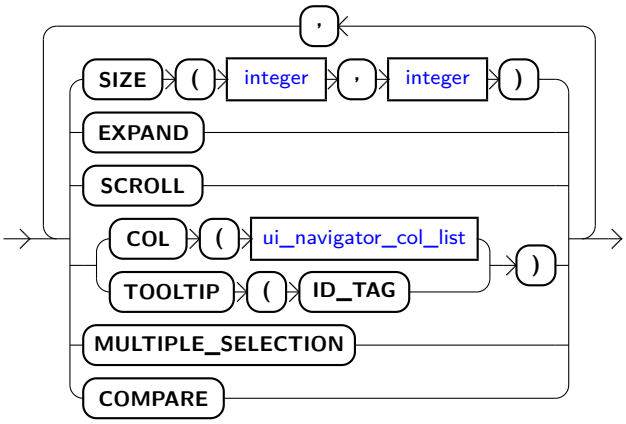
Navigator type	description
ICONVIEW	Show as icons. See section Navigator IconView page 130.
DIAGRAM	Create a diagram to place, move and connect items. See section Navigator Diagram page 131.

ui_navigator_list



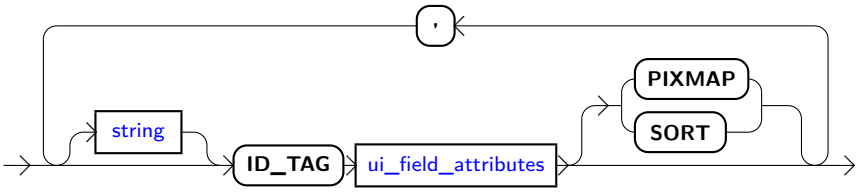
The **COL** option is mandatory when using structures that do not have a field named **name** (examples page 126). The **TAG** identifier must be declared (see **COL** declaration page 123) to define the columns that should be displayed.

ui_navigator_option_list



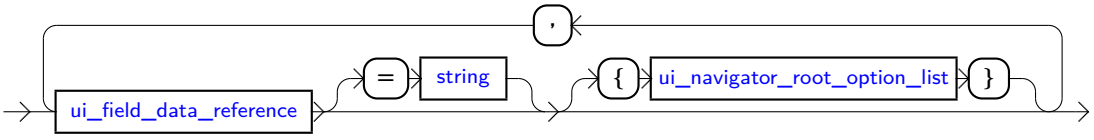
Navigator options	description
SIZE	Size of navigator window.
EXPAND	TODO.
COL	Columns declarations.
TOOLTIP	the value of the referenced structure field is shown as the tooltip of this row
MULTIPLE_SELECTION	enable the selection of multiple rows (see function statement **GET_SELECTION** on page 205).
COMPARE	This navigator is used to display a compare result (see function statement **COMPARE** on page 195).

ui_navigator_col_list



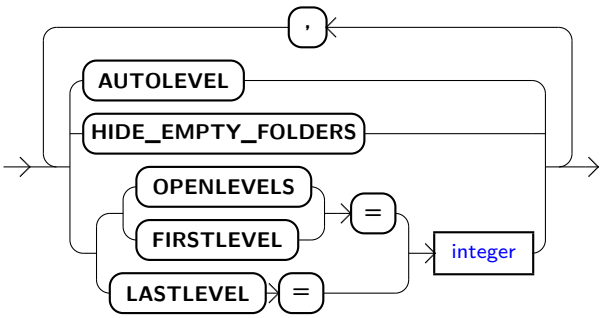
Columns declaration	description
`string`	column title
`ID TAG`	references a structure field previously declared in datapool section (see section data_item_options on page 40).
`ui field attributes`	defines, how fields are displayed **(*scale :width :precision :tsep)**. See section ui_field_attributes page 76.
PIXMAP	TODO
SORT	The navigator rows are sorted using this column. You can select an other column using the mouse.

ui_navigator_root_list



Root list	description
`data reference`	references a structure variable (Structure definition page 49)
string	use this string instead of the structure variable's **LABEL** in the header node

ui_navigator_root_option_list



By default, the **NAVIGATOR** shows a header node (using the structure's **LABEL**) to group its entries, and shows all nodes open. Alternative behaviour can be configured using the following options.

Level refers to the depth of a nested variable:

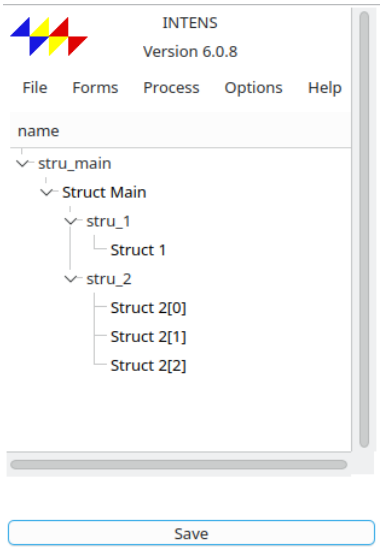
- parent is level 1
- parent.elements is level 2
- parent.elements[n].name is level 3

Root list options	description
AUTOLEVEL	Hides the header node of structs with a single entry.
HIDE_EMPTY_FOLDERS	Does not display header nodes without entries.
OPENLEVELS	Opens (expands) nodes up to and including tree-level 'integer'. Closes (collapses) nodes after level 'integer'.
FIRSTLEVEL	Does not display header nodes before level 'integer'. This option is required for **DIAGRAM** to specify the level to render.
LASTLEVEL	Hides nodes after level 'integer'.

Example of structure definition with name items:

```
DATAPOOL
  STRUCT
    STRU_1 {
      STRING {EDITABLE}
        name ,
        data_item_1 ,
        data_item_2
      ;
    };
  STRUCT
    STRU_2 {
      STRING {EDITABLE}
        name ,
        data_item_3 ,
        data_item_4
      ;
    };
  STRUCT
    STRU_MAIN {
      STRING {EDITABLE}
        name ;
      STRU_1 stru_1;
      STRU_2 stru_2;
    };
  STRU_MAIN
    stru_main
  ;
END DATAPOOL;
```

```
UI_MANAGER
  NAVIGATOR
    navigator_identifier
    (
      stru_main[*]
    )
  ;
END UI_MANAGER;
```



Example of structure definition without name items:

```
DATAPOOL
  STRUCT
    STRU_1 {
      STRING {EDITABLE}
        noname    {TAG = (tag_col_1)},
        data_item_1    {TAG = (tag_col_2)},
        data_item_2    {TAG = (tag_col_3)}
      ;
    };
  STRUCT
    STRU_2 {
      STRING {EDITABLE}
        noname    {TAG = (tag_col_1)},
        data_item_3    {TAG = (tag_col_2)},
        data_item_4    {TAG = (tag_col_3)}
      ;
    };
  STRUCT
    STRU_MAIN {
      STRING {EDITABLE}
        noname    {TAG = (tag_col_1)};
      STRU_1  stru_1;
      STRU_2  stru_2;
    };

  STRU_MAIN stru_main;
END DATAPOOL;
```

Example of corresponding navigator definition:

```
UI_MANAGER
  NAVIGATOR
  navigator_identifier{
    COL ("Label column1" tag_col_1:20,
         "Label column2" tag_col_2:10,
         "Label column3" tag_col_3:10)}(
       stru_main[*]);
END UI_MANAGER;
```

Example of functions to handle events like **select** or **drag and drop**:

```
DESCRIPTION "Implementing menu and drag'n'drop functions";
DATAPOOL
  STRUCT Motor {
    INTEGER   id;
    REAL      nom_power        {TAG=Tpower};
    STRING    id_manuf         {TAG=Tmanuf};
    STRING    model            {TAG=Tname};
    ...
  };
  Motor motors  {FUNC=select_motor};
END DATAPOOL;
FUNCTIONS
  FUNC INIT {
    motors[0].nom_power = 2.5;
    motors[0].id_manuf = "ABB";
    motors[0].model="QU 132";

    motors[1].nom_power = 7.5;
    motors[0].id_manuf = "Siemens";
    motors[0].model="1PH8131";
    ...
  };
  FUNC select_motor {
    IF( REASON_SELECT ){
      SET_MSG( THIS.model, " selected" ); }
    ELSE IF( REASON_UNSELECT ){
      SET_MSG( THIS.model, " unselected" ); }
    ELSE IF( REASON_ACTIVATE ){
      SET_MSG( THIS.model, " activated" ); }
    ELSE IF( REASON_DROP ){
      SET_MSG( SOURCE.model, " dropping to ", THIS.model ); }
  };
END FUNCTIONS;
UI_MANAGER
  NAVIGATOR
    motornav { COL( "" Tname:20
                  , "Manuf." Tmanuf:12
                  , "Output [kW]" Tpower:15 )} (
      motors[*] = "Motors" );
END UI_MANAGER;
```

To add a popup **MENU** to a navigator tree, the menu identifier must be the same as the structure identifier:

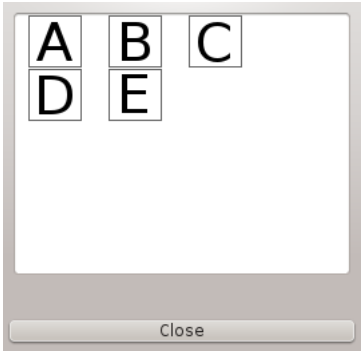
```
DESCRIPTION "Implementing menu and drag'n'drop functions";
DATAPOOL
  STRUCT Motor {
    INTEGER   id;
    REAL      nom_power        {TAG=Tpower};
    STRING    id_manuf         {TAG=Tmanuf};
    STRING    model            {TAG=Tname};
    ...
  };
  Motor motors  {FUNC=select_motor};
END DATAPOOL;
...
UI_MANAGER
  NAVIGATOR
    motornav { COL( "" Tname:20
                  , "Manuf." Tmanuf:12
                  , "Output [kW]" Tpower:15 )} (
      motors[*] = "Motors" );

  MENU Motor (
    FUNC DeleteThisMotor_func = "Delete This Motor"
   ,FUNC MoveThisMotor_func = "Move This Motor"
  );
END UI_MANAGER;
```

4.6.12 Navigator IconView

This **NAVIGATOR** type is used to show several elements that can be dragged to a **DIAGRAM**.

The **NAVIGATOR** itself has one **COL** with a **TAG**. The data item (parent in the example below) must have an attribute with that tag (name in the example). The **ICONVIEW** shows all children of it that have an attribute with that tag (parent.elements[*] with name in the example).

```
DATAPOOL
  STRUCT
    Elements {
      STRING {EDITABLE}
        name { TAG=(nametag) }, type; };
  STRUCT
    Parent {
      STRING {EDITABLE}
        name { TAG=(nametag) };
      Elements elements; };
  Parent parent;
END DATAPOOL;

UI_MANAGER
  NAVIGATOR { ICONVIEW }
    iconView_navigator{
      SIZE(200, 200), COL ("" nametag)}(
      parent { AUTOLEVEL });
  FORM
    iconView_form { HIDE_CYCLE } (
      iconView_navigator );
END UI_MANAGER;

FUNCTIONS
  FUNC
    INIT {
      parent.elements[0].name = "A";
      parent.elements[1].name = "B";
      ...

      parent.elements[0].type = "A";
      parent.elements[1].type = "B";
      ...
    };
END FUNCTIONS;
```



4.6.13 Navigator Diagram

This **NAVIGATOR** type is used to show and create a structure. Elements are dragged from an **ICONVIEW** or a **NAVIGATOR**.

4.6.14 Navigator Pixmaps

The navigator uses pixmaps to represent elements. This section explains the process used to resolve which pixmaps are displayed.

The routine consists of five sequential steps. For each step, the program tries the first possibility against all subsequent steps. If no match is found, it tries the second possibility, and so on. The process stops as soon as a valid pixmap is located.

- Determine Identifier
  - Value of **STRING** attribute 'node_name' (**DIAGRAM** only)
    This is only evaluated if the **STRING** attribute 'type' also has a value. If 'node_name' is used, the next two steps are skipped, and the program proceeds directly to searching for file extensions.
  - Value of **STRING** attribute 'type' (**DIAGRAM** and **ICONVIEW** only)
  - Variable name of the element list:
    (e.g., 'elements' in the **ICONVIEW** example)
- Determine pixmapName
  - Search the resource file section ([Navigator], [Diagram] or [IconView]) for an entry: identifier.iconPixmap;
    If no entry is found, pixmapName defaults to the identifier.
- Search for a pixmap with the following name patterns:
  - pixmapName
  - lower(identifier) (all lower case)
  - lower(identifier)-small
  - 'default' (**DIAGRAM** only)
- Search for the following pixmap types (by extension):
  - *.xpm
  - *.bmp
  - *.jpg
  - *.pbm
  - *.pgm
  - *.png
  - *.pnm
  - *.ppm
  - *.tif
  - *.svg
  - *.eps
- Search the following directories (in order):

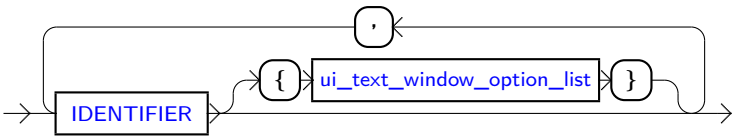
- Directories defined in **BITMAP_PATH** environment variable (separated by ':' on Linux or ';' on Windows).
- **APPHOME**/bitmaps (where **APPHOME** is an environment variable).
- **IntensHome**/bitmaps (**IntensHome** is the parent directory of the current **INTENS** executable used).
- ./bitmaps (local subdirectory).
- . (current working directory).

4.6.15 Text-Window

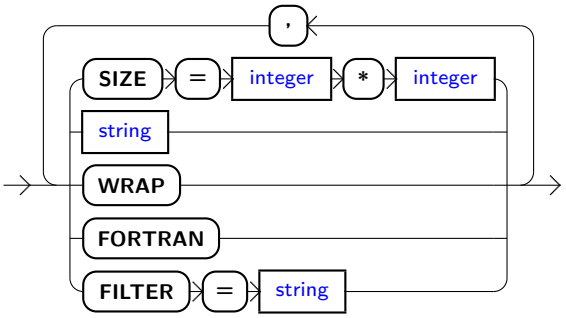
All output that is created by external programs is displayed in scrolled text windows. Unless otherwise specified the 'standard output' is printed in the text window called **STD_WINDOW** while the 'standard error' is linked to the **LOG_WINDOW**. These two windows are usually placed in the main window. They can easily be moved to any other form, as described in: example folder on page 139, section form on page 142.

Additional text windows can be created by using the **TEXT_WINDOW** declaration.
To print out data in such a window, you have to refer to the text window identifier in the processgroup-option **DISPLAY** explained in section OPERATOR on page 166.

ui_text_window_list



ui_text_window_option_list



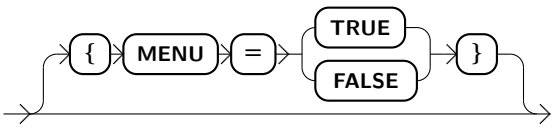
option	description
identifier	Window identifier (appears in the save-menu also)
WRAP	wrap the words, if the line is too long
SIZE	defines the displayed `height` and `width` of the text window as number of characters. The text window itself will always be sized according to the other fieldgroups within the same form.
FORTRAN	The text-window expects data in Fortran printfile-format.
FILTER	The option declares an external program (string), which preprocesses the text before sending it to the printer (usually an ASCII-to-PostScript converter, default is a2ps)

Example:

```
TEXT_WINDOW Listing_Tcmo {
  SIZE=36*148
 ,FORTRAN
 ,FILTER="/usr/local/bin/a2ps -nP -ns -nL -nH -1 -l -8 -F6.8"
};
```

STD_WINDOW and LOG_WINDOW options

ui_std_window_options



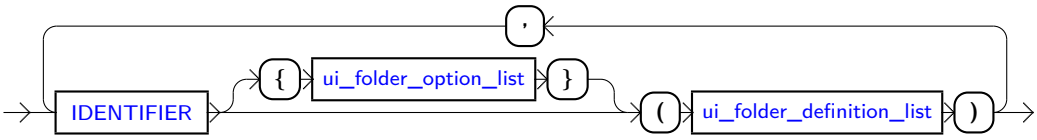
option	description
MENU	Show the window in the print menu. (Default is **FALSE**). If the window is shown, the menu label can be renamed as shown in : example of ui_manager menu on page 149

```
STD_WINDOW { MENU=TRUE };
```

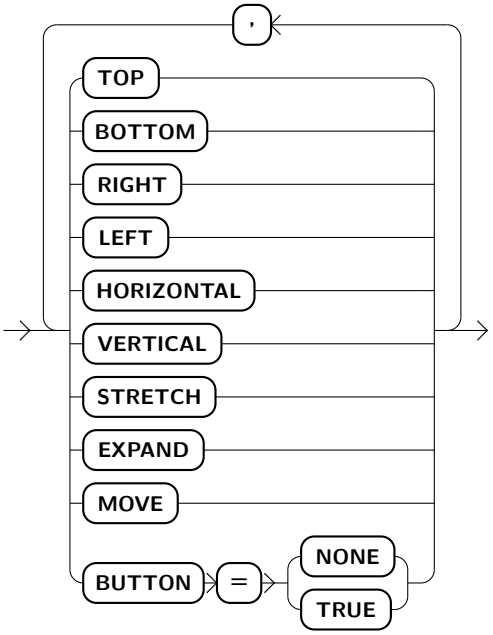
4.6.16 Folder

INTENS can group fieldgroups, plots and tables in a sequence of file folders with labelled tab buttons. (see example on page 139). Each tab button has a page associated with it which is made visible by clicking on its tab button.

ui_folder_list



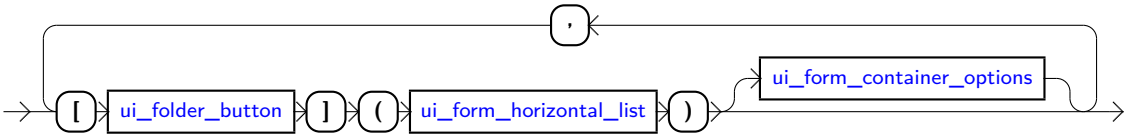
ui_folder_option_list



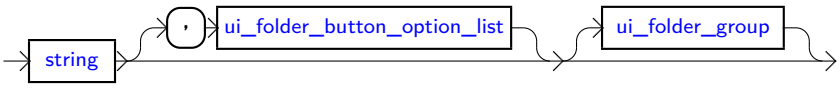
Options	description
TOP	places the tab buttons on the top side of the form.
BOTTOM	places the tab buttons on the bottom side of the form.
LEFT	places the tab buttons on the left side of the form.
RIGHT	places the tab buttons on the right side of the form.
STRETCH	All tab buttons have equal size.
HORIZONTAL	All tab button labels are horizontally oriented.
VERTICAL	All tab button labels are vertically oriented.
EXPAND	The folder can be expanded (when its form is expanded).
MOVE	Enable the possibility to move tabs.

BUTTON=NONE	Do not create tabs. You can show the different tabs using **MAP** (tab_group) in a **FUNCTION**.
BUTTON=TRUE	Show the tab of a **FOLDER** with only one page. Without this option, a single tab is not shown.

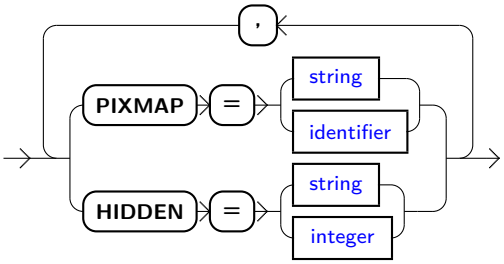
ui_folder_definition_list



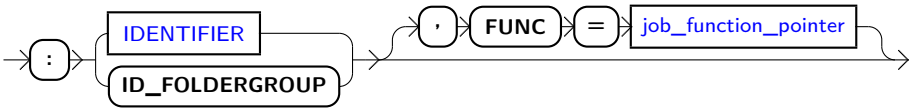
ui_folder_button



ui_folder_button_option_list



ui_folder_group



The following example shows a series of folder declarations. The tab buttons (strings in double quotes) may be associated to a tab group (is the identifier after the colon). The advantage of tab button grouping is to have an application selecting and popping up a series of folder-pages with one tab button click only. A tab group can only address one tab button per folder. This is the only limitation.

Each tab button may be associated with a function by the keyword **FUNC**. Functions are invoked on selection of the tab button. Other functions of tab buttons, which belong to the same tab group are ignored.

folder entries	description

`horizontal list`	see section ui_form_list page 140
`label string`	label of folder tab
`group identifier`	by selecting a folder having same id as other folders, each other folder will be selected also
FUNC	identifier of previously defined function, which will be executed by selecting the folder
PIXMAP	show the pixmap in the folder tab, left of the label
HIDDEN	folder button is not created when string or int evaluates to **TRUE** (string: not empty(“”), “0” or “false” (case insensitive), int: not 0). This makes it easy to hide or show a folder button using **HIDDEN** = RESOURCE(“...”)

```
DESCRIPTION "Example FOLDER";
  ...
UI_MANAGER
 FOLDER
   Data { LEFT } (  ["A" : tab_a, FUNC = func_a ] (data_A),
                    ["B" : tab_b ] (data_B),
                    ["C" : tab_c ] (data_C)
),
   Text { LEFT, STRETCH }(  ["Standard"] (STD_WINDOW),
                            ["Log"]      (LOG_WINDOW),
                            ["Input"]    (Data)
),
   Plots { TOP, VERTICAL } (  ["Plot 1" : tab_a ] (plot1),
                              ["Plot 2" : tab_b ] (plot2),
                              ["Plot 3" : tab_c ] (plot3)
);
END UI_MANAGER;
  ...
END.
```

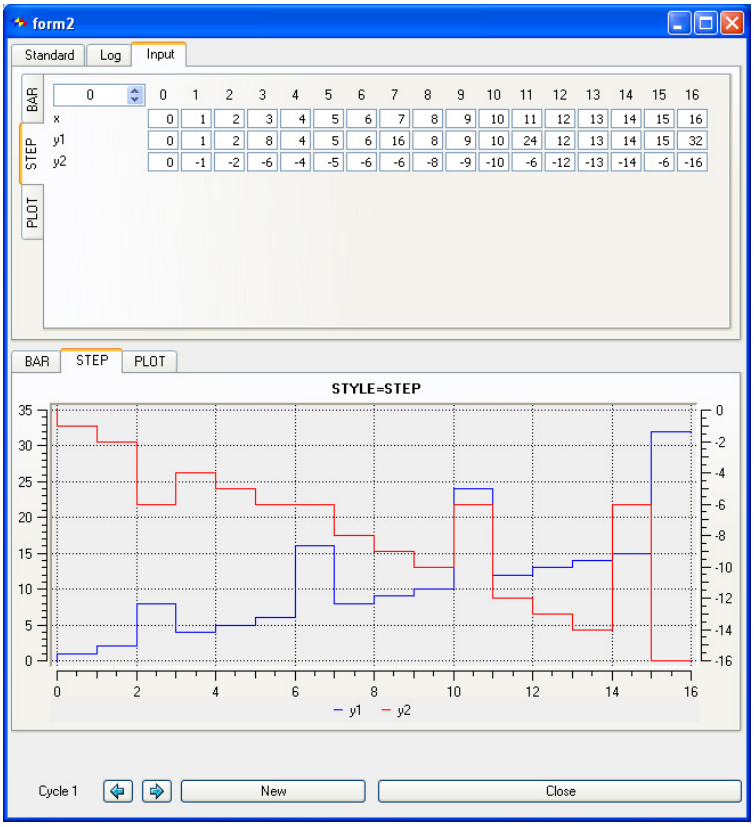


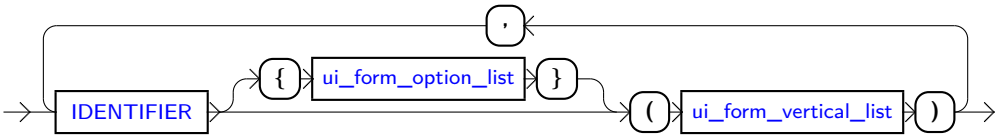
Figure 18: example of nested folders

4.6.17 Form

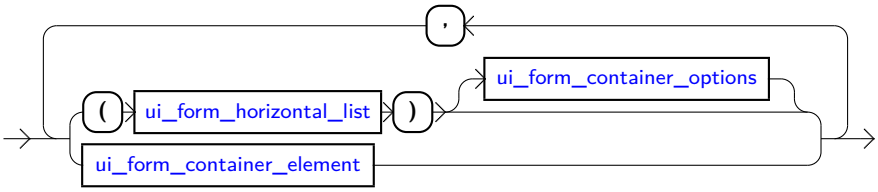
A form is a Qt dialog window that contains vertically and horizontally aligned form items and a button bar. Each form item, except **STD_WINDOW**, **LOG_WINDOW**, **SEPARATOR**, **VOID** and **STRETCH**, must reference a previously declared **FOLDER**, **FIELDGROUP**, **PLUGIN**, **UNIPLOT**, **LISTPLOT**, **PLOT3D**, **IMAGE**, **LIST**, **LINEPLOT**, **PLOT2D**, **TEXT_WINDOW**, **NAVIGATOR**, **THERMO**, **TIMETABLE** or **TABLE**.

Form items also are called **CONTAINER ELEMENT**. See ui_form_container_element on page 142.

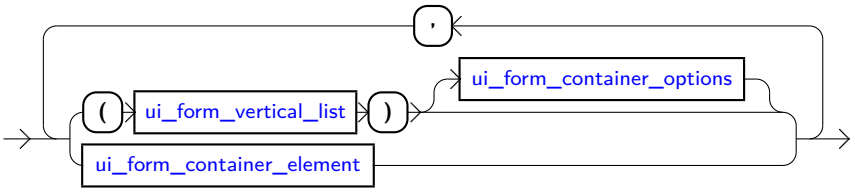
ui_form_list



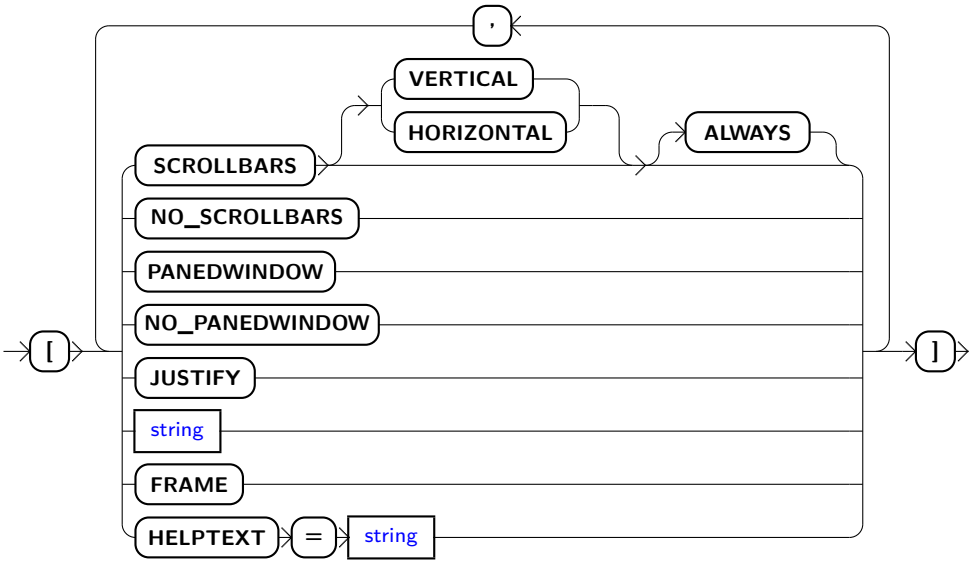
ui_form_vertical_list



ui_form_horizontal_list



ui_form_container_options

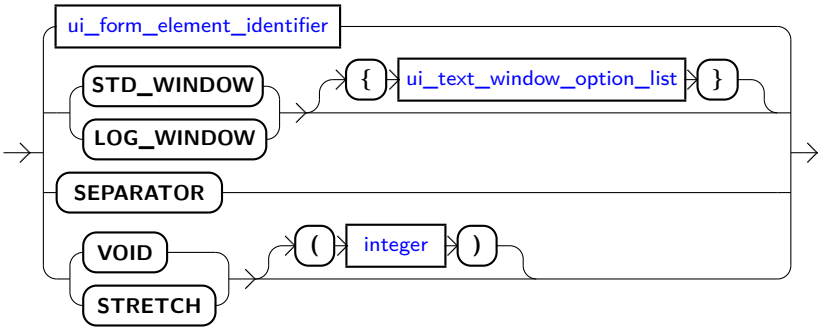


item_option	description
SCROLLBARS	The container will have vertical and/or horizontal scroll bars.

options	horizontal	vertical
	as needed	as needed
ALWAYS	always	always
HORIZONTAL	as needed	never
HORIZONTAL ALWAYS	always	as needed
VERTICAL	never	as needed
VERTICAL ALWAYS	as needed	always

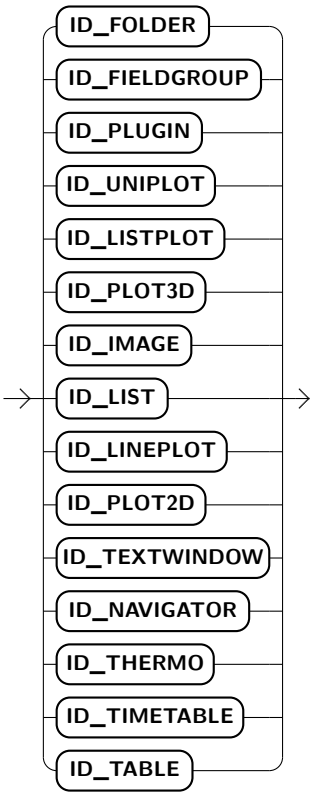
NO_SCROLLBARS	The container has no scroll bars.
PANEDWINDOW	The container will have a scrollable window separator (paned window).
NO_PANEDWINDOW	The container has no scrollable window separator (no paned window).
JUSTIFY	The rows or columns of the fieldgroups in the container are justified (aligned).
string	Display this string along with **FRAME**. Has no effect without **FRAME**.
FRAME	Display a frame around the container.
HELPTEXT	This text appears in the statusbar (or as a tooltip) as soon as the mouse-pointer is over this container.

ui_form_container_element



container element	description
STD_WINDOW	Standard window.
LOG_WINDOW	Log window.
SEPARATOR	Separator line.
VOID (n)	an empty space of n pixels
STRETCH (n)	define a stretch factor of n. The container is expanded here if additional space is available. A higher stretch factor takes more of the available space.

ui_form_element_identifier



All form items which are within a second pair of parentheses are placed horizontally, the others vertically within the form. This is explained in the following figure:

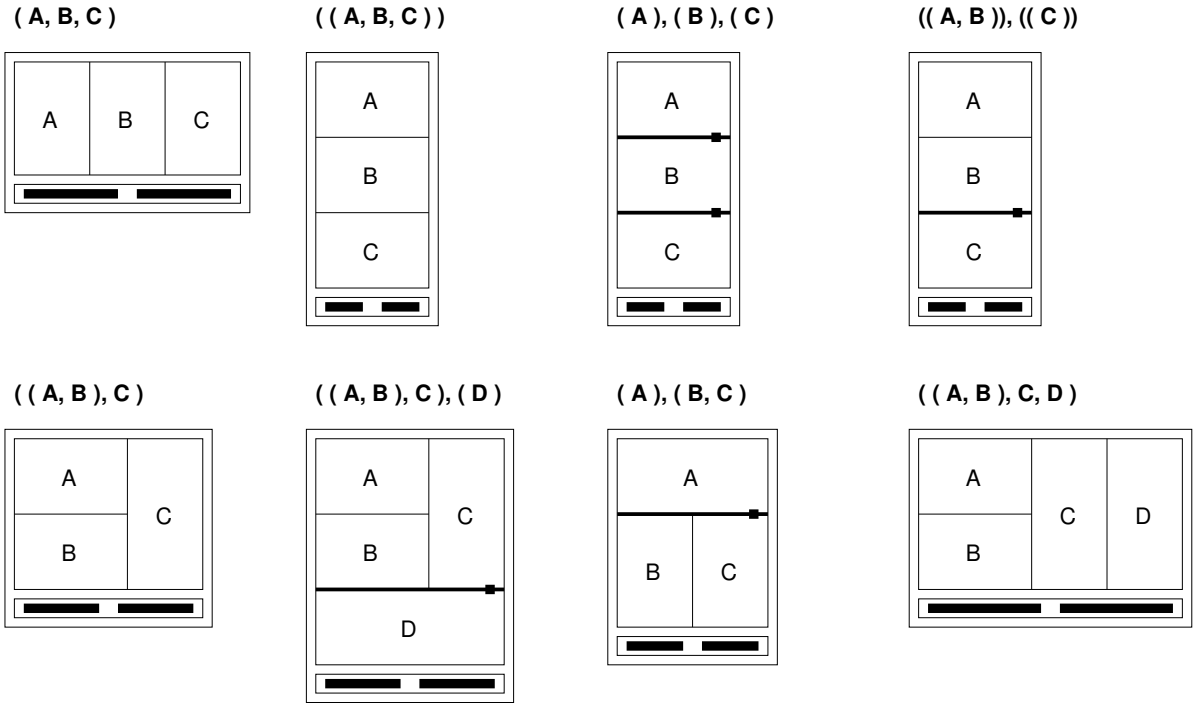
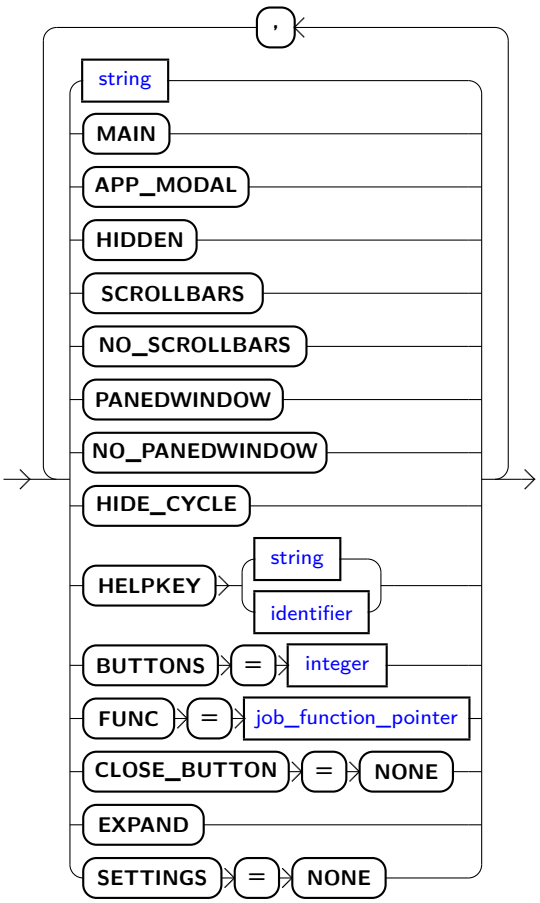


Figure 19: FORM item arrangement examples

The forms appearance can be changed with the following options:

ui_form_option_list



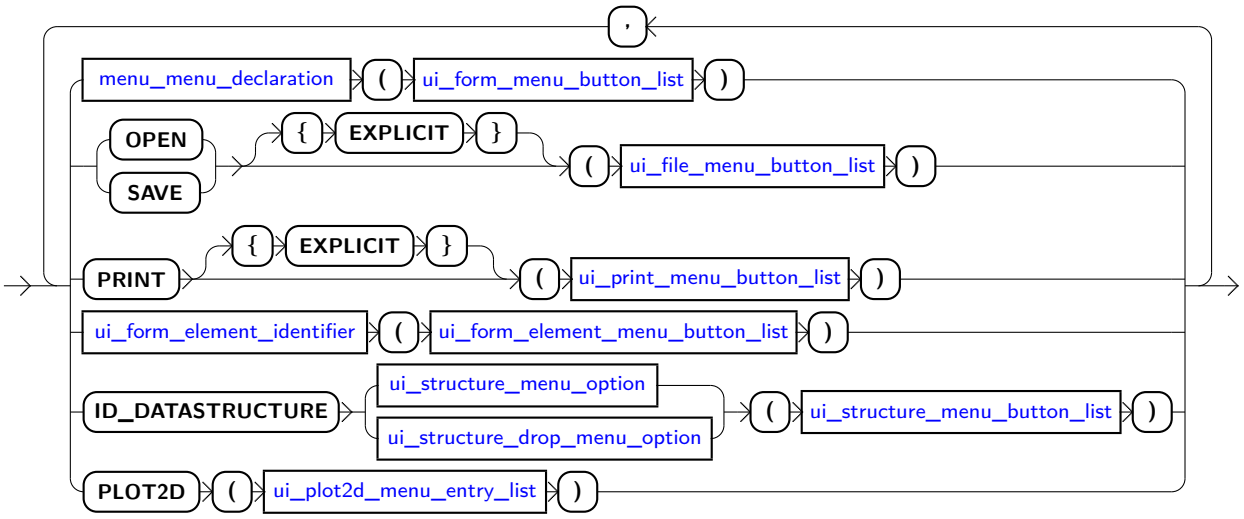
option	description
`string`	defines the name of the window
MAIN	places this form in the main window.
APP_MODAL	opens the window modal.
HIDDEN	No menu entry will be created for this form. You have to use the **MAP FORM** command to map this form on the screen (see section Description on page 204). When you create a menu entry for this form, you don't need this option (see section Menu on page 147).
SCROLLBARS	The form has vertical and horizontal scroll bars as needed. This is the default layout.
NO_SCROLLBARS	The form has no scroll bars.
PANEDWINDOW	The fieldgroups will have a scrollable window separator.
NO_PANEDWINDOW	The fieldgroups will not have a scrollable window separator.
HIDE_CYCLE	The button to change the datapool **CYCLE** is not created.

HELPKEY	A help button is created that references a page in the help file. (see section HELPFILE on page 31)
BUTTONS	Limits the number of buttons on the same action bar line and creates additional lines if needed.
USERGROUPS	A list of usergroups or the identifier of a predefined group of usergroups. (see section **??** on page **??**)
FUNC	Defines the function that will be called when a form is opened, closed or activated. (see also section FUNCTIONS on page 189)
CLOSE_BUTTON	Do not show a **Close** button at the bottom of the **FORM**.
EXPAND	Adapt the size of the **FORM** when it is opened and when an element in it is shown or hidden.

4.6.18 Menu

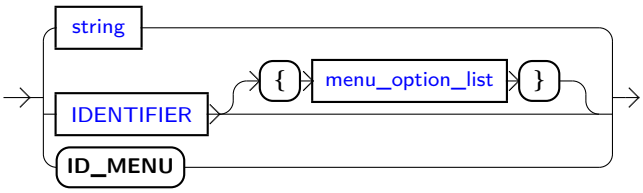
INTENS automatically generates a number of menus to display forms, to open, save and print files. These menus can be rearranged to create a specific menu structure.

ui_menu_list

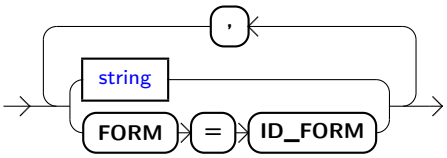


MENU Syntax	description
OPEN	References the Open submenu in the menu **File**.
SAVE	References the Save submenu the menu **File**.
PRINT	References Print submenu in the menu **File**.
EXPLICIT	Only show these explicit entries (hide automatic entries).
ui form element identifier	References the right click menu of a **NAVIGATOR**, **LIST** or **THERMO**.
ID_DATASTRUCTURE	References the right click menu of a **DATASTRUCTURE**.
PLOT2D	References the right click menu of all **PLOT2D**.

menu_menu_declaration



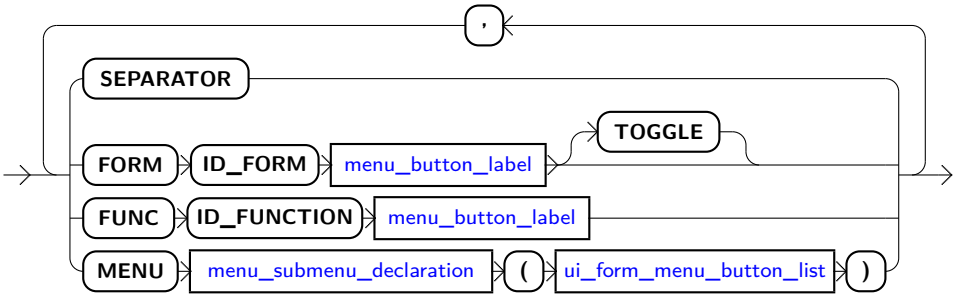
menu_option_list



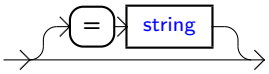
MENU options	description
string	Define a menu label (default is the menu id)
FORM	Places the menu in the form referenced by identifier.

The user can display a form by activating the corresponding menu button. **INTENS** creates the necessary buttons automatically in the form menu of the main window. This menu structure can be specified with the following syntax.

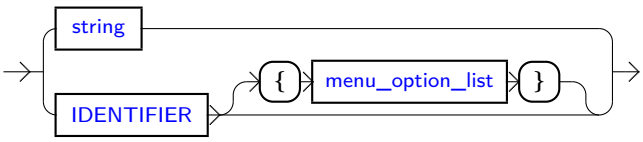
ui_form_menu_button_list



menu_button_label



menu_submenu_declaration



MENU Syntax	description
MENU	creates a new submenu.
FORM	creates a menu button within the submenu containing the name of the form to be displayed with this button.
TOGGLE	The menu entry is created as a toggle, which opens and closes the form.
FUNC	calls a function defined in section FUNCTIONS page 189.

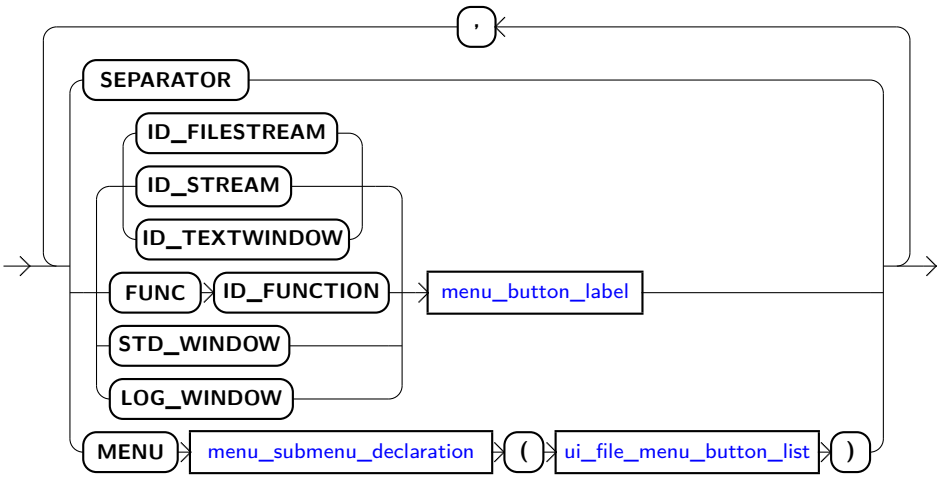
SEPARATOR	creates a separator line.

Example:

```
MENU
   "Results" (
      FORM Form_1,
      FORM Form_2,
      MENU "Plots" (
         FORM curve_form
      )
    );
```

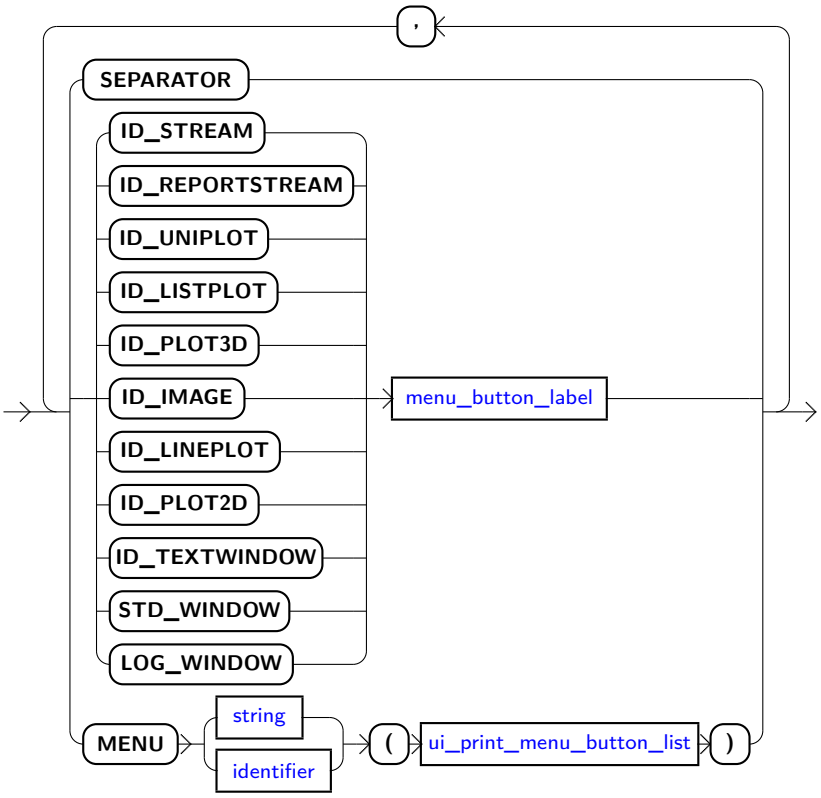
The file menus **OPEN**, **SAVE** and **PRINT** can be rearranged as well.

ui_file_menu_button_list



MENU Syntax	description
MENU	creates a new submenu.
`identifier`	creates a file menu button within the corresponding submenu. Must be a previously declared FILESTREAM or plotdiagram identifier.
SEPARATOR	creates a separator line.
STD_WINDOW	references the standard window and sets the menu label.
LOG_WINDOW	references the log window and sets the menu label.
FUNC	calls a function defined in section FUNCTIONS page 189.

ui_print_menu_button_list



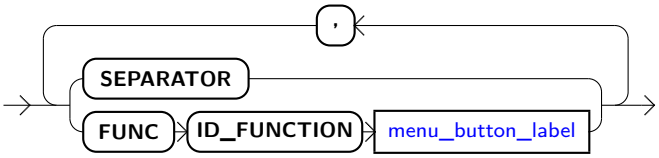
MENU Syntax	description
SEPARATOR	creates a separator line.
`menu button label`	creates a print menu button for a previously declared element.
STD_WINDOW	references the standard window and sets the menu label.
LOG_WINDOW	references the log window and sets the menu label.
MENU	creates a new submenu.

STD_WINDOW and **LOG_WINDOW** are only shown in the print menu if their **MENU** option is TRUE (see page 135).

Example of a print menu:

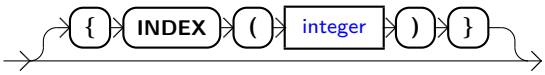
```
MENU
   PRINT (
      MENU Windows (
         STD_WINDOW = "Standard",
         LOG_WINDOW = "Log"
      ),
      MENU Plots (
         Plot1 = "First Plot",
         Plot2 = "Second Plot"
      )
   );
```

ui_form_element_menu_button_list



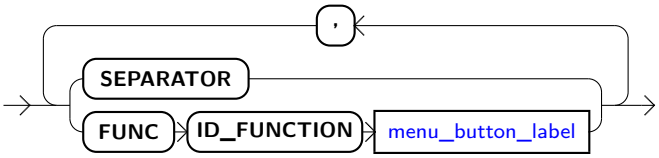
MENU Syntax	description
SEPARATOR	creates a separator line.
FUNC	calls a function defined in section FUNCTIONS page 189.

ui_structure_menu_option



ui_structure_drop_menu_option

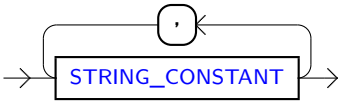
ui_structure_menu_button_list



MENU Syntax	description

INDEX	you can define multiple menus for the same structure for the **DIAGRAM NAVIGATOR** (See section Navigator Diagram page 131) Use **INDEX** option to declare which menu you are defining. defaults to 0.
DROP	create a DROP menu. The menu is opened when an item is dropped to this item.
SEPARATOR	creates a separator line.
FUNC	calls a function defined in section FUNCTIONS page 189.

ui_plot2d_menu_entry_list



All **PLOT2D**s have the same right click menu. The default menu contains all possible entries. If that is too much, the desired entries can be defined. And the order can be changed. As this is valid for all **PLOT2D**, it can only be done once.

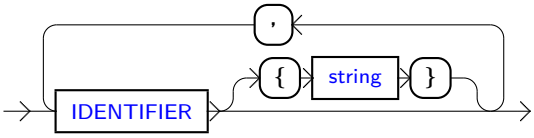
These are the possible entries:

string	**menu entry** description
Redraw	**Redraw** Redraw the plot.
Zoom	**UI Mode** Select a UI mode. See paragraph UI Mode in section Plot2d on page 108.
Annotation	**Show X-Annotation-Labels ...** Show or hide the X-Annotation labels. (only in plots with X-Annotations).
Reset	**Zoom Out** Reset the zoom.
Print	**Print ...** Print the plot.
Config	**Configuration ...** Open the configuration dialog where you can choose what to plot where.
Cycles	**Select cycles ...** Open the dialog to select what cycles to plot. The menu entry is only active when more than one cycles are present.
Scale	**Scale ...** Open the scale dialog to scale the axes.
Logarithmic	**Logarithmic** Submenu to select what axes should be scaled logarithmic.
Style	**Style Y1, Style Y2** Two submenus to select the style for the graphs on the respective y-axis.
Copy	**Copy** Copy the plot to the clipboard.
Fullscreen	**Fullscreen** Open a new window with only this **PLOT2D**.

Property	**Property** Show Plot2D Properties window.

4.6.19 Psplot

ui_psplot_list



PSPLOT Syntax	description
`IDENTIFIER`	Psplot identifier (name).
`string`	Psplot title.

4.6.20 Image

An image shows scanning probe data sent to **INTENS** trough an MFM socket (see section Sockets on page 175).

ui_image_list

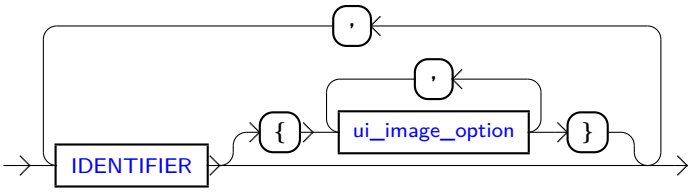


IMAGE Syntax	description
`IDENTIFIER`	Image identifier (name).

ui_image_option

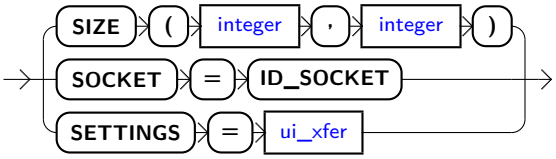


image option	description
SIZE	Width and height of the image.
SOCKET	The image will show data aquired trough this socket. (see page Sockets on page 175). Several images normally use the same socket.
SETTINGS	store image settings in ui_xfer.
`ui xfer`	Data item (see section ui_xfer on page 69).

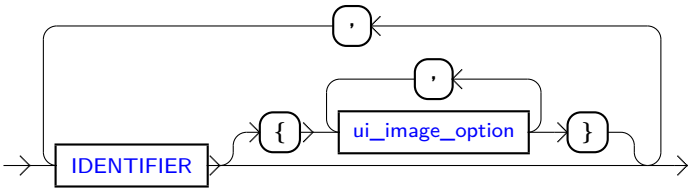
The **SETTINGS** structure contains the following variables :

```
STRUCT IMAGE_SETTINGS_STRUCT {
  INTEGER
    channel      // references MFM_channel
  , direction    // 0: forward, 1: backward
  , processing   // 0: raw, 1: avg, 2: line fit, 3: plane fit
  , mapping      // 0: automatic, 1: manual, 2: std dev
  , scaling      // 0: automatic, 1: manual
  ;
  REAL
    max          // maximum value
  , min          // minimum value
  , weight       // std dev parameter
  , x            // x scaling factor
  , y            // y scaling factor
  ;
};
```

4.6.21 Lineplot

An lineplot shows scanning probe data sent to **INTENS** trough an MFM socket (see section Sockets on page 175).

ui_lineplot_list

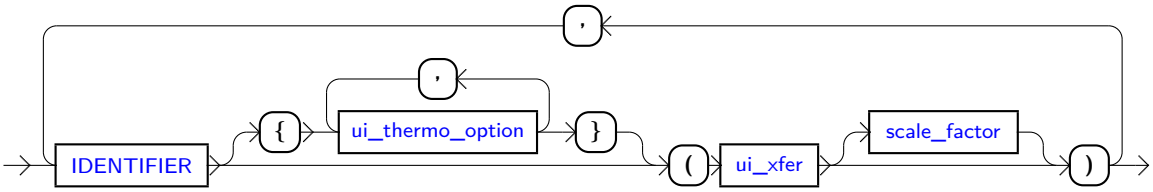


LINEPLOT Syntax	description
`IDENTIFIER`	Lineplot identifier (name).
`ui image option`	see section Image on page 154.

4.6.22 Thermo

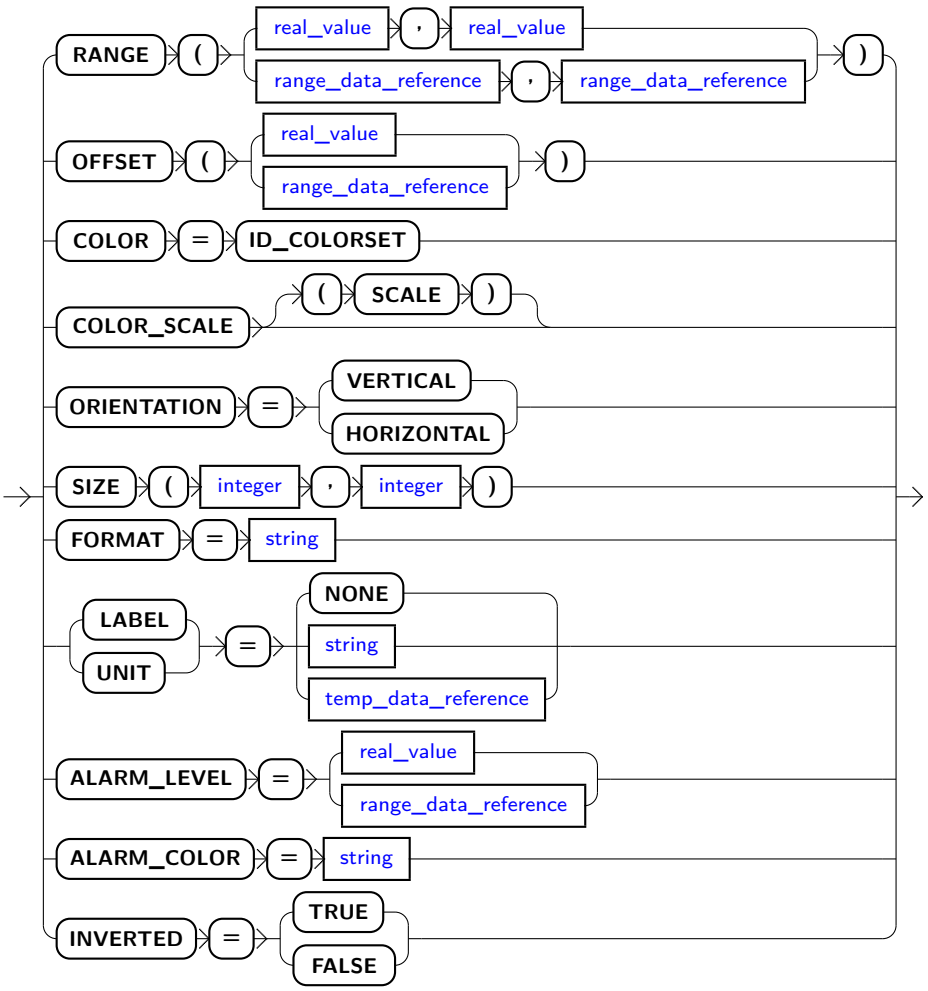
A thermo shows the value of a data item in a graphical way.

ui_thermo_list



THERMO Syntax	description
`IDENTIFIER`	Thermo identifier (name).
`ui xfer`	Data item (see section ui_xfer on page 69).
`scale factor`	see section Scale Factors on page 20.

ui_thermo_option



thermo option	description
RANGE (min, max)	Thermo range from min to max. min, max can be constant (real_value) or variable (st_data_reference, see section Referencing data variables (Data item) on page 61)
OFFSET	Offset added to the value before looking up the color in the `ID_COLORSET`.
COLOR	Use colors defined in colorset (see section Data ColorSet page 47)
COLOR_SCALE	The thermo is shown as a color scale. **SCALE**: The colors (in the color scale) are interpolated.
ORIENTATION	Thermo orientation.
SIZE (w, h)	Width and height of the thermo in pixels.
FORMAT	Axis number format. "10" : width of the number "10g" : width of the number, change to scientific if number is too wide
LABEL	Thermo label.
UNIT	Thermo unit. **NONE** : no label or unit. string : constant label or unit (see section String on page 21). **temp_data_reference** : variable label or unit (references a data item declared in the datapool, see section Data Reference on page 51).
ALARM_LEVEL	Specify the alarm threshold. The part above this value is shown in **ALARM_COLOR**.
ALARM_COLOR	Specify the alarm color. The part above **ALARM_LEVEL** is shown in this color.
INVERTED	Default is **TRUE**. If set to **FALSE**but max < min: use a different color.

The following example shows the configuration of a thermo and how it will be displayed by **INTENS** (see page 159)

```
UI_MANAGER
  THERMO
    speed_thermo {
      RANGE ( speed_limit*0, speed_limit )
    , COLOR = speed_color
    , ORIENTATION = VERTICAL
    , SIZE ( 100, 200 )
    , FORMAT = "4"
    , LABEL = "Speed:"
    , UNIT = UNIT(speed)
    } (
      speed
    )
  ;
END UI_MANAGER;
```

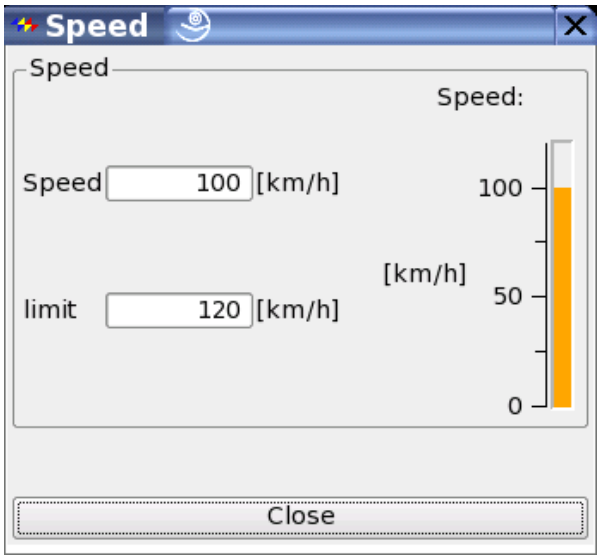
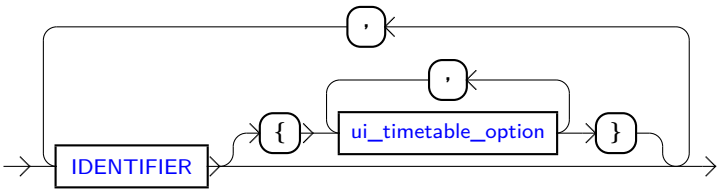


Figure 20: example of Thermo

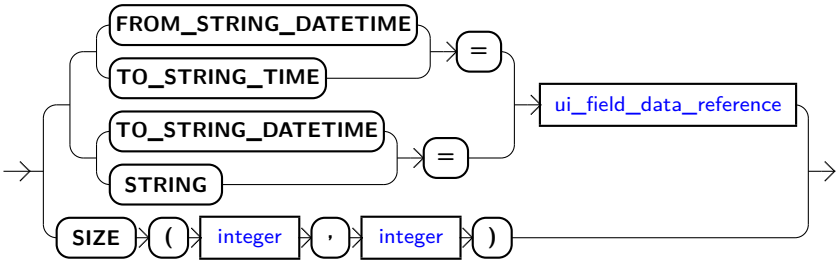
4.6.23 Timetable

ui_timetable_list



TIMETABLE Syntax	description
IDENTIFIER	Timetable identifier (name).

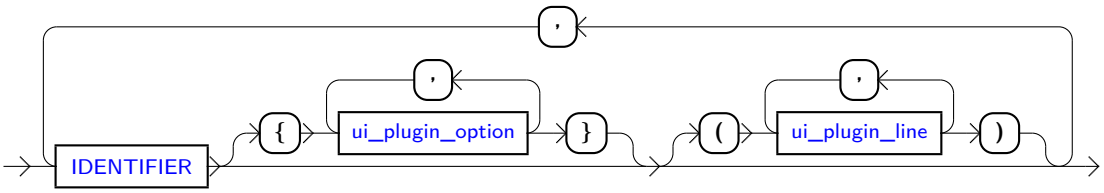
ui_timetable_option



timetable option	description
FROM_STRING_DATETIME	TODO
TO_STRING_TIME	TODO
TO_STRING_DATETIME	TODO
SIZE	TODO

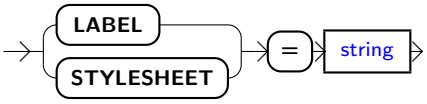
4.6.24 Plugin

ui_plugin_list



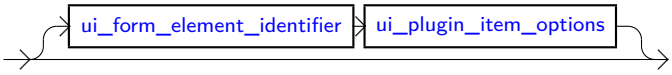
PLUGIN Syntax	description
`IDENTIFIER`	Plugin identifier (name).

ui_plugin_option

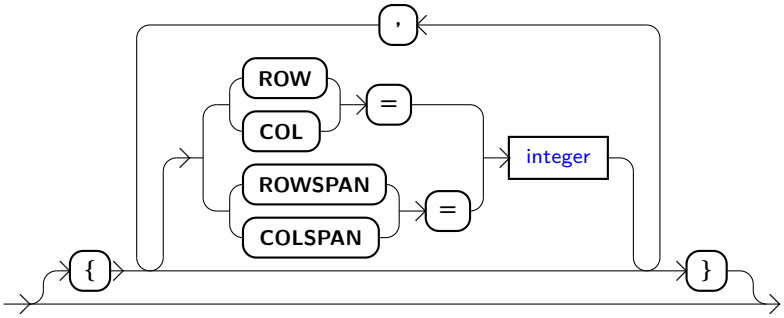


plugin option	description
LABEL	TODO
STYLESHEET	TODO

ui_plugin_line



ui_plugin_item_options



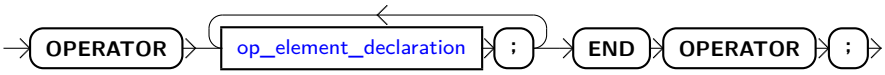
plugin option	description
ROW	TODO
ROWSPAN	TODO
COL	TODO
COLSPAN	TODO

4.7 OPERATOR

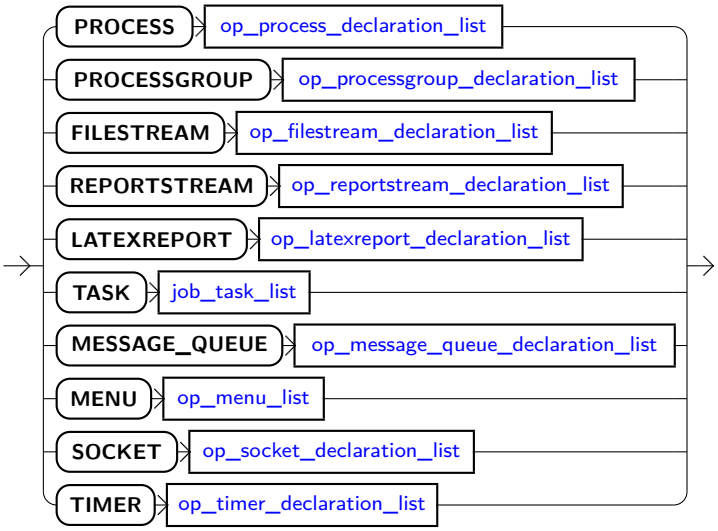
4.7.1 Description

The Operator communicates with the operating system and the external calculation programs using the pipe mechanism or the MathLink protocol. Several (one to many) calculation programs are combined to a process group that can be started and stopped by the user activating a push button. Data can also be read from and saved to files.

operator_description



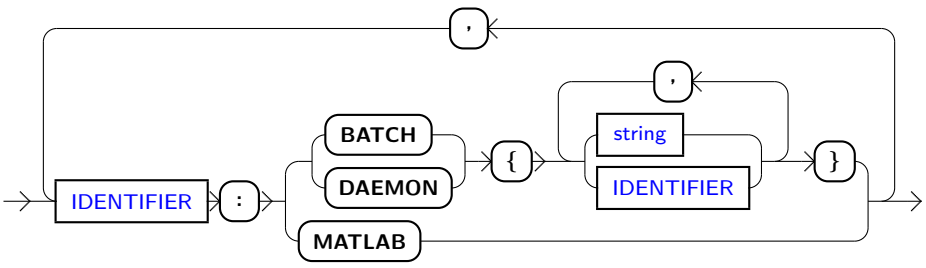
op_element_declaration



4.7.2 Process

Each external program has to be declared as **PROCESS** to be included in one or more **PROCESSGROUP**s.

op_process_declaration_list



A calculation process must have a type and a unique name (identifier).

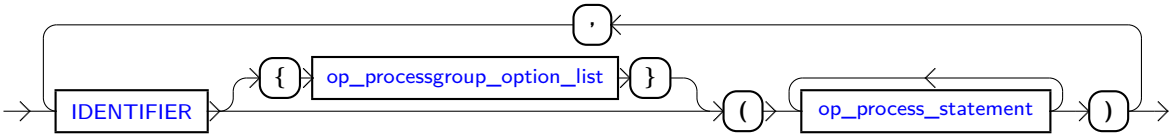
PROCESS types	description
`string`	The external program and its arguments.
`FIFO`	identifier of named pipe.
BATCH	This is a executable batch program that reads from standard input and writes to standard output. The string defines the (preferably complete) path name.
DAEMON	Starts the executable batch program described in string. User must care about ending/closing of daemon-process. No datatransfer through streams are possible.
MATLAB	This is a **Matlab**-function that is included in a Matlab file. The process name is identical to the function name.

Example:

```
PROCESS
   mech_proc : BATCH {
      "/tracomo/alpha/cmech"
     ," ", "1"
     ," ", mechres
     ," ", mechdyn
     ," ", "/home/tar/intens/tracomo/tp/who"
     }
;
```

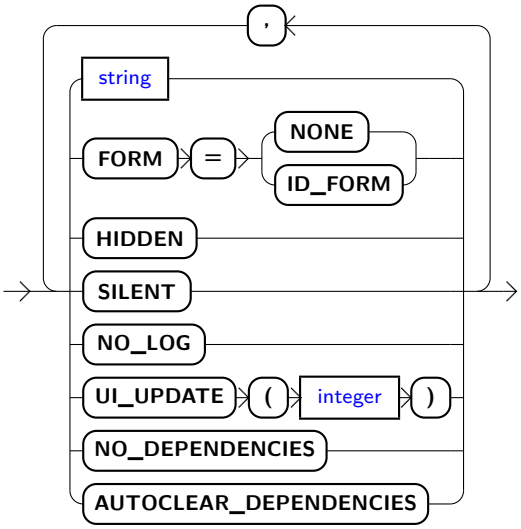
4.7.3 Process Group

op_processgroup_declaration_list



A calculation **PROCESS** must be included in a **PROCESSGROUP** which may contain any number of processes. The input and output formats of each process is defined by previously declared streams (or plot groups of type **UNIPLOT**). Unless explicitly specified, the IO channels used are standard input, standard output and standard error.

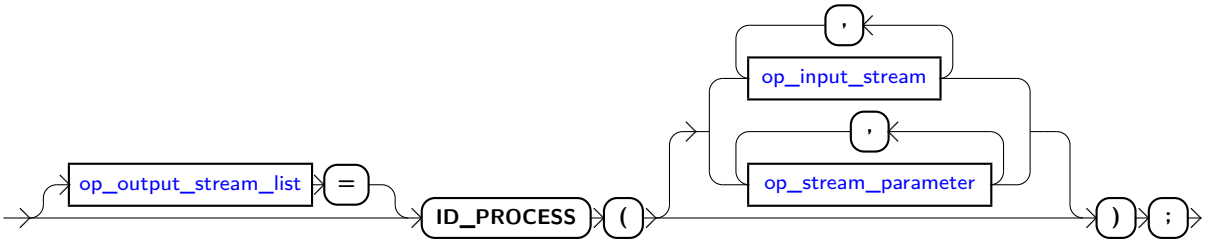
op_processgroup_option_list



process group options	description
`string`	defines the label of the push button, that starts the **PROCESSGROUP**. The default value is the name (`identifier`) of the process group.
FORM	creates the push button in the button bar of the form `identifier`. The push button is created in the main window if this option is missing.
FORM= **NONE**	This **PROCESSGROUP** will NOT have an associated button.
UI_UPDATE(n)	Updates the user interface after n lines received while processing a previously defined process.
NO_LOG	Does not print log-messages (i.E. ABORT : PROCESSGROUP) to the **LOG_WINDOW**.
SILENT	The busy cursor is shown instead of the Process Dialog.
HIDDEN	**PROCESSGROUP** is not added to the process-menu.

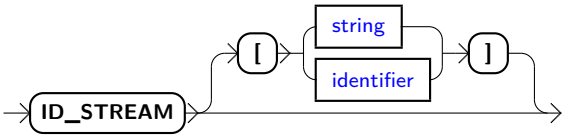
NO_DEPENDENCIES	No dependencies between input and output are added. (see paragraph Dependency on page 52)
AUTOCLEAR_DEPENDENCIES	Dependencies are cleared without user confirmation. (see paragraph Dependency on page 52)

op_process_statement

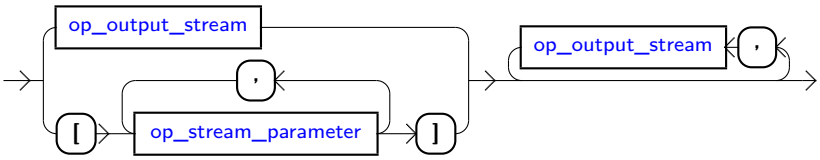


process statement	description
ID_PROCESS	Must be a previously declared **process-identifier** (section Process page 162).

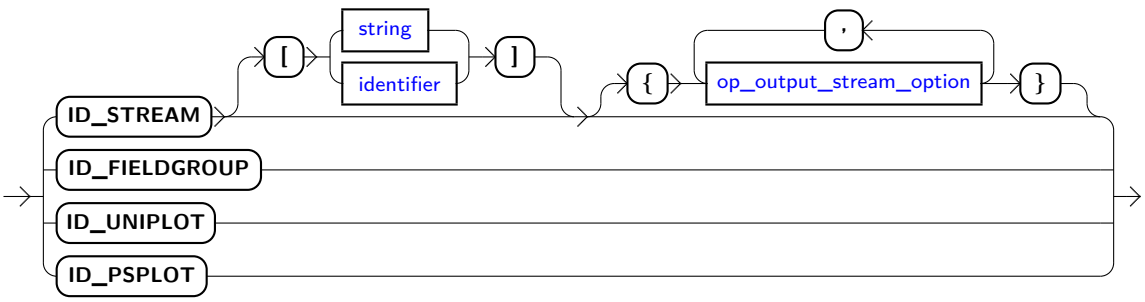
op_input_stream



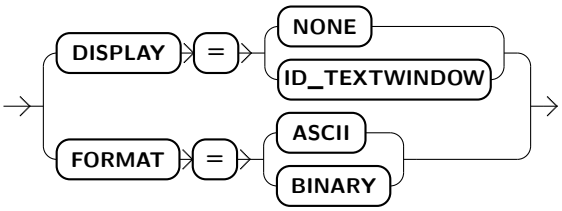
op_output_stream_list



op_output_stream

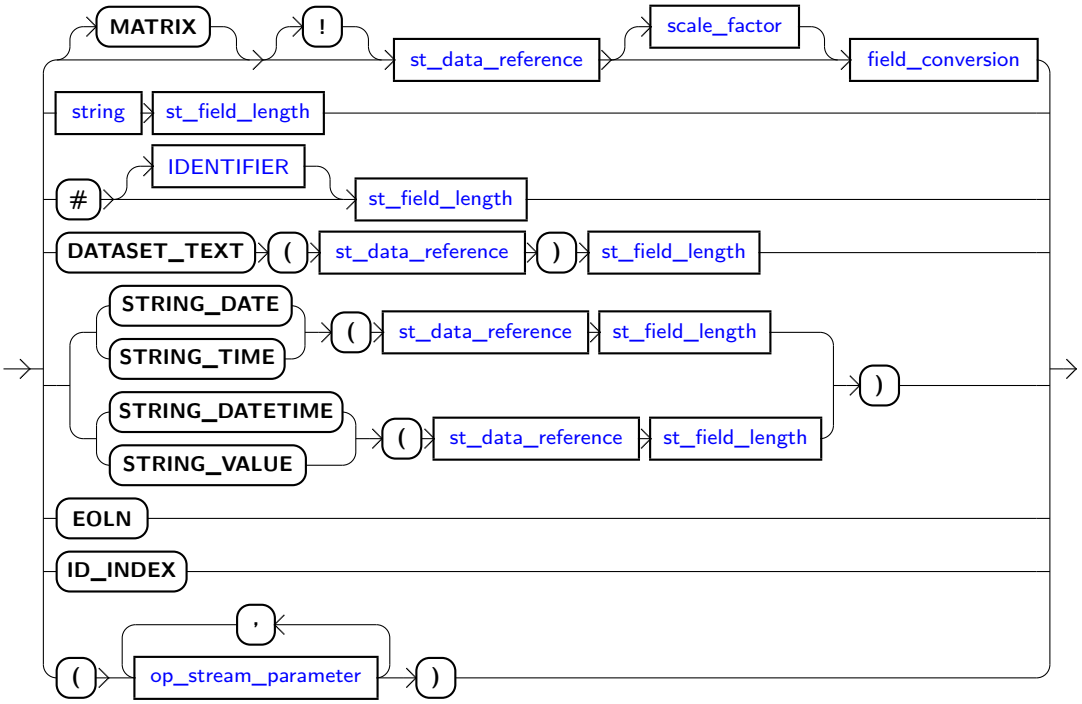


op_output_stream_option



output stream options	description
DISPLAY	Output that is read by **INTENS** from the calculation process can be directed to a user defined text window (default is **STD_WINDOW**).
DISPLAY=NONE	The output can be suppressed by setting **DISPLAY** to **NONE**.
FORMAT	TODO **ASCII** TODO **BINARY** TODO

op_stream_parameter



Instead of declaring a **STREAM** in the **STREAMER** section and using it, a stream can be defined here (without an identifier, it cannot be used elsewhere). The options are described in section st_format_command on page 54.

The FIFO (named pipe) can be specified either as identifier or string:

fifo name	description
`identifier`	The fifo name is generated by appending the current process number to the identifier. The fifo is created in the directory `/tmp` (and deleted after exiting **INTENS**). Each **INTENS** application has thus unique fifos. The same identifier can be used as argument for the batch process.
`string`	The string defines the fifo name that **INTENS** will create at startup and delete after exiting.

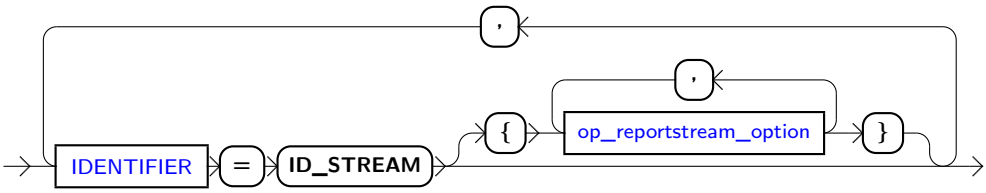
Example:

```
PROCESSGROUP
   inmo_prog {"Inmo"} (
      jobinfo_stream {DISPLAY=NONE} =
         seq_proc( dummy_in_stream );

      plot_result_stream {DISPLAY=NONE} =
         mech_inmo_proc( mech_in_stream );

      dummy_out_stream {DISPLAY=Listing_Inmo},
      res_inmo_stream[inmores] =
         inmo_proc( inmo_in_stream );
   )
;
```

4.7.4 Reportstreams

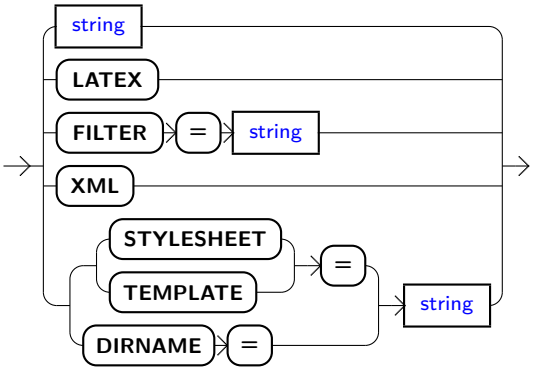
Report streams are used to send streams to the script ReportConv.py, which transform the contents to the requested format (Postscript, PDF, etc.)

op_reportstream_declaration_list



reportstream	description
`identifier`	Identifies the Reportstream.
`stream id`	References the stream defined in section STREAMER (STREAMER page 52).

op_reportstream_option



reportstream options	description
`label string`	defines the label of the push button that opens the printer selection box
LATEX	The stream contains LaTeXcommands. (See Latexreports on page 168.)
FILTER	The stream is piped to a unix-process. (Works similar to filestream-option **PROCESS**, see page 169.)
XML	TODO
STYLESHEET	TODO
TEMPLATE	TODO
DIRNAME	TODO

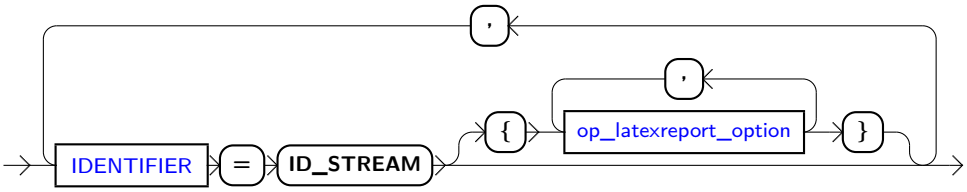
Example:

```
REPORTSTREAM
    print_motor_16 = motor_stream_16 {
       "Motor V1.6"
      ,LATEX
      }
;
```

4.7.5 Latexreports

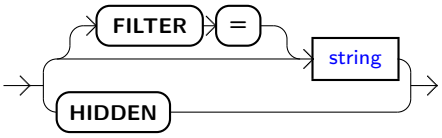
Latexreports are used to create a pdf file using latex. Data as defined in a stream, possibly processed by the program provided with the option **FILTER**, is written to a temporary .tex file. LaTeX then creates a pdf file from that .tex file.

op_latexreport_declaration_list



latexreport	description
`identifier`	Identifies the Latexreport.
`stream id`	References the stream defined in section STREAMER (STREAMER page 52).

op_latexreport_option



latexreport options	description
`string`	Defines the label of the push button that opens the printer selection box.
FILTER	The stream data is piped to a batch process. (Works similar to filestream-option **PROCESS** on page 169.)
HIDDEN	Don't add the **LATEXREPORT** to the **File Print** menu.

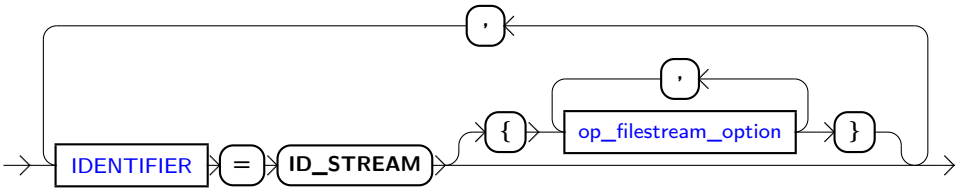
Example:

```
LATEXREPORT
    print_motor_16 = motor_stream_16 {
       "Motor V1.6"
      }
;
```

4.7.6 Filestreams

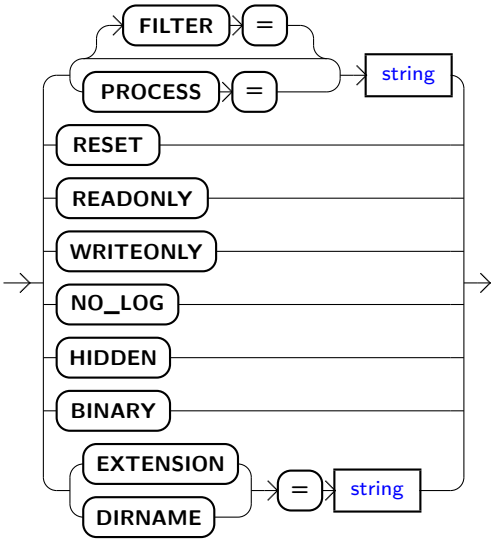
File streams are used for loading and saving streams to and from files.

op_filestream_declaration_list



filestream	description
`identifier`	Identifies the filestream
`stream id`	References the stream defined in section STREAMER (STREAMER page 52)

op_filestream_option



filestream options	Bedeutung
`string`	defines the label of the push button that opens the file selection dialog
FILTER	defines the file filter in the file selection dialog (default is *)
PROCESS	defines the executable batch program that is called for a reading or writing operation. The program must read from standard input and write to standard output. This program can be used for format conversions.
READONLY	The stream is used for reading files only. It is not added to the **File Save** menu.
WRITEONLY	The stream is used for writing files only. It is not added to the **File Open** menu.
RESET	A read operation resets the modify status of the datapool items and deletes the associated results.
NO_LOG	Does not print log-messages to the **LOG_WINDOW**.

EXTENSION	Defines extension text, which will be automaticly added to the filename.
HIDDEN	Don't add the **FILESTREAM** to the **File Open** and **File Save** menus.
DIRNAME	Defines the default directory for this filestream.

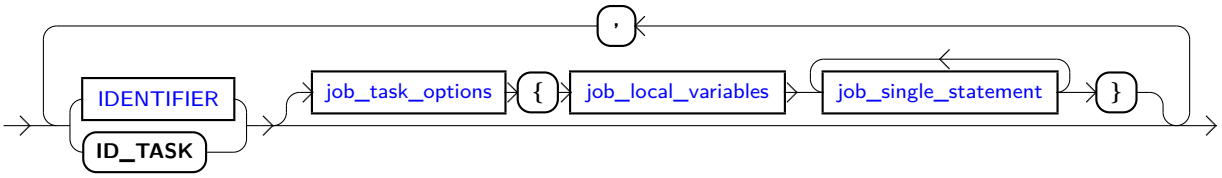
Example:

```
FILESTREAM
    get_motor_16 = open_motor_stream_16 {
       "Motor V1.6"
      ,PROCESS="/home/tar/intens/tracomo/tp/inmo/pmotor"
      ,READONLY, RESET, FILTER="INM (*.inm)"};
```

4.7.7 Tasks

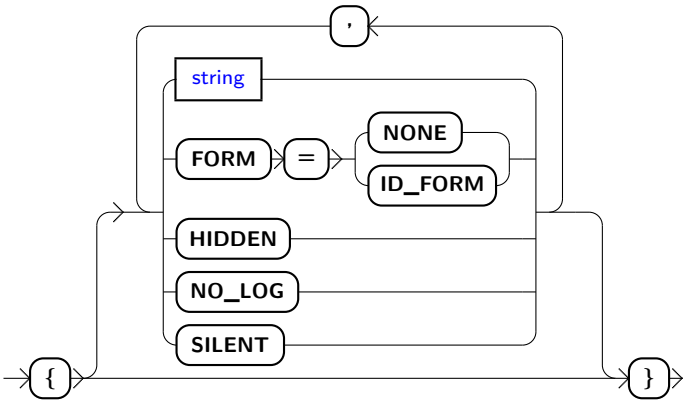
Tasks provide a process control feature: the execution of user defined process groups can be programmed using a C-like syntax (see section FUNCTIONS on page 189).

job_task_list



task	description
`identifier`	Name of the task
`statement`	Task statements have the same syntax as function statements. (see section FUNCTIONS on page 189)

job_task_options



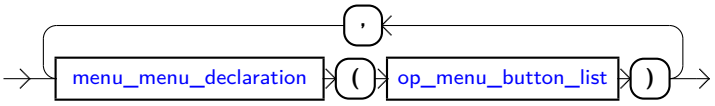
task options	description
`string`	defines the label of the push button, that starts the TASK. The default value is the name (`identifier`) of the task.
FORM	creates the push button in the button bar of the form `identifier`. The push button is created in the main window if this option is missing.
FORM=NONE	If the identifier is **NONE** no button will be created at all.
HIDDEN	Does not create a menu-entry in process-menu.
NO_LOG	Does not print log-messages (i.E. BEGIN : TASK, END : TASK, ABORT : TASK) to the **LOG_WINDOW** (except **PRINT**, **SET_ERROR** and **ABORT** messages).
SILENT	The busy cursor is shown instead of the Process Dialog.

Example:

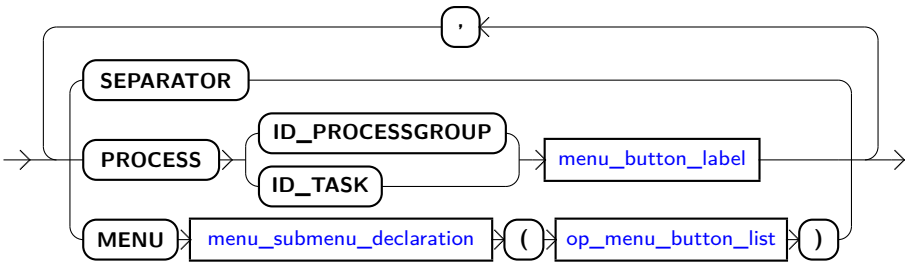
```
OPERATOR
 TASK
    task_identifier {"button label"}(
      RUN processgroup_id_1;
      RUN processgroup_id_2;
   )
  ;
END OPERATOR ;
```

4.7.8 Menu

op_menu_list



op_menu_button_list



The menu buttons for the declared process groups and tasks can be explicitly created and appended in different menus using the **MENU** declaration.

MENU Syntax	description
MENU	creates a submenu.
PROCESS	creates a menu button within that submenu containing the name of the process group or task to be started with this button.
SEPARATOR	creates a separator line

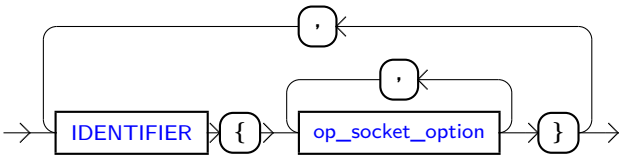
Example:

```
OPERATOR
 MENU
    Process {FORM = Form_1, "Calculations" }(
      PROCESS Process_1,
      PROCESS Process_2,
      SEPARATOR,
      MENU "more Processes" (
         PROCESS Process_3
      )
   )
  ;
END OPERATOR ;
```

4.7.9 Sockets

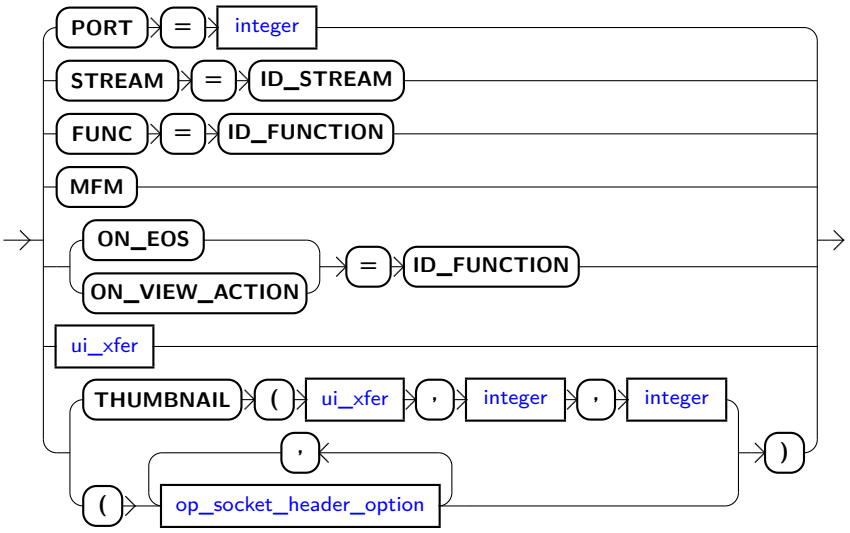
Sockets define a tcp server socket created when the application starts.

op_socket_declaration_list



socket	description
`identifier`	Identifies the Socket.

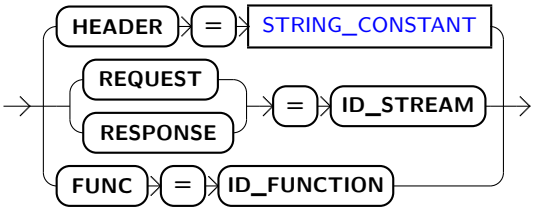
op_socket_option



socket option	description
PORT	tcp/ip port the server socket listenes at
STREAM	References a stream defined in section STREAMER (page 52) Data sent to this socket is read and stored in the datapool according to this stream.
FUNC	Function to call when all data is sent to this socket (“EOF” on a single line) and, if a **STREAM** is defined, after the data has been read using that **STREAM**.
MFM	MFM socket. Data sent to this socket is interpreted as MFM scan data.
ON_EOS	Function to call when the stream starts with “EOS” (End Of Scan). In this case, no data is stored in the datapool even if a socket is defined.

ON_VIEW_ACTION	Function to call when a view action is called from an image using this socket. (for MFM sockets only)
`ui xfer`	Store scan data in ui_xfer(see section ui_xfer on page 69). Three wildcards are required (scan, direction, channel). (for MFM sockets only)
THUMBNAIL	Data item to store thumbnail of scan data, followed by width and height in pixels. (for MFM sockets only)

op_socket_header_option



Different types of messages can be sent to a single socket. They are identified by their **HEADER** (see send_action on page 212).
A different **REQUEST**, **RESPONSE** and **FUNC** is used for each type.

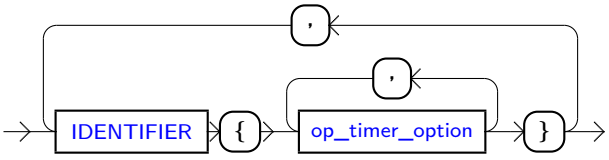
header option	description
HEADER	Defines the header string used to identify the message type.
REQUEST	References a stream defined in section STREAMER (page 52)
	Data sent to this socket is read and stored in the datapool according to this stream.
RESPONSE	References a stream defined in section STREAMER (page 52)
	A response is read from the datapool and sent according to this stream.
FUNC	Function to call when all data with this **HEADER** is sent to this socket and, if a **REQUEST** is defined, after the data has been read using that **STREAM** and, if a **RESPONSE** is defined, before that response is sent.

4.7.10 Timer

Timers are used to start a function periodically. They are defined in the OPERATOR. They are started or stopped using timer function statements (see timer_statement on page 213).

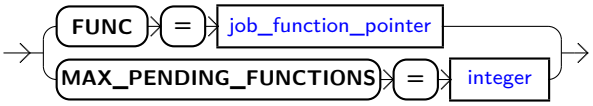
See **MESSAGE_QUEUE PUBLISH - SUBSCRIBE** example with **TIMER** on page 184.

op_timer_declaration_list



timer	description
`identifier`	Identifies the Timer.

op_timer_option



timer options	description
FUNC	Defines the function that will be called repeatedly by the timer. (see also section FUNCTIONS on page 189)
MAX_PENDING_FUNCTIONS	Defines the maximal number of pending functions. This includes a possibly running function, the functions waiting to be executed (after the running function) and the **FUNC** to be started by the timer. When this number exceeds **MAX_PENDING_FUNCTIONS**, **FUNC** is not started. The timer continues and **FUNC** may be started after another **PERIOD**. 0 (default): no maximum, **FUNC** is always started. 1: **FUNC** is started when no function is running or waiting. 2: **FUNC** is started when a function is running or not, but when no function is waiting. n: **FUNC** is started when at most n - 2 functions are waiting.

4.7.11 Message Queue

Message Queues are used to communicate with other programs using ZeroMQ. Each message can have multiple parts. The first part (**HEADER**) is used to identify the message.

The communication is done over TCP/IP.

Two patterns are implemented:

- **PUBLISH - SUBSCRIBE**
  Asynchronous pattern. One process publishes. Subscribed processes receive the messages. See **MESSAGE_QUEUE PUBLISH - SUBSCRIBE** example with **TIMER** on page 184.

- **REQUEST - REPLY**
  Synchronous pattern. One process sends a request to one other process. Depending on the **HEADER**, the request data is written to the **DATAPOOL** using the given **STREAM**(s). An optional function is called (if given).
  The response (the given **RESPONSE STREAM**(s)) are then sent back.

Message Queues can also be used to communicate with **PLUGIN**s (see section Plugin on page 161). Requests can be sent to plugins and published messages can be subscribed to. This is done using function statements. See message_queue_statement on page 214.

PUBLISH and **REQUEST** message queues are used by the **PUBLISH** and **REQUEST** function statements. See message_queue_statement on page 214.

SUBSCRIBE message queues listen for messages on a **PORT** on a **HOST**. They subscribe to one or more headers from a publisher (those they want to receive). The message (data after the header) is written to the given **STREAM**(s). When a **FUNC**tion is defined, it is called. Every **HEADER** has its own **STREAM**(s) and **FUNC**tion.

REPLY message queues listen for requests on a local **PORT**. They reply to one or more headers. The message (data after the header) is written to the given **REQUEST** stream(s). When a **FUNC**tion is defined, it is called. Finally, a response is sent (empty or defined by **RESPONSE**). Every **HEADER** has its own **REQUEST**(s), **FUNC**tion and **RESPONSE**(es).

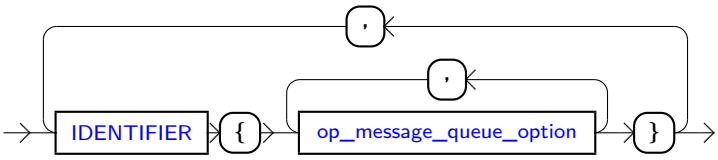
SUBSCRIBE and **REPLY**: When a **FUNC**tion is given, it is called after a message has arrived and its data is written to the **DATAPOOL**.

Note: Only one **FUNC**tion runs at a time. When a message arrives (and a **FUNC**tion is given), an already running function delays the writing to the **DATAPOOL** and the running of the **FUNC**tion.

When a **PLUGIN** is given, the received message is also forwarded to it.

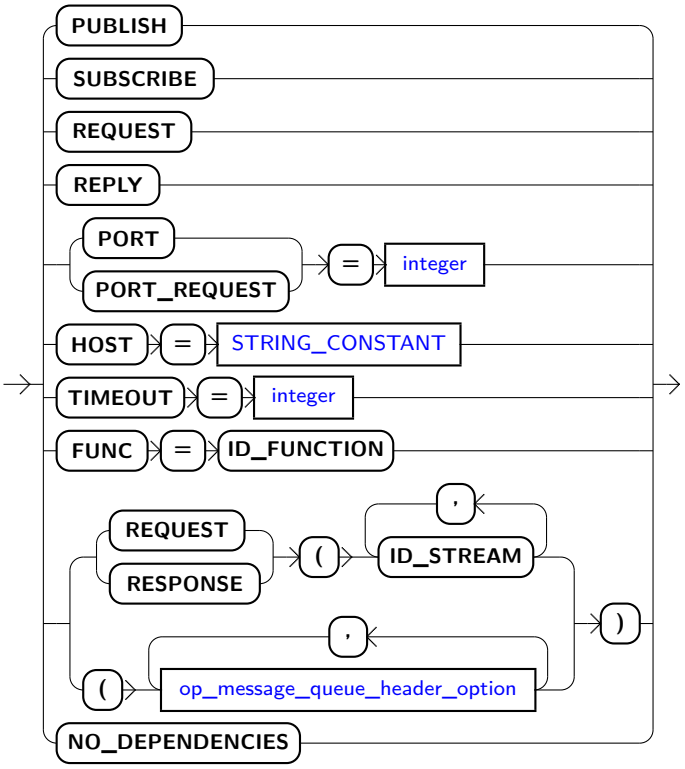
The **HEADER**s and the corresponding options are given in the op_message_queue_header_option on page 180:

op_message_queue_declaration_list



message queue	description
`identifier`	Identifies the Message Queue.

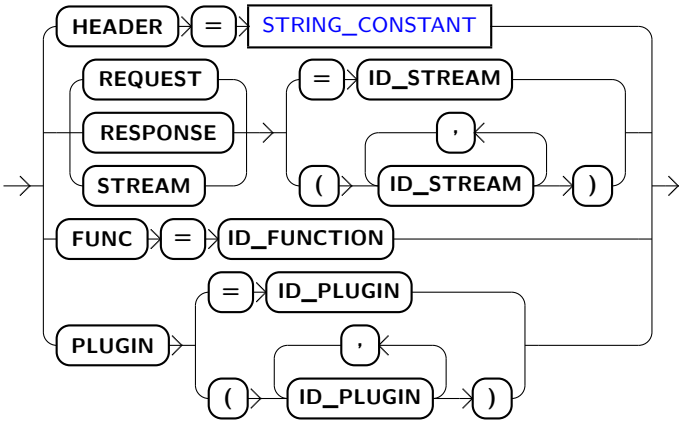
op_message_queue_option



message queue options	description
PUBLISH	Create a publish message queue to publish messages on a **PORT**.
SUBSCRIBE	Subscribe to messages published by another programm. **PORT** must be given.
REQUEST	Create a request message queue to send requests to another program. **PORT** must be given.
REPLY	Create a reply message queue to listen for requests on a **PORT**.
PORT	TCP port used for the communication.

HOST	**SUBSCRIBE** or **REQUEST**: Hostname where the other program listens. Defaults is “localhost”. **SUBSCRIBE**: Set **HOST** to ”” if it is not known yet or to delay subscribing. Use **SET_MQ_HOST** to set the **HOST** later (see message_queue_action on page 214).
TIMEOUT	**REQUEST**: Timeout in seconds used with **REQUEST** when the **REQUEST** itself does not define a **TIMEOUT**. Number of seconds to wait for a connection to the receiver and its **RESPONSE**. Default is 10 seconds. Set **TIMEOUT** to 0 to have no **TIMEOUT**.
FUNC	**REPLY**: Function used when the received header is unknown.
REQUEST	**REPLY**: Request stream(s) used when the received header is unknown.
RESPONSE	**REPLY**: Response stream(s) used when the received header is unknown.

op_message_queue_header_option



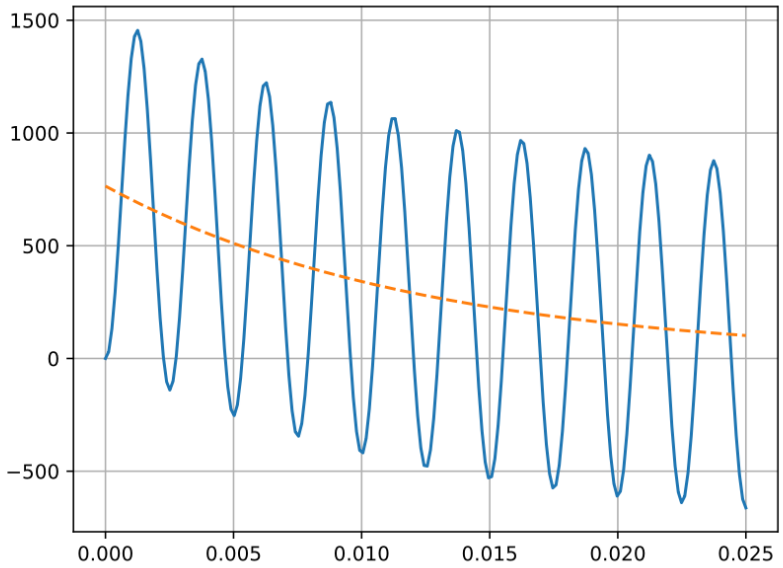
mq header options	description
HEADER	String that identifies the message.
REQUEST	**REPLY**: The received message(s) is/are written to this/these stream(s).
RESPONSE	**REPLY**: Stream(s) define the response message(s).
STREAM	**SUBSCRIBE**: The received message is written to this/these stream(s).
FUNC	Function to call after receiving the message(s).
PLUGIN	Forward the received message to the plugin.

Example:

```
DATAPOOL
  STRUCT
    MqResponse {
      STRING {SCALAR} status;
      STRING message;
    }
  ;
  MqResponse mq_response;
  STRUCT ResultReport {
    CDATA short_circuit;
  };
  ResultReport report;
END DATAPOOL;

STREAMER
  sc_input_stream{JSON}(motor, variant.short_circuit);
  sc_output_stream{JSON}(result);
  mq_response_stream{JSON}(mq_response);
END STREAMER;

OPERATOR

  MESSAGE_QUEUE python_mq {
    REQUEST,
    HOST=RESOURCE("API_GATEWAY_HOST"),
    PORT_REQUEST=RESOURCE("API_GATEWAY_PORT"),
    TIMEOUT=0
  };

  TASK calculate {_("Calculate (api-gateway)")}{
    REQUEST(MESSAGE_QUEUE=python_mq,
              HEADER="scimcalc",
              REQUEST(sc_input_stream),
              RESPONSE(mq_response_stream, sc_output_stream)
            );
   IF (mq_response.status != "ok" && VALID(mq_response.message)) {
     MESSAGEBOX("<h3>" & _("Message") & "</h3><p>" +
                  mq_response.message + "</p>");
   }
  };
```

4.7.12 Examples

The following example shows a sample **OPERATOR** configuration to display a matplotlib diagram created by a python program scplot.py:



The python program reads the JSON formatted input parameters from stdin, creates the plot in SVG format and prints the dict value a base64 encoded string in JSON format to stdout:

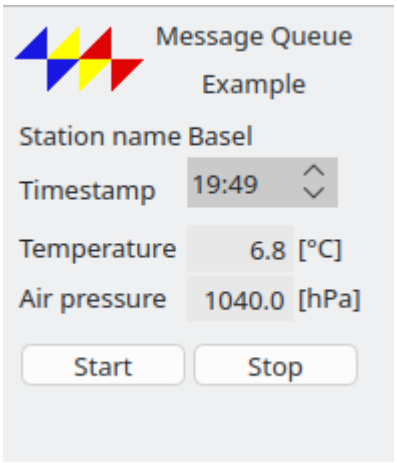
```
pars = json.load(sys.stdin)
# calculate and create matplotlib fig
...
imgbuffer = io.BytesIO()
fig.savefig(imgbuffer, format='svg')
plt.close(fig)
# Convert the SVG data to base64
svgplot = base64.b64encode(imgbuffer.getvalue()).decode()
encoding = ''
result = {}
result['plot'] = encoding + svgplot
result['time'] = t.tolist()
result['current'] = current.tolist()
json.dump(result, sys.stdout)
```

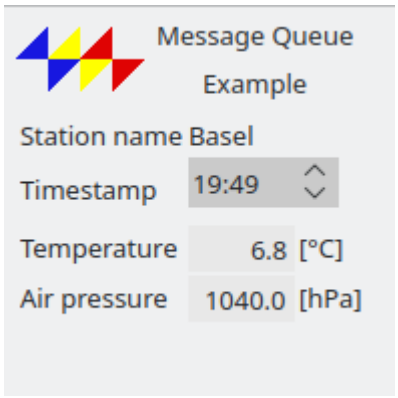
The plot is displayed in the highlighted multi line field:

```
DESCRIPTION "Short Circuit Current Fit";

DATAPOOL
  STRUCT FitParameters {
    REAL {EDITABLE, SCALAR} Io, T, f1, tmax;
    INTEGER {EDITABLE, SCALAR} nsamples;
  };
  FitParameters {SCALAR} pars;

  STRUCT ResultStruct {
    REAL time, current;
    STRING plot;
  };
  ResultStruct results;
END DATAPOOL;

STREAMER
  input_stream{JSON}(pars);
  result_stream{JSON}(results);
END STREAMER;

OPERATOR
  PROCESS fit_plot: BATCH{"tee p.in| ./scplot.py | tee r.out"};
  PROCESSGROUP fit_plot_pg {_("Fit Plot"), HIDDEN, SILENT, NO_LOG}(
    result_stream{DISPLAY=NONE} = fit_plot(input_stream);
  );
END OPERATOR;

UI_MANAGER
  FIELDGROUP
    matplotlib_plot(results.plot:30:20 {EXPAND});

FORM main_window_form{MAIN}(
  matplotlib_plot
);

END UI_MANAGER;

END.
```

NOTE: the unix filter `tee` in BATCH is used here for debugging purposes.

MESSAGE_QUEUE PUBLISH - SUBSCRIBE example with TIMER The following example shows a **PUBLISH - SUBSCRIBE MESSAGE_QUEUE** pattern. It also shows the usage of **TIMER**.





The example consists of four files:

- publisher.des : INTENS application that publishes weather data using ZeroMQ.
- subscriber.des : INTENS application that receives the published data.
- common.inc : Common code included in both publisher.des and subscriber.des.
- timer.inc : **TIMER** code included in publisher.des.

publisher.des:

```
DESCRIPTION "Publisher";

INCLUDE common.inc
INCLUDE timer.inc

UI_MANAGER
  FORM
    main_form { MAIN, HIDE_CYCLE } (
      weather_fg
    , timer_button_fg
    )
  ;
END UI_MANAGER;

OPERATOR
  MESSAGE_QUEUE
    weather_publisher {
      PUBLISH               // MessageQueue Pattern
    , PORT=5561             // TCP port
    }
  ;
END OPERATOR;

FUNCTIONS
  FUNC
    publish_weather_func {
      PUBLISH (
        MESSAGE_QUEUE = weather_publisher
      , HEADER="Weather"
      , RESPONSE=weather_stream
      );
    }
  ;

  FUNC
    INIT {
      weather.stationName = "Basel";
      t = 0;
    }
  ;
END FUNCTIONS;

END.
```

subscriber.des:

```
DESCRIPTION "Subscriber";

INCLUDE common.inc

UI_MANAGER
  FORM
    main_form { MAIN, HIDE_CYCLE } (
      weather_fg
    )
  ;
END UIMANAGER;

OPERATOR
  MESSAGE_QUEUE
    weather_subscriber {
      SUBSCRIBE             // *MessageQueue Pattern
    , HOST="localhost"      // Hostname
    , PORT=5561             // TCP port
    , (
        HEADER="Weather"
      , STREAM=weather_stream
      )
    }
  ;
END OPERATOR;

END.
```

common.inc:

```
// Common parts of the message queue examples
DATAPOOL
  STRUCT
    Weather {
      STRING
        stationName { LABEL="Station name", LABEL }
      , timestamp   { LABEL="Timestamp", STRINGTIME }
      ;
      REAL
        temperature { LABEL="Temperature", UNIT="[°C]" }
      , airPressure { LABEL="Air pressure", UNIT="[hPa]" }
      ;
    }
  ;
  Weather weather;
END DATAPOOL;

UI_MANAGER
  FIELDGROUP
    weather_fg (
      LABEL(weather.stationName) weather.stationName { COLSPAN=2 }
    , LABEL(weather.timestamp)   weather.timestamp   { COLSPAN=2 }
    , LABEL(weather.temperature) weather.temperature:6:1
        UNIT(weather.temperature)
    , LABEL(weather.airPressure) weather.airPressure*1e-2:6:1
        UNIT(weather.airPressure)
    )
  ;

  // Hide STDWINDOW and LOGWINDOW from MAIN form
  FORM
    stdwin_form { "Standard output", HIDECYCLE
                , CLOSE_BUTTON=NONE } (
      STD_WINDOW { SIZE=24*80 }
    )
  , logwin_form { "Log output", HIDECYCLE
                , CLOSE_BUTTON=NONE } (
      LOG_WINDOW { SIZE=24*80 }
    )
  ;
END UI_MANAGER;

STREAMER
  weather_stream { JSON } (
    weather
  );
END STREAMER;
```

timer.inc:

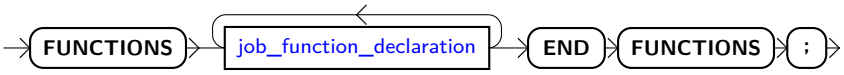
```
// Timer parts for publisher.des
DATAPOOL
  REAL t;
  STRING { EDITABLE }
    start { LABEL="Start", BUTTON, FUNC=start_timer_func }
  , stop  { LABEL="Stop" , BUTTON, FUNC=stop_timer_func };
END DATAPOOL;

UI_MANAGER
  FIELDGROUP
    timer_button_fg (
      start   stop );
END UI_MANAGER;

OPERATOR
  TIMER
    publish_weather_timer {
      FUNC=update_and_publish_weather_func
    };
END OPERATOR;

FUNCTIONS
  FUNC
    update_and_publish_weather_func {
      // update (simulate) weather data
      t += 0.1;
      weather.timestamp   = CURRENT_TIME;
      weather.temperature = 15 + SIN(t) * 10;
      weather.airPressure = 102300 + COS(t) * 3000;

      RUN ( publish_weather_func );
    },
    start_timer_func {
      START (
        publish_weather_timer
      , PERIOD=1
      , DELAY=5
      );
    },
    stop_timer_func {
      STOP ( publish_weather_timer );
    };
END FUNCTIONS;
```

4.8 FUNCTIONS

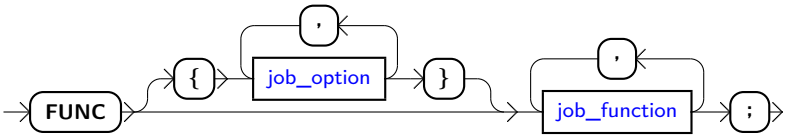
4.8.1 Description

Functions are mostly used to check user input (see section DATAPOOL on page 35).

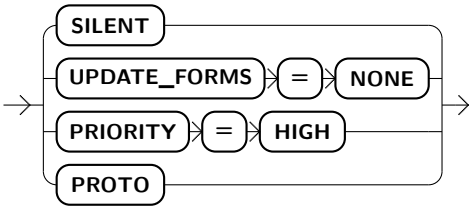
functions_description



job_function_declaration

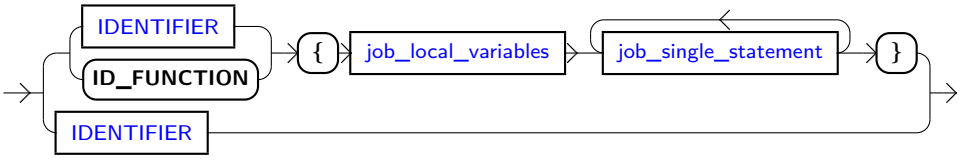


job_option



option	description
SILENT	The functions write nothing to the log window (except **PRINT**, **SET_ERROR** and **ABORT** messages), the status line and no busy cursor appears.
UPDATE_FORMS=NONE	The forms are not updated when the function is done.
PRIORITY=HIGH	When a user action, a message queue message, a timer starts a function and another function is already running, the new function is normally added at the end of the function queue: it will run after all other waiting functions finished. This option adds the function at the beginning of the function queue: it will run before the other waiting functions.
PROTO	TODO.

job_function



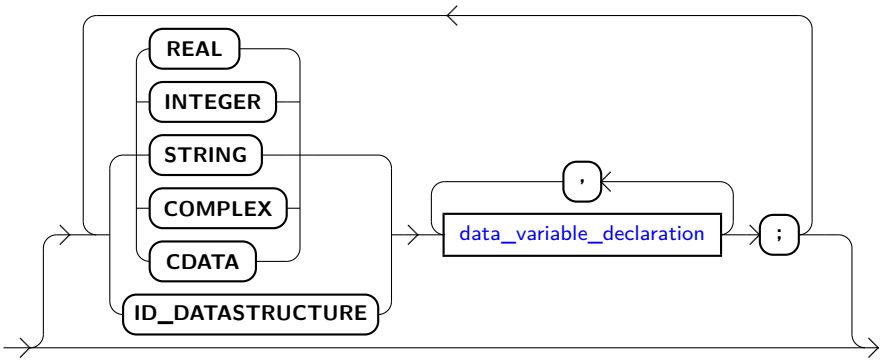
The following function identifiers have a special meaning. If they are defined, INTENS calls these functions automatically in certain situations.

As of version 5.2.1, INTENS no longer calls these functions after cycle statements (see section cycle_statement on page 208).

function identifier	description
INIT	• at startup • after clear cycle from Cycle Dialog
QUIT	• before quitting INTENS
ON_CYCLE_EVENT	• after cycle events (clear, delete, new, rename, switch) from Cycle Dialog[1]. Within the function, the event can be distinguished using • **REASON_CYCLE_CLEAR** • **REASON_CYCLE_DELETE** • **REASON_CYCLE_NEW** • **REASON_CYCLE_RENAME** • **REASON_CYCLE_SWITCH**
ON_CYCLE_SWITCH	• after cycle events (delete, new, switch) from Cycle Dialog[1]
AFTER_UPDATE_FORMS	• after a gui update (after forms are updated)

[1] Menu Options > Cycle or ctrl-l

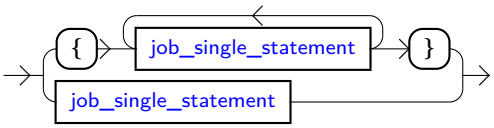
job_local_variables



Job local variables are defined just like *data items* (see section Data Item on page 36). The only difference are the *data variable attributes* which are not available for *job local variables*.

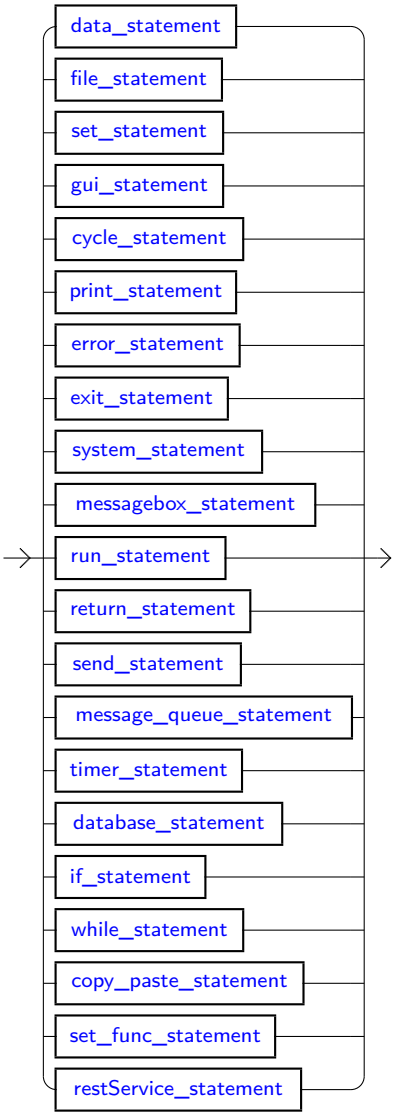
4.8.2 Statements

job_statement

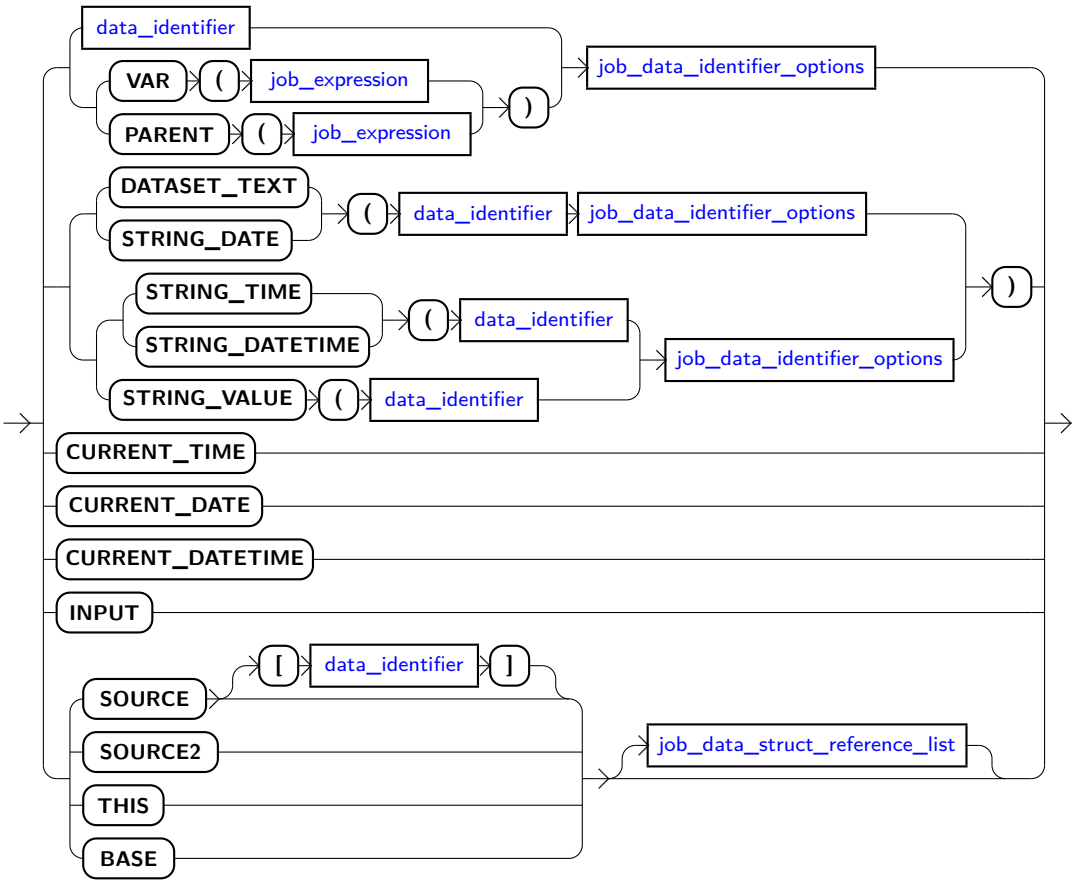


Function statements have the same syntax as task statements (OPERATOR section Tasks).

job_single_statement

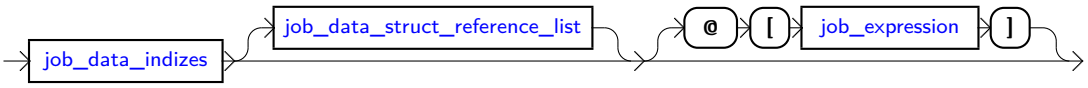


job_data_reference



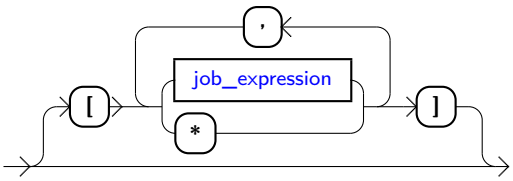
statements	description
VAR(string-item)	references a datapool item element. String dataitem contains the identifier of a data item. By changing the contents of string-item at runtime, specified datapool item is referenced.
PARENT	references parent structure of `job_expression`
DATASET_TEXT	references the string of the **SET** associated with the item (can only be used if the item has a **SET**)
STRING_*	see section st_format_command on page 54
CURRENT_*	references the current time, date or datetime
INPUT	references the user input of the field that called the function
SOURCE	references structure of data item, which was dragged or connected. (see example on page 128)
SOURCE2	references second structure of data item, which was connected (see section Navigator Diagram page 131).
THIS	references structure of data item, which called the function. (see Example 2 on page 238, 128)
BASE	references structure of called function. (see Example 3 on page 239)

job_data_identifier_options

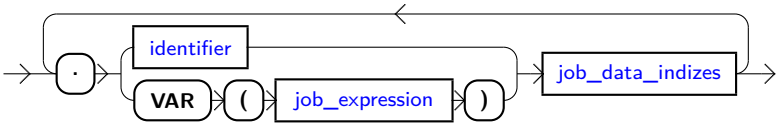


statements	description
`indexes`	references a specific element of a (possibly multidimensional) array
`struct_reference_list`	reference a specific element of a structure (i.E. motor.weight)
@	reference an element in a specific **CYCLE** default is current **CYCLE**

job_data_indizes

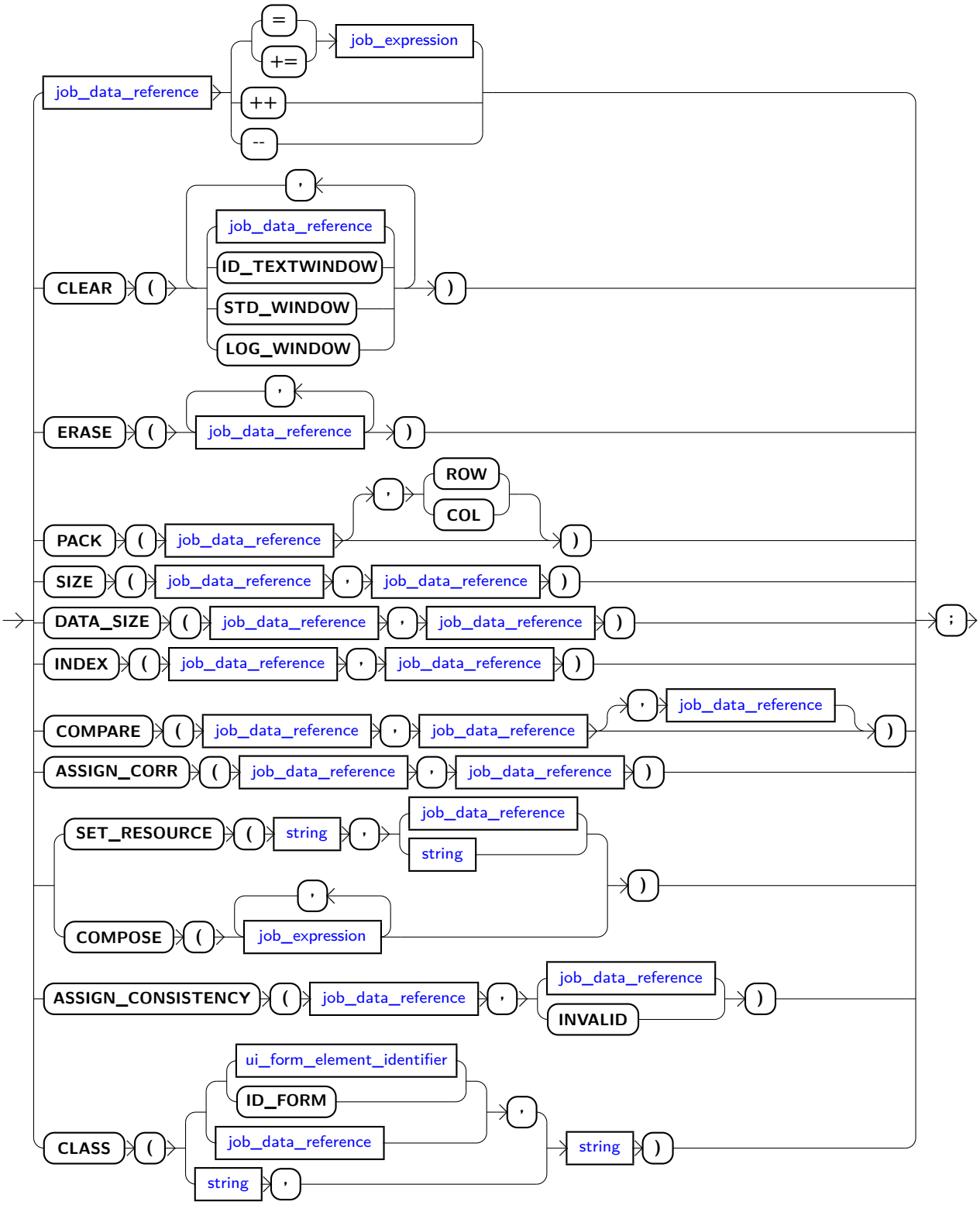


job_data_struct_reference_list



statements	description
VAR(string-item)	references a datapool item element. String dataitem contains the identifier of a data item. By changing the contents of string-item at runtime, specified datapool item is referenced. Note: only one **VAR** is allowed in a `job_data_reference`.

data_statement

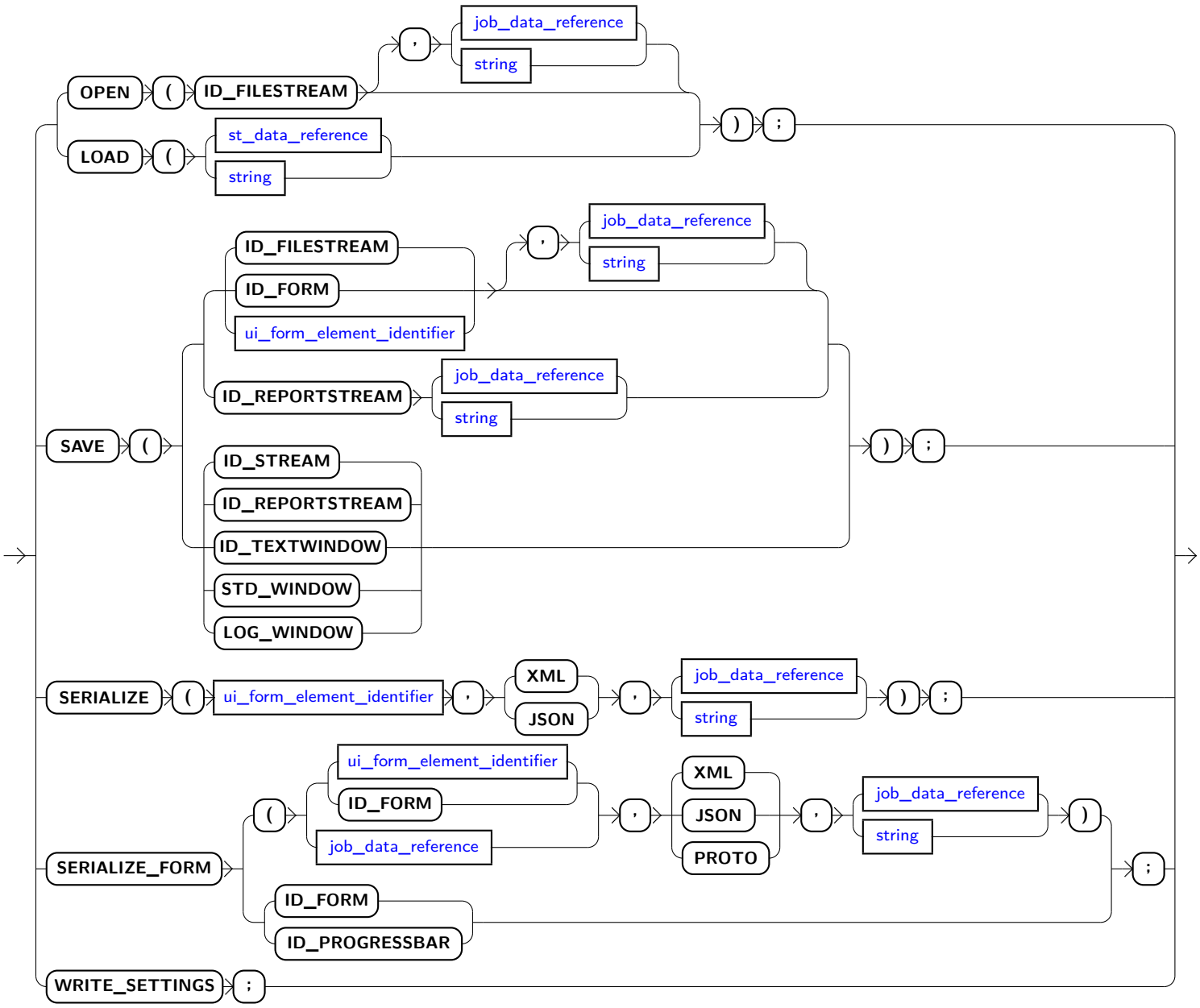


data statement	description
`job_data_reference`	references a (possibly indexed) datapool element.
CLEAR	clears all values of the specified datapool item elements.

ID_TEXTWINDOW	clears the content of the text window. (see section Text-Window on page 134)
STD_WINDOW	clears the content of the text window **STD_WINDOW**. (see section Text-Window on page 134)
LOG_WINDOW	clears the content of the text window **LOG_WINDOW**. (see section Text-Window on page 134)
ERASE	clears all values and attributes (color, etc.) of the specified datapool item elements.
PACK	removes empty entries `job_data_reference` should have wildcard(s) (nothing is done otherwise). The wildcard(s) specify the dimension on which **PACK** operates.
ROW	(=default) with two wildcards, empty rows (first wildcard) are removed.
COL	with two wildcards, empty columns (second wildcard) are removed.
SIZE	stores the number of elements of the first argument into the second. When the first argument has multiple dimentions (on the last level), the second stores a list of sizes: data.value[2,3] = 0; **SIZE**(data.value, size); -> size[0] is 3, size[1] is 4. The first argument may have wildcards. The second argument gets the size of every wildcard: **SIZE**(data[*].value[*,*], size); -> size[0] is size of data, size[1] is size of first dimension of data.value, size[2] is size of second dimension of data.value
DATA_SIZE	stores the size of the first argument (number of characters when written as JSON) into the second. **TRANSIENT** items are ignored (not counted) Can be used to know the data size of an object before if is sent to the database.
INDEX	**INDEX** is used to know the index of the field that called the function. stores up the index of the first argument into the second. The first argument may be **THIS**, **PARENT**(**THIS**), **PARENT**(**PARENT**(**THIS**)), etc., **BASE**. Example: data[2].value[3] was changed on the GUI and a function was called. After `INDEX(THIS, idx);`, idx is 2. See data_expression on page 225 and function_expression on page 228 for other **INDEX** expressions.

COMPARE	compare structure data. All arguments must be of the same data type (usually a Structure). With a third argument, the second and third arguments are compared. Without a third argument, the values of the second argument in all cycles are compared. The result is written to the first argument. It can be used in • a NAVIGATOR with the option **COMPARE** (see section Navigator on page 123). • the compare dialog: fill COMPAREDIALOG struct and call MAP (COMPARE_DIALOG);
ASSIGN_CORR	assign corresponding elements from second argument structure to first argument structure.
SET_RESOURCE	Set value of a resource property (with name **string**). The value is written to the user resource file when the **INTENS** application is quit or when **WRITE_SETTINGS** is called. It can then be used when the **INTENS** application is started next time(s).
COMPOSE	Compose string: write values into string. Needed for translatable strings. The (format-)string (first argument) contains %1, %2, ... and the value(s) of the corresponding data reference is put in these places. Up to 15 values are implemented.
ASSIGN_CONSISTENCY	Assign the (value of the) second argument to the first. **INVALID**: clear the first arguments value. Then, clear all dependent results (see paragraph Dependency on page 52). Dependency checking is otherwise only done when a value is entered in the gui, but not with a normal assignment (a = b;).
CLASS	Set or change the class property of an ui element. When using qss stylesheets, this changes the style of the ui element.

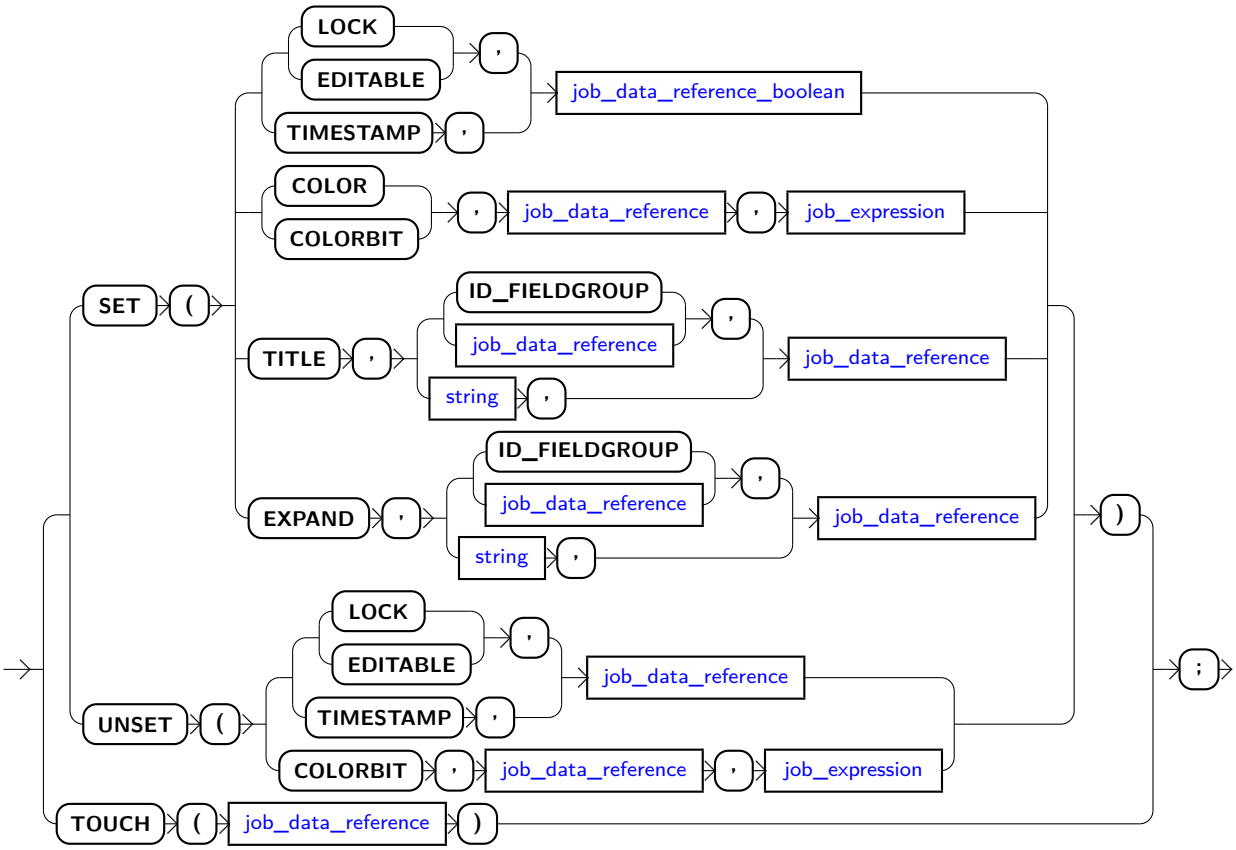
file_statement



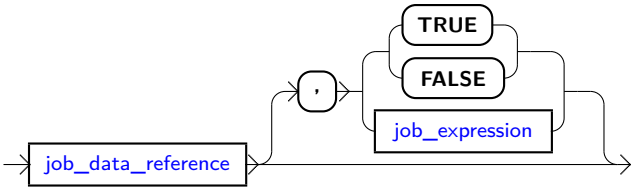
data statement	description
OPEN	Read a file and write its content to the **DATAPOOL**. The **STREAM** of the **FILESTREAM** (first argument) defines the files format and the variables to write to. Opens the file OPEN dialog unless the filename is given as the second argument.
`job data reference`	string item containing the filename No file selection dialog window is opened and the **OPEN** or **SAVE** operation is done using this filename. References a data item declared in the datapool (section Data Reference page 51).

LOAD	Parse the content of the **STRING** data reference.
SAVE	Saves the contens of the first argument to a file. Opens the file SAVE dialog window unless the filename is given as the second argument.
ID_STREAM	Saves the content of the stream. (see section STREAMER on page 52)
ID_REPORTSTREAM	Saves the content of the reportstream. (see section Report-streams on page 167)
ID_TEXTWINDOW	Saves the content of the text window. (see section Text-Window on page 134)
STD_WINDOW	Saves the content of the text window **STD_WINDOW**. (see section Text-Window on page 134)
LOG_WINDOW	Saves the content of the text window **LOG_WINDOW**. (see section Text-Window on page 134)
SERIALIZE	Writes the definition and content of a ui element to a file.
SERIALIZE_FORM	TODO
WRITE_SETTINGS	Write settings to user resource file now - as done when the **INTENS** application is quit.
XML	Use **XML** format.
JSON	Use **JSON** format.
PROTO	TODO.

set_statement



job_data_reference_boolean

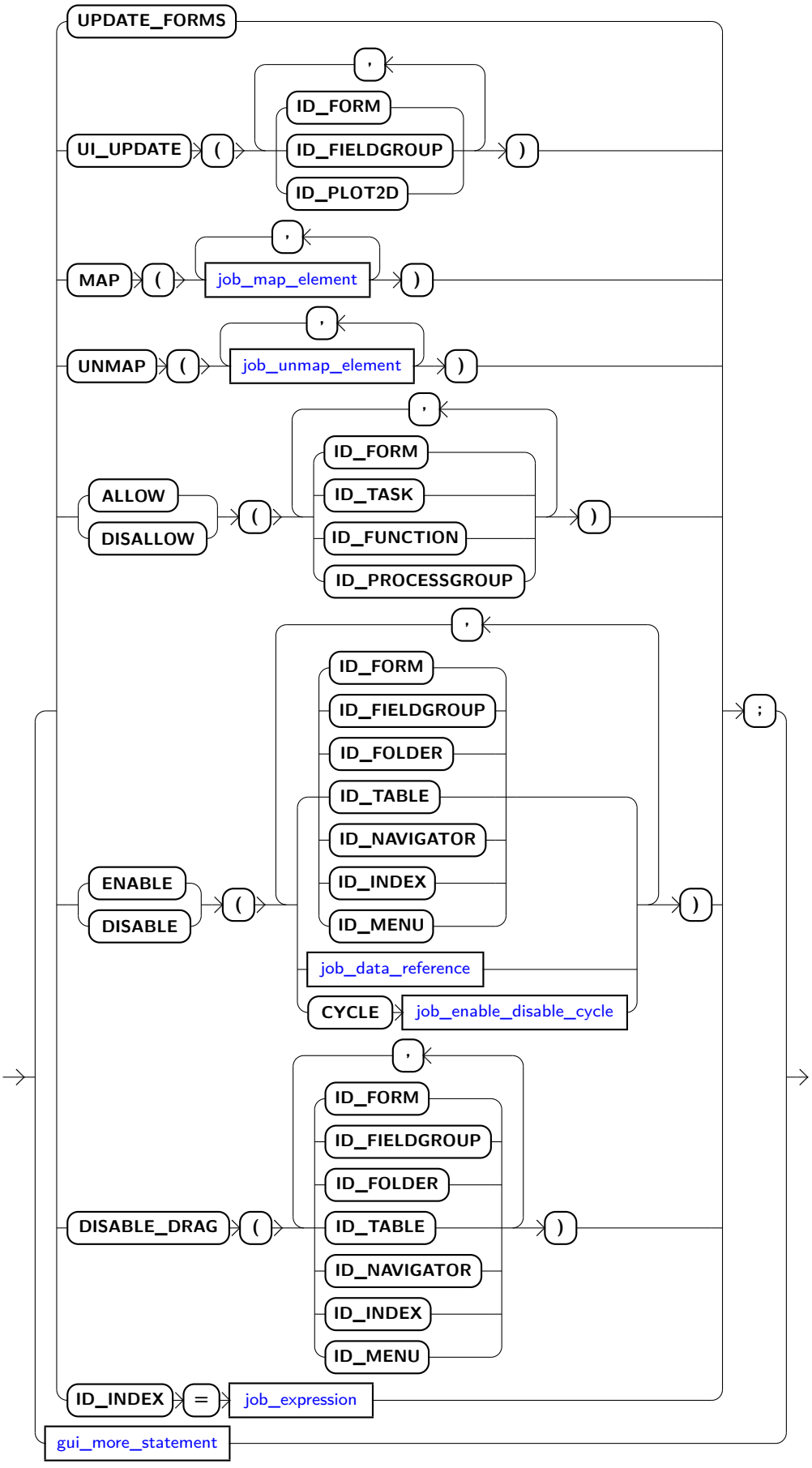


These statements are used to dynamically change the attributes of **DATAPOOL** elements or **FIELDGROUP**s at runtime.

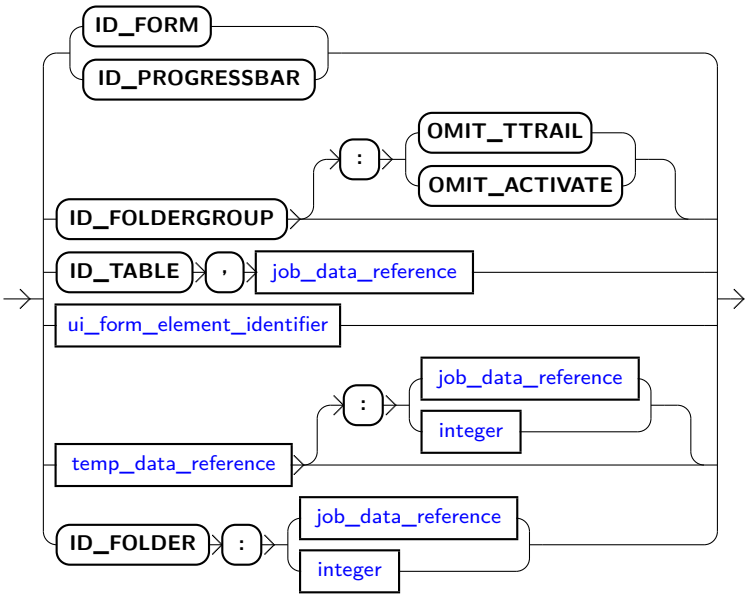
set statement	description
`bool value`	**TRUE** or **FALSE**. If omitted, TRUE is the default value. **SET** FALSE is equivalent to **UNSET**.
LOCK	See **LOCKABLE** attribute (section data_variable_attributes page 39).
EDITABLE	See **EDITABLE** attribute (section data_variable_attributes page 39).
TIMESTAMP	Efects the modify status. (See also DB_MANAGER MODIFY page **??**)
COLOR	SET the color to corresponding value 1-8 (or 1 - 255 with the **NO_COLORBIT** options). Value 0 is used to reset color settings.

	Previously set colors will be lost.
COLORBIT	**SET** the color to corresponding value 1-8.
	UNSETting higher level color (smaller value), the item is displayed whith next lower lever color.
TITLE	**SET** the title of the **FIELDGROUP** to the given string value (third argument).
EXPAND	Expand or collapse the **ACCORDION FIELDGROUP**.
TOUCH	Sets the attribute timestamp of `job_data_reference`. This causes the next gui update to update the gui elements. The data timestamp is not touched, so `MODIFIED ( job_data_reference )` stays at it was.

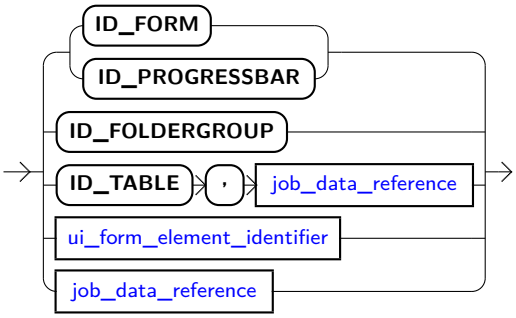
gui_statement



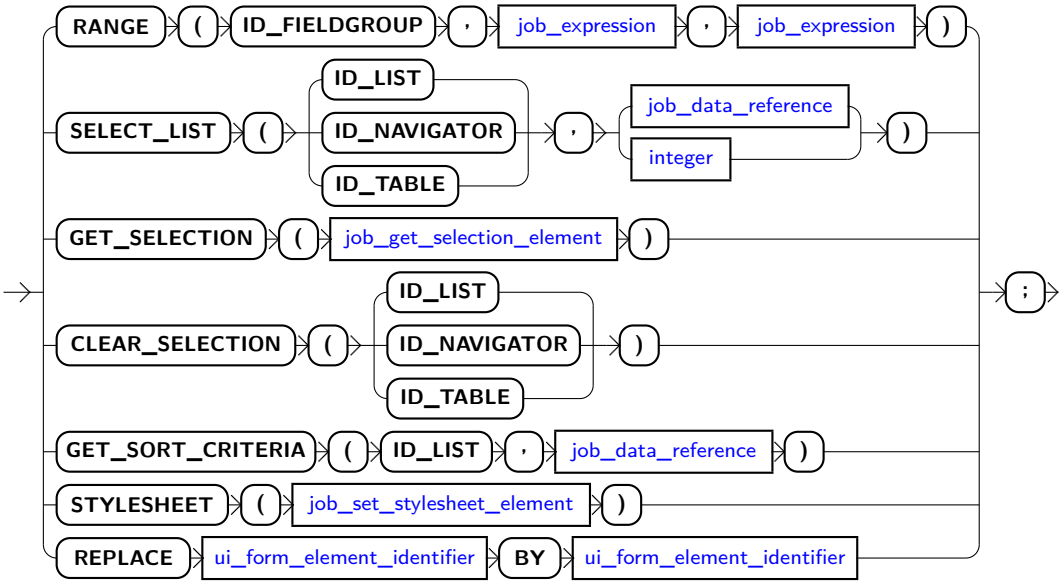
job_map_element



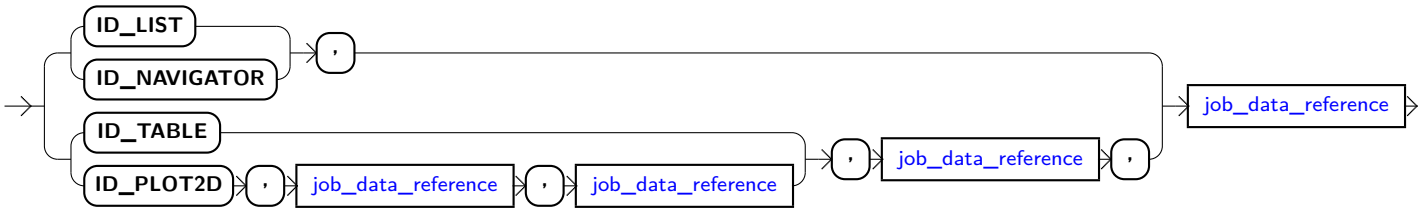
job_unmap_element



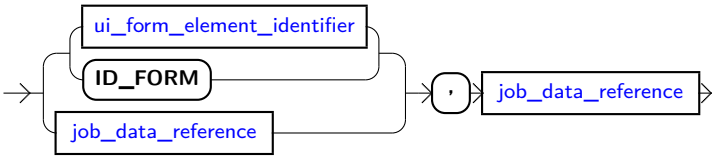
gui_more_statement



job_get_selection_element



job_set_stylesheet_element



gui statement	description
`identifier`	previously declared identifiers
UPDATE_FORMS	updates all forms
UI_UPDATE	Update a gui element.

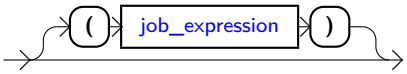
MAP	shows the form, progressbar, folder tab, table line or gui element referenced by the identifier. **OMIT_TTRAIL**: The folder tab mapping is not handled by transactions (ABORT, UNDO, ...). **OMIT_ACTIVATE**: The folder tab button is visible again (after an **UNMAP**), but not activated. `temp_data_reference` : The value of a string variable is used as the identifier. This can be used to map something different in different situations. To map a folder tab, use its **ID_FOLDERGROUP** or the identifier of the **FOLDER**, followed by a `:` and the tab index (as a integer or a variable) To map a table line, the second argument is the data reference shown in that table line, including the wildcards, as given in the **TABLE** definition.
UNMAP	closes (form, progressbar) or hides (folder tab, table line, gui element) specified element. `temp_data_reference` : The value of a string variable is used as the identifier. This can be used to map / unmap something different in different situations. It is possible to unmap (hide) menu entries. Here are a few examples: • ”File->Open” • ”File->Save” • ”File->Print” • ”Option” • ”Help->Copyright”
ALLOW	mapping, opening or execution of specified identifier is allowed.
DISALLOW	mapping, opening or execution of specified identifier is not allowed (Menu buttons are gray).
ENABLE	input fields (data items) or menu items are enabled (editable, choosable).
DISABLE	input fields (data items) or menu items are disabled (no entries are possible. Menu buttons are gray).
CYCLE	Disable or enable all input fields of a cycle.
ID_INDEX	previously declared index-object (page 95) is set to expression.
RANGE	Set range of previously declared **FIELDGROUP** (with option **TABLESIZE**) (see section ui_fieldgroup_option on page 78) to the interval [first job_expression, second job_expression]. This may also reduce the number of rows/lines shown to less than **TABLESIZE** if the **FIELDGROUP** has the option **STRETCH**.

SELECT_LIST	Select the line(s) defined in the second argument (array with integer index(es)) in the **LIST**, the **TABLE** or the **NAVIGATOR**.
GET_SELECTION	**LIST** and **NAVIGATOR**: write the selected row(s) index(es) to the second argument (array with integer index(es)). **TABLE**: write the selected cell(s) row and column index(es) to the second (rows) and third (cols) arguments (array with integer index(es)). When (at least) one complete row and/or column is selected, the index(es) of all completely selected row(s) and/or column(s) are writen to the second (rows) and third (cols). **PLOT2D**: write the coordinates (x and y), the y-axis type (1 or 2) and the y-axis title of the selected points to the arguments (x, y, axisType, axisLabel).
CLEAR_SELECTION	**LIST**, **TABLE** and **NAVIGATOR**: unselect all elements
GET_SORT_CRITERIA	**LIST**: write the current sort criteria to the variable (job_data_reference)

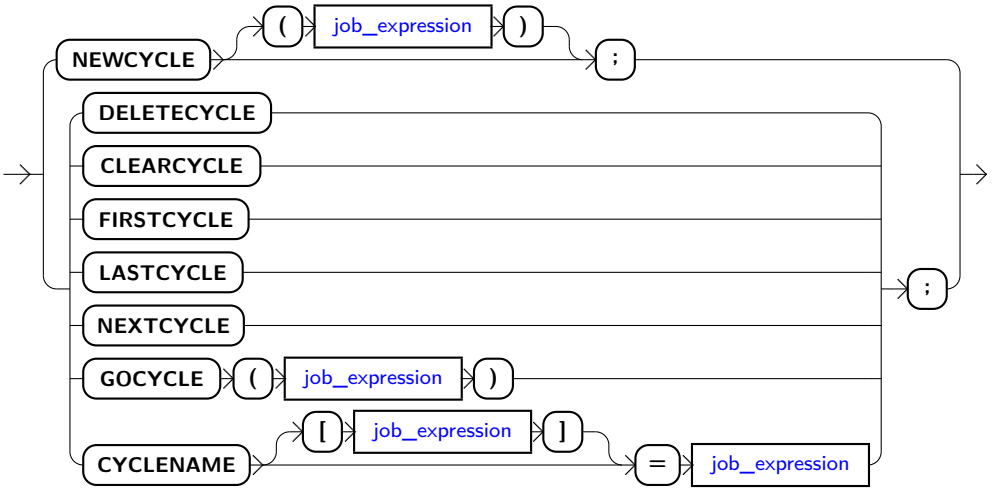
STYLESHEET	To customize their look, you can set the Qt Style Sheet (qss) string of a gui element or a data reference. The second argument (a data reference), contains the string. With a data reference, the qss string is passed to the gui field showing that data reference.
	Example (gui element): use a picture as the background of a **PLOT2D**:
	`"QwtPlotCanvas{background-repeat: no-repeat; border-image: url(./bitmaps/fans/xy.png) 0 0 0 0 stretch stretch;}"`
	Example (data reference): show a data item with bold font:
	`"QLineEdit{font-weight:bold;}"`

REPLACE	Replace the first gui element by the second.

job_enable_disable_cycle



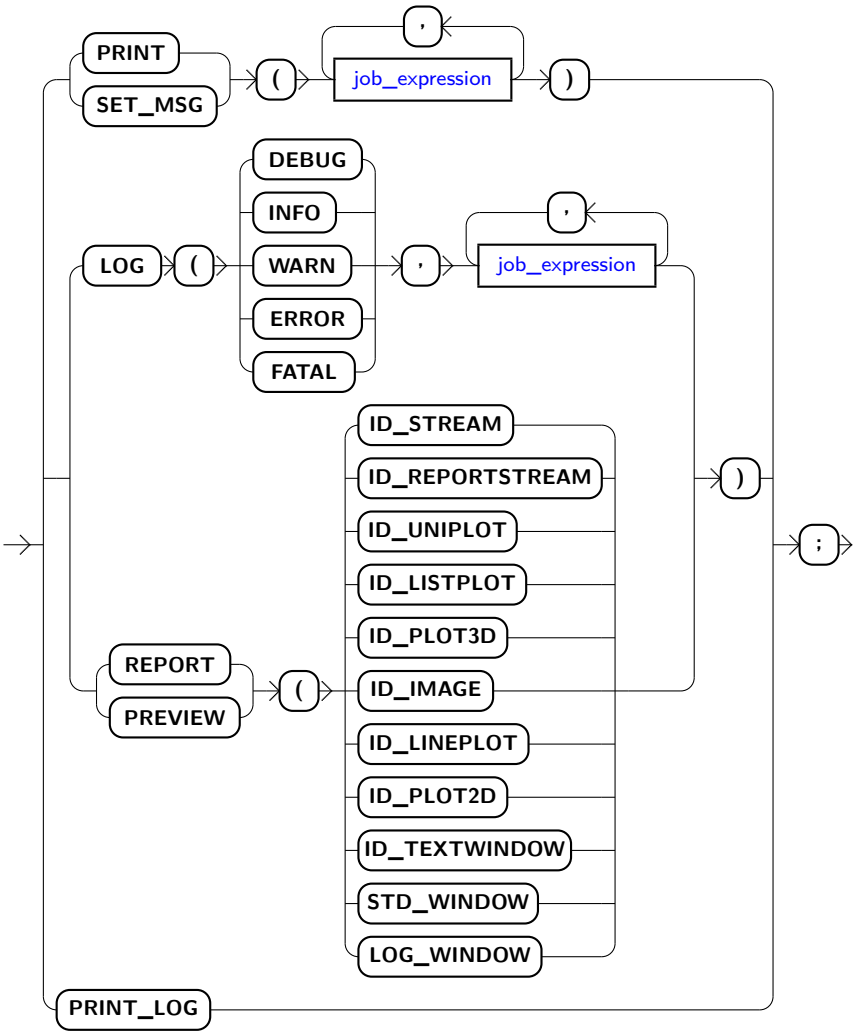
cycle_statement



The cycle statement is used to make a copy of the entire datapool, switch between cycles, delete a cycle, etc.

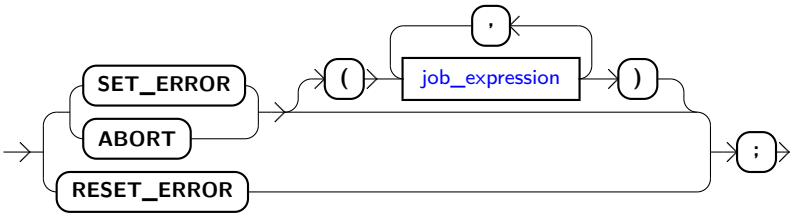
cycle statement	description
FIRSTCYCLE	switches to the first cycle
LASTCYCLE	switches to the last cycle
NEXTCYCLE	switches to the next cycle. If already in last cycle, a new cycle is created.
CLEARCYCLE	clears all values of recent cycle whithout removing it. And executes INIT-function.
DELETECYCLE	deletes current cycle (as long as another one remains)
GOCYCLE	switches to cycle no. '(int)'
NEWCYCLE	creates a new cycle (with optional name)
CYCLENAME	sets name of current or specified cycle

print_statement



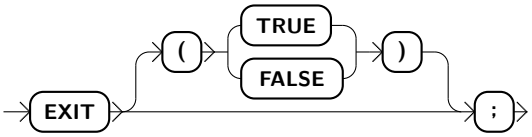
print statement	description
PRINT	Prints its arguments to **LOG_WINDOW**.
SET_MSG	Text to be printed after execution.
LOG	Log a message with the given level.
REPORT	Opens the printer dialog to print, save or preview the specified reportstream.
PREVIEW	Creates and opens the report (i.E. using a pdf reader).

error_statement



error statement	description
SET_ERROR	prints its arguments to a dialog window.
ABORT	works like **SET_ERROR** and aborts the running task immediately.
RESET_ERROR	resets error setting.

exit_statement

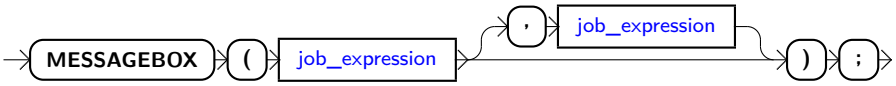


exit statement	description
EXIT	quit INTENS execution.

system_statement

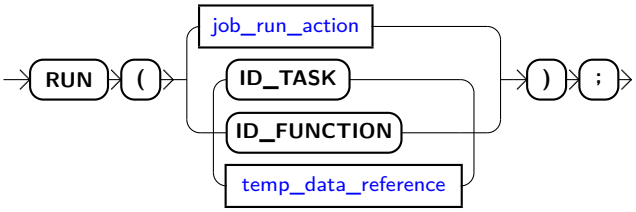
system statement	description
BEEP	play a system beep.

messagebox_statement

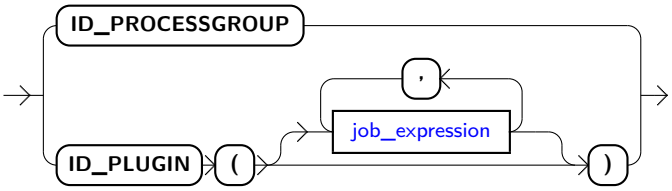


messagebox statement	description
MESSAGEBOX	Displays a messagebox with an OK-Button. The first job_expression provides the message to be displayed. You can optionally provide the title as the second job_expression. The default title is “Inform”.

run_statement



job_run_action

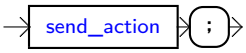


run statement	description
RUN	Starts the process group, function, task referenced by the identifier
`temp_data_reference`	The value of a string variable is used as the identifier. This can be used to run something different in different situations.

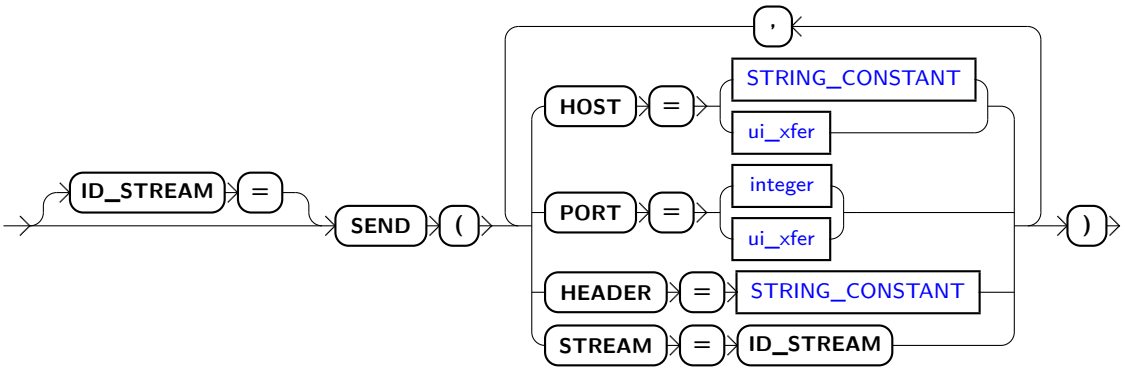
return_statement

return statement	description
RETURN	exits the function back to calling instance.

send_statement



send_action



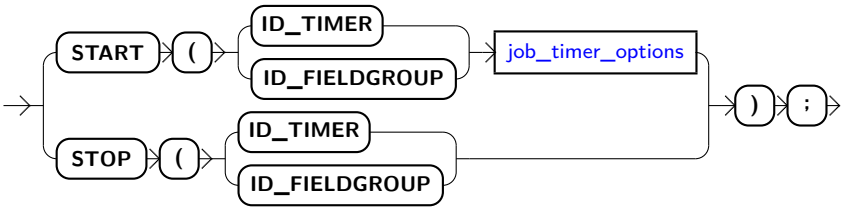
send statement	description
HOST	host name or ip address.
PORT	host port number.
HEADER	header string.
STREAM	data.

Send hhHEADERddddddddDATA using tcp to host:port.

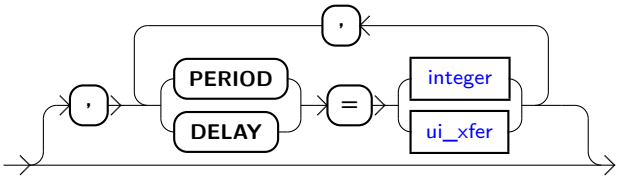
hh	number of characters of HEADER
dddddddd	number of characters of DATA

Timer statements are used to start and stop a timer. See Timer on page 177 and **MESSAGE_QUEUE PUBLISH - SUBSCRIBE** example with **TIMER** on page 184.

timer_statement

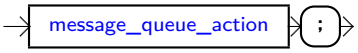


job_timer_options

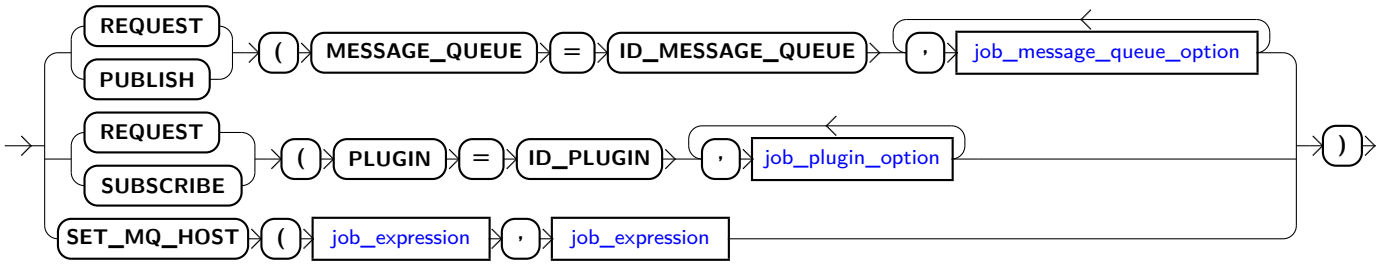


timer statement	description
START	Start the timer.
STOP	Stop the timer.
ID_TIMER	The timer to start or stop.
PERIOD = n	The timer calls its function every n seconds. Default is 60 seconds. **PERIOD** = 0: call the function once, without **DELAY**.
DELAY = n	The timer starts to call its function after n seconds. Default is 0: start to call the function after **PERIOD** seconds.

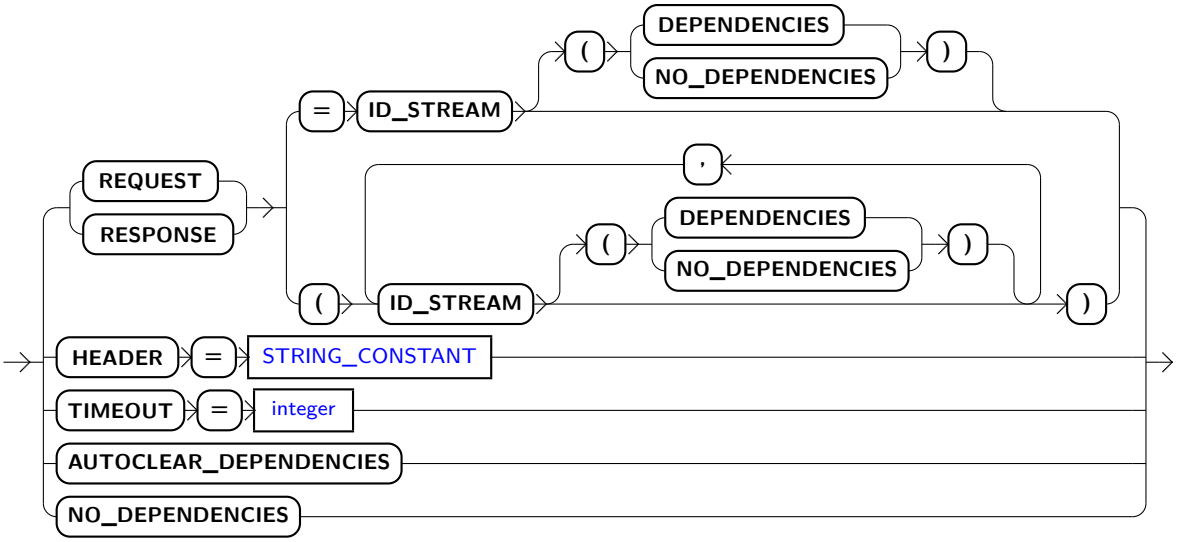
message_queue_statement



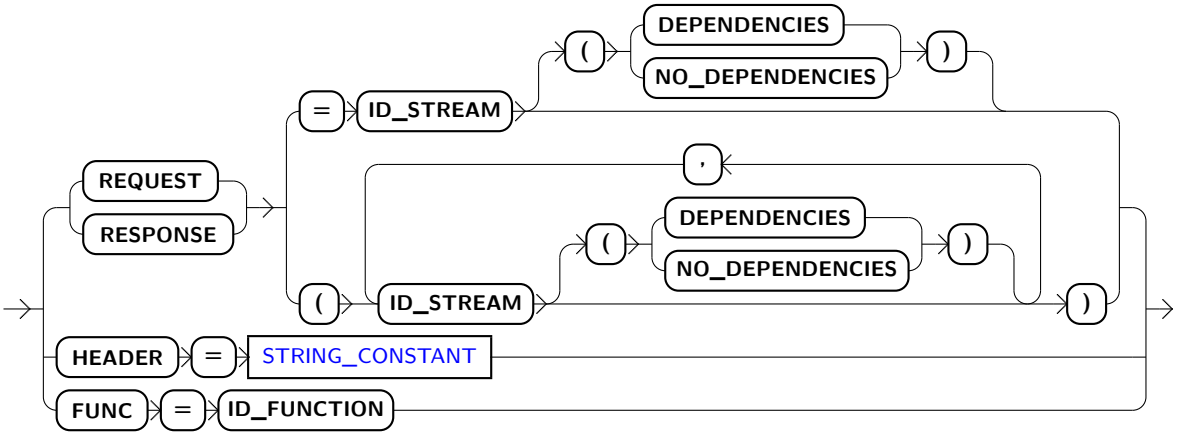
message_queue_action



job_message_queue_option



job_plugin_option



Message Queue statements are used to send data (a message) via ZeroMQ, to subscribe to a ZeroMQ publisher of a plugin or to set the **HOST**of a **SUBSCRIBE MESSAGE_QUEUE** (see section Message Queue on page 178). A message can be sent to another program using a

MESSAGE_QUEUE defined in the **OPERATOR**. And it can be sent to a **PLUGIN** defined in the **UI_MANAGER** (see section Plugin on page 161).

The **SUBSCRIBE** statement is used to subscribe to a **PLUGIN**s publisher.

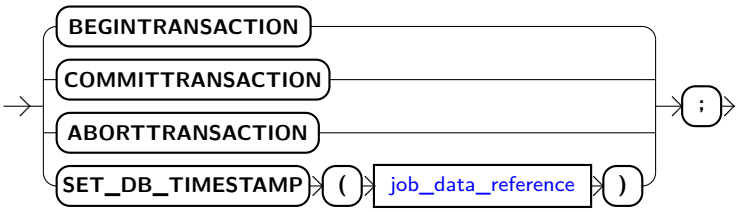
MESSAGE_QUEUE or **PLUGIN** as well as **HEADER** are mandatory. The other options can be given as needed.

See **MESSAGE_QUEUE PUBLISH - SUBSCRIBE** example with **TIMER** on page 184.

message queue statement	description
REQUEST	Send a request message.
PUBLISH	Publish a message.
SUBSCRIBE	Subscribe to a publisher of a plugin.
SET_MQ_HOST	Set the **HOST** (second argument) of a **SUBSCRIBE MESSAGE_QUEUE** (first argument). This can be done once for every **SUBSCRIBE MESSAGE_QUEUE**, but only when its **HOST** is set to "" (empty string) in the definition (in the **OPERATOR**). Setting the **HOST** makes the **MESSAGE_QUEUE** subscribe to the publisher.
MESSAGE_QUEUE	Send the message using this message queue.
PLUGIN	Send the message or subscribe to this plugin.
REQUEST	**REQUEST**: Stream(s) define the message(s) to send.
RESPONSE	**REQUEST**: The received message(s) is/are written to this/these stream(s). **SUBSCRIBE**: The received message(s) is/are written to this/these stream(s). **PUBLISH**: Stream(s) define the message(s) to publish.
DEPENDENCIES	**REQUEST**: This stream is used with dependencies. Default, unless commandline option defaultMessageQueueDependencies false is used. (see paragraph Dependency on page 52)
NO_DEPENDENCIES	**REQUEST**: This stream is not used with dependencies. (see paragraph Dependency on page 52)
HEADER	Message header: first part of the message. The receiver uses it to identify the message type.

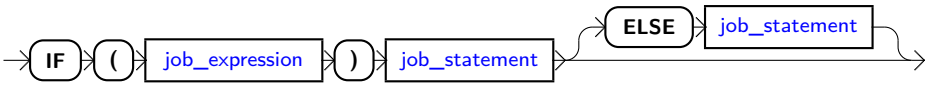
TIMEOUT	**REQUEST**: Number of seconds to wait for a connection to the receiver and its **RESPONSE**. Without this option, the **TIMEOUT** of the **MESSAGE_QUEUE** is used (default = 10 seconds). Set **TIMEOUT** to 0 to have no **TIMEOUT**.
AUTOCLEAR_DEPENDENCIES	Dependencies are cleared without user confirmation. (see paragraph Dependency on page 52)
FUNC	**SUBSCRIBE**: Function to call after receiving the message.

database_statement



database statement	description
BEGINTRANSACTION	TODO
COMMITTRANSACTION	TODO
ABORTTRANSACTION	TODO
SET_DB_TIMESTAMP	Sets the database timestamp of `job_data_reference` to the data timestamp. Afterwards, `MODIFIED ( job_data_reference )` returns false.

if_statement



if statement	description
IF	executes the `statement`, if the `expression` is not 0 (true)
ELSE	executes the `statement`, if the if-expression was 0 (false)

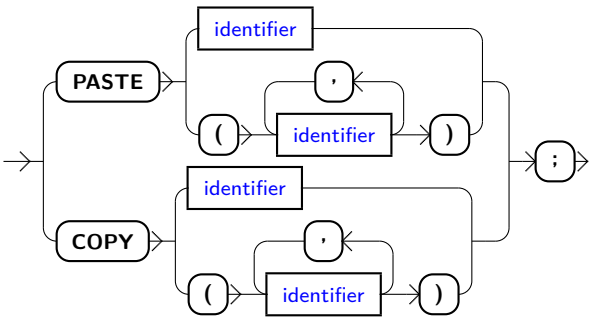
while_statement

while statement	description
WHILE	executes the `statement` while the `expression` is not 0 (true)

Example:

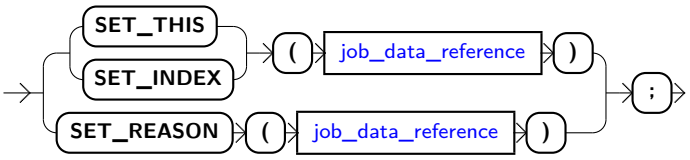
```
FUNCTIONS
  FUNC function_identifier {
    IF( data_item_identifier_1 == 99 ){
       data_item_identifier_2 = "Error message" ;
    }
    WHILE( data_item_identifier_1 <= 99 ){
       data_item_identifier_1++ ;
    }
  }
;
END FUNCTIONS;
```

copy_paste_statement



copy paste statement	description
COPY	Copies a streamable object (LIST, TABLE, STREAM) into the clipboard. Or copies a screenshot of a ui_form_element_identifier or a FORM into the clipboard.
PASTE	Pastes the clipboard to a streamable object (LIST, TABLE, STREAM).

set_func_statement



These statements are used to change function attributes.

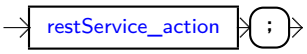
When a function is called, **INTENS** sets some attributes that can be used by the function to know why it was called. Sometimes, such a function should be called from another function and these attributes have to be changed so that the function does what is needed.

Use these statements carefully as they may change the behaviour of the rest of the function, including possible parent functions.

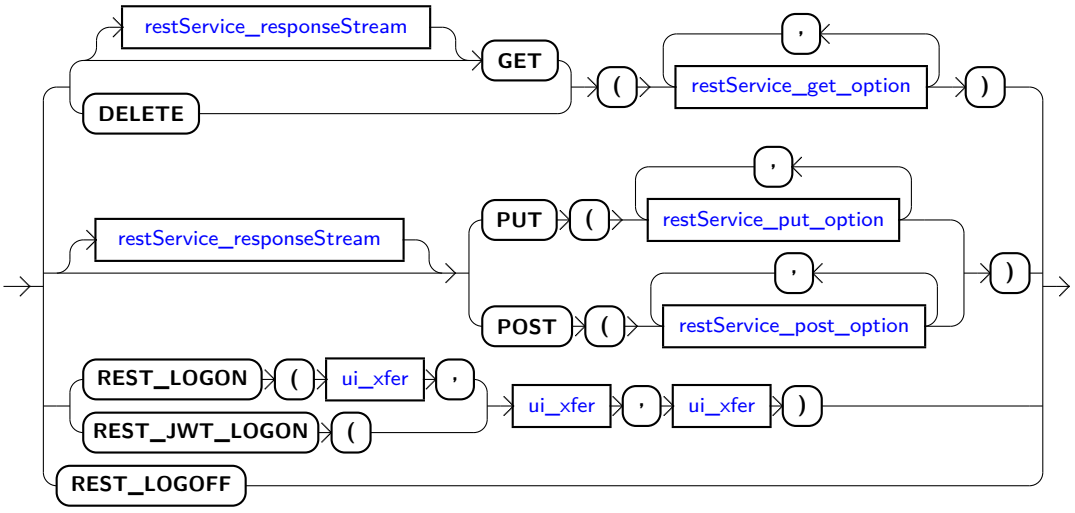
statement	description
SET_THIS	**THIS** now references `job_data_reference`. See job_data_reference on page 193.
SET_INDEX	Set **INDEX** to the value of `job_data_reference`. See data_expression on page 225.
SET_REASON	Set the **REASON**. To set **REASON_INPUT**, do `s = ``INPUT'';` `SET_REASON(s);`. See Reason on page 234.

Rest Service Intens knows the following actions to communicate with an INTENS db service (REST).

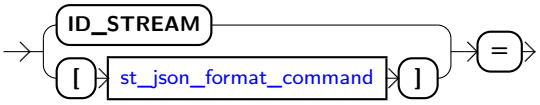
restService_statement



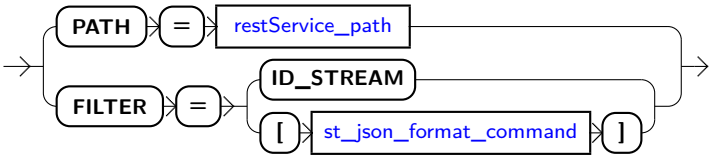
restService_action



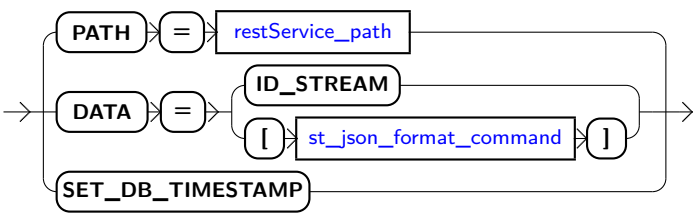
restService_responseStream



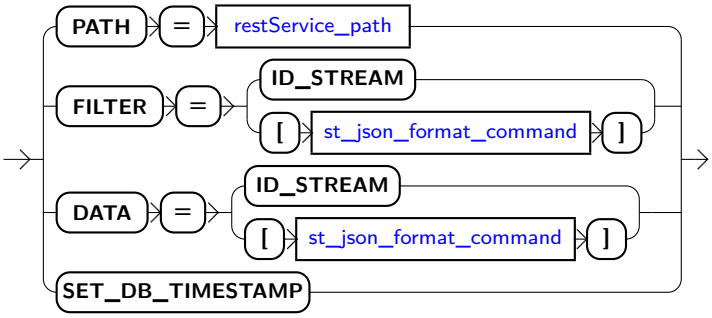
restService_get_option



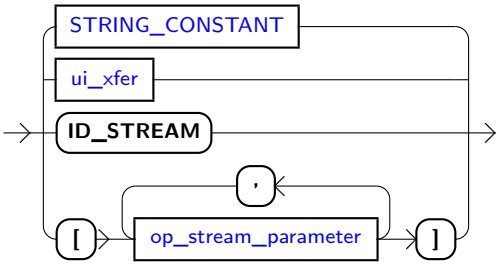
restService_put_option



restService_post_option



restService_path



restService statement	description
GET	Send a GET request.
DELETE	Send a DELETE request.
PUT	Send a PUT request.
POST	Send a POST request.
PATH	Second part of the *URL* the request is sent to. The first part is defined in the login dialog. It defaults to environment variable *REST_SERVICE_BASE* or to the first value of the environment variable *REST_SERVICE_BASE_LIST* (list of urls separated by ';'). **ID_STREAM**: use an **URL** stream for percent-encoding
DATA	Defines the stream that contains the data for **PUT** and **POST** requests.
FILTER	Defines the stream that contains filter parameters for **GET**, **DELETE** and **POST** requests. Valid entries are appended to the URL in the form name=value or name=(value[0],value[1], ...).
`ui xfer`	Data item (see section ui_xfer on page 69).
SET_DB_TIMESTAMP	Update the database timestamp of **DATA**.
REST_LOGON	Set rest credentials. The three arguments are base url, username and password.

REST_JWT_LOGON	Set rest credentials. The two arguments are base url and jwt (json web token).
REST_LOGOFF	Clear rest credentials. Login will appear with next **GET**, **PUT** or **DELETE**.

If an input stream is defined (**ID_STREAM** =), the results are written to it and the database timestamp of its data items is set.
Username and password are automatically asked for when a **GET**, **PUT** or **DELETE** request is called and the credentials are missing. They are kept in memory for later usage until the application is closed or **REST_LOGOFF** is called in a function. They are also written to the environment variable REST_SERVICE_AUTHHEADER, i.E. to be used in **BATCH** programms called from the application: base64 encoded "<username>:<password>".

Example:

```
FUNCTIONS
  FUNC
    getMotor {
      [motor] = GET( PATH="/components/123" );
    },
    saveMotor {
      [motor] = PUT( PATH="/components"
                   , DATA="motor_stream"
                   , SETDBTIMESTAMP );
    }
  ;
END FUNCTIONS;
```

Rest Service: Version control Sometimes, an INTENS application requires a **minimal database version** or the database requires a minimal INTENS **application version**. Login to a database is not allowed when these requirements are not met.

Here is how that version control works and how you can use it.

The **database version** is defined in a (single) component of type AppVersionCtrl (stored in the database) with the attribues

- app_major (mandatory)
- app_minor (optional, defaults to 0)
- app_patch (optional, defaults to 0)

The **application version** is defined using the **DATAPOOL** variables

- REST_SERVICE.APP_VERSION_MAJOR (mandatory)
- REST_SERVICE.APP_VERSION_MINOR (optional, defaults to 0)
- REST_SERVICE.APP_VERSION_PATCH (optional, defaults to 0)

The **minimal database version** is defined using the **DATAPOOL** variables

- REST_SERVICE.DB_VERSION_MAJOR (mandatory)
- REST_SERVICE.DB_VERSION_MINOR (optional, defaults to 0)
- REST_SERVICE.DB_VERSION_PATCH (optional, defaults to 0)

When none of these are set, there is no version control and login is allowed.

Another possibility to disable version control is to set the **DATAPOOL** variable

REST_SERVICE.DB_VERSION_IGNORE to a value > 0.

Otherwise, version control is active and login is not allowed in any of the following situations:

- **database version** or **application version** are not set
- **application version** $<$ **database version**
  database version defines the minimal **application version** required
- **minimal database version** is set and **database version** $<$ **minimal database version**
  minimal database version defines the minimal **database version** required

So, to use the version control, you have to

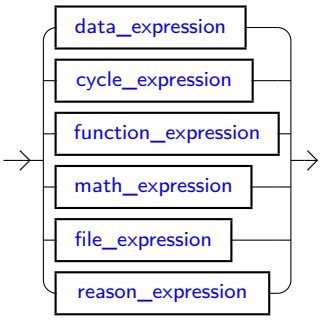
- require a minimal INTENS **application version**
  Define **database version** in the database and **application version** in the **DATAPOOL**. **application version** must not be lower.

and you can additionally

- require a **minimal database version**
  Define **minimal database version** in the **DATAPOOL**. **database version** must not be lower.

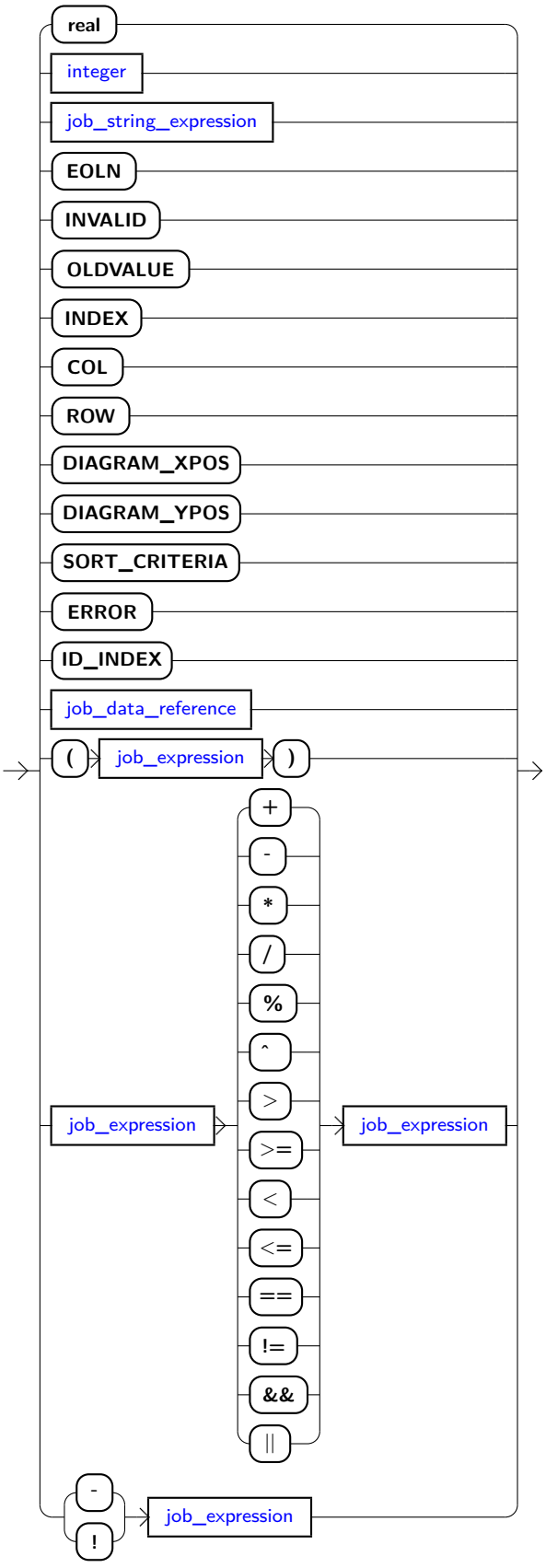
4.8.3 Expressions

job_expression



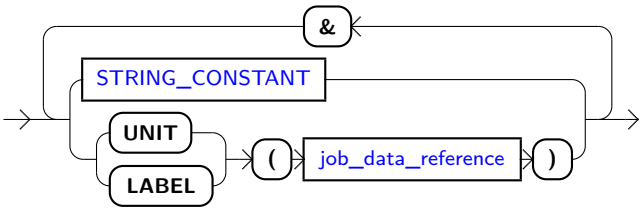
expression	description
`Data expression`	is called whithout any parameters. (Page 225)
`Cycle expression`	is called whithout any parameters. (Page 227)
`Function expression`	must be acompanied by parameters put in parethesis. (Page 228)
`Math expression`	calculates a value using parameter(s). (Page 231)
`Filename expression`	gives filename, basename or dirname of a parameter. (Page 233)
`Reason expression`	is used to test where the function itself is called from. (Page 234)

data_expression



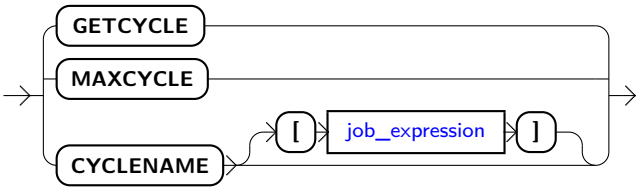
data expression	description
`real`	floating point number (constant)
`integer`	integer number (constant)
`string`	(see section String on page 21)
EOLN	represents an end of line character
INVALID	returns an invalid value
OLDVALUE	returns the old (previous) value of the changed data item
INDEX	represents the index of the field that called the function. Changing data[2].value[3], **INDEX** is set to 3. See data_statement on page 195 and function_expression on page 228 for other **INDEX** expressions.
COL	returns the column index of the field that called the function
ROW	returns the row index of the field that called the function
DIAGRAM_XPOS	returns the x coordinate of the diagram element that called the function
DIAGRAM_YPOS	returns the y coordinate of the diagram element that called the function
ERROR	returns True if a error has occured
ID_INDEX	references index-object previously declared in ui_manager (page 95)
-	inverts the sign of job_expression (`j = -i;`)
!	represents a logical **NOT**

job_string_expression



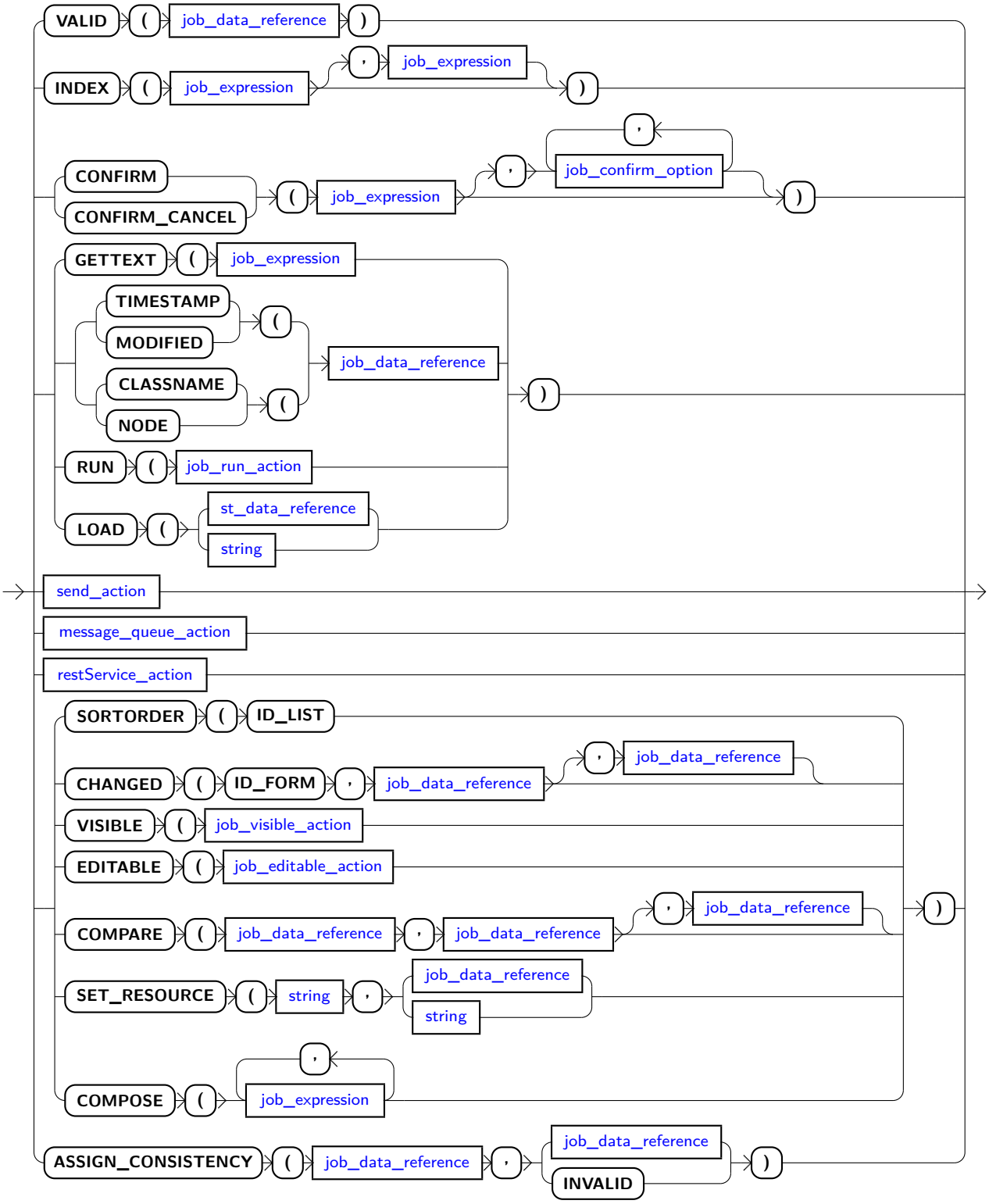
string expression	description
LABEL	returns previously defined label of data-item. (section data_item_options page 40)
UNIT	returns previously defined units of data-item. (section data_item_options page 40)

cycle_expression



cycle expression	description
GETCYCLE	returns current cycle no.
MAXCYCLE	returns ammount of cycles
CYCLENAME	returns name of current cycle
CYCLENAME[i]	returns name of cycle `i`.

function_expression

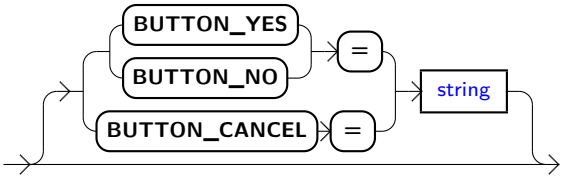


function expression	description
VALID	returns true if the expression is valid

INDEX	returns the index of the field that called the function. The first argument defines the dimension (counted from right to left - the last dimension being dimension 0), the optional second argument defines the structLevel (defaults to the last (highest)). Example: changing data[1,2].value[3,4]: **INDEX**(1) is 3 (same as **INDEX**(1,1) **INDEX**(1,0) is 1 **INDEX**(0,0) is 2 **INDEX**(0,1) is 4 (same as **INDEX**(0) or **INDEX**) See data_statement on page 195 and data_expression on page 225 for other **INDEX** expressions.
CONFIRM	displays a messagebox with a yes and no button. It evaluates to 1(yes) or 0(no).
CONFIRM_CANCEL	displays a messagebox with a yes, no and cancel button. It evaluates to 1(yes), 0(no) or **INVALID**(cancel).
GETTEXT	displays a dialog with the given text and an input field where the user enters a text.
TIMESTAMP	Retruns the data timestamp of **job_data_reference**.
MODIFIED	Is **job_data_reference** modified (compares db and data timestamps)?
CLASSNAME	returns previously defined classname of data-item. (section data_variable_attributes page 39)
NODE	returns node name of argument. Useful with **INPUT**, **THIS** or **BASE**. (see Example 3 on page 239)
RUN	run the argument.
LOAD	parse the content of the **STRING** data reference
SORTORDER	returns the current **SORTORDER** of the **LIST**. The **SORTORDER** is the number of the column used to sort the list, starting with 1. Hidden columns are counted. Descending sort returns a negative number. Defaults to 0.
CHANGED	Has any data item that shows in the **FORM** changed after timestamp (stored in first **job_data_reference**)? With a second **job_data_reference**, only data items that belong to that struct object are checked. **TRANSIENT** data items are ignored. When **FORM** is the **MAIN FORM**, data items in all **FORM**s are checked.
VISIBLE	returns true if the **FORM** or the **FOLDER** group is visible (MAPPED).
EDITABLE	returns true if the variable or **FIELDGROUP** is editable.

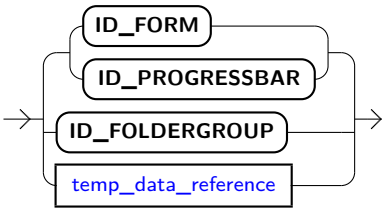
COMPARE	see **COMPARE** function statement on page 195
SET_RESOURCE	Set value of a resource property (with name **string**). The value is written to the user resource file when the **INTENS** application is quit or when **WRITE_SETTINGS** is called. It can then be used when the **INTENS** application is started next time(s).
COMPOSE	Compose string: write values into string. Needed for translatable strings. The (format-)string (first argument) contains %1, %2, ... and the values of the corresponding data references are put in these places. Up to 15 values are implemented.
ASSIGN_CONSISTENCY	Assign the (value of the) second argument to the first. **INVALID**: clear the first arguments value. Then, clear all dependent results (see paragraph Dependency on page 52). Dependency checking is otherwise only done when a value is entered in the gui, but not with a normal assignment (a = b;).

job_confirm_option



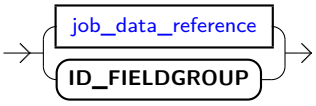
confirm option	description
BUTTON_YES	label shown on the yes button
BUTTON_NO	label shown on the no button
BUTTON_CANCEL	label shown on the cancel button

job_visible_action



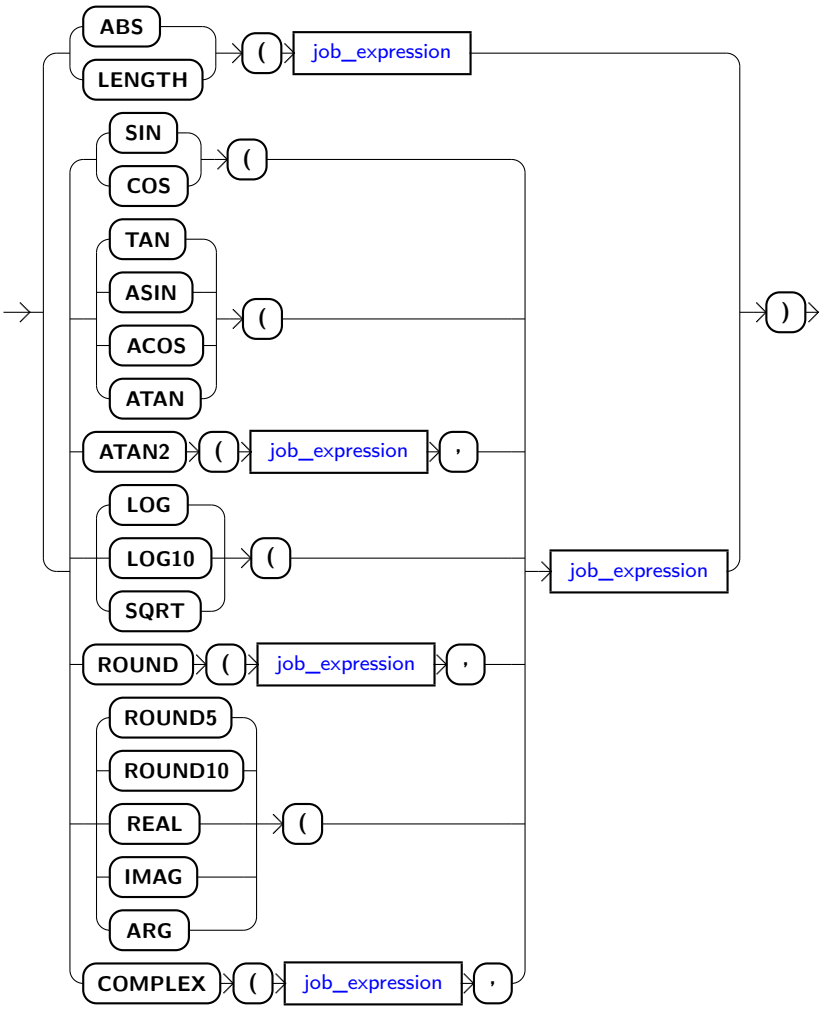
	description
`temp_data_reference`	The value of a string variable is used as the identifier. This can be used to check the visibility of something different in different situations.

job_editable_action



	description
`job_data_reference`	Check if a variable is editable.
ID_FIELDGROUP	Check if a **FIELDGROUP** is editable.

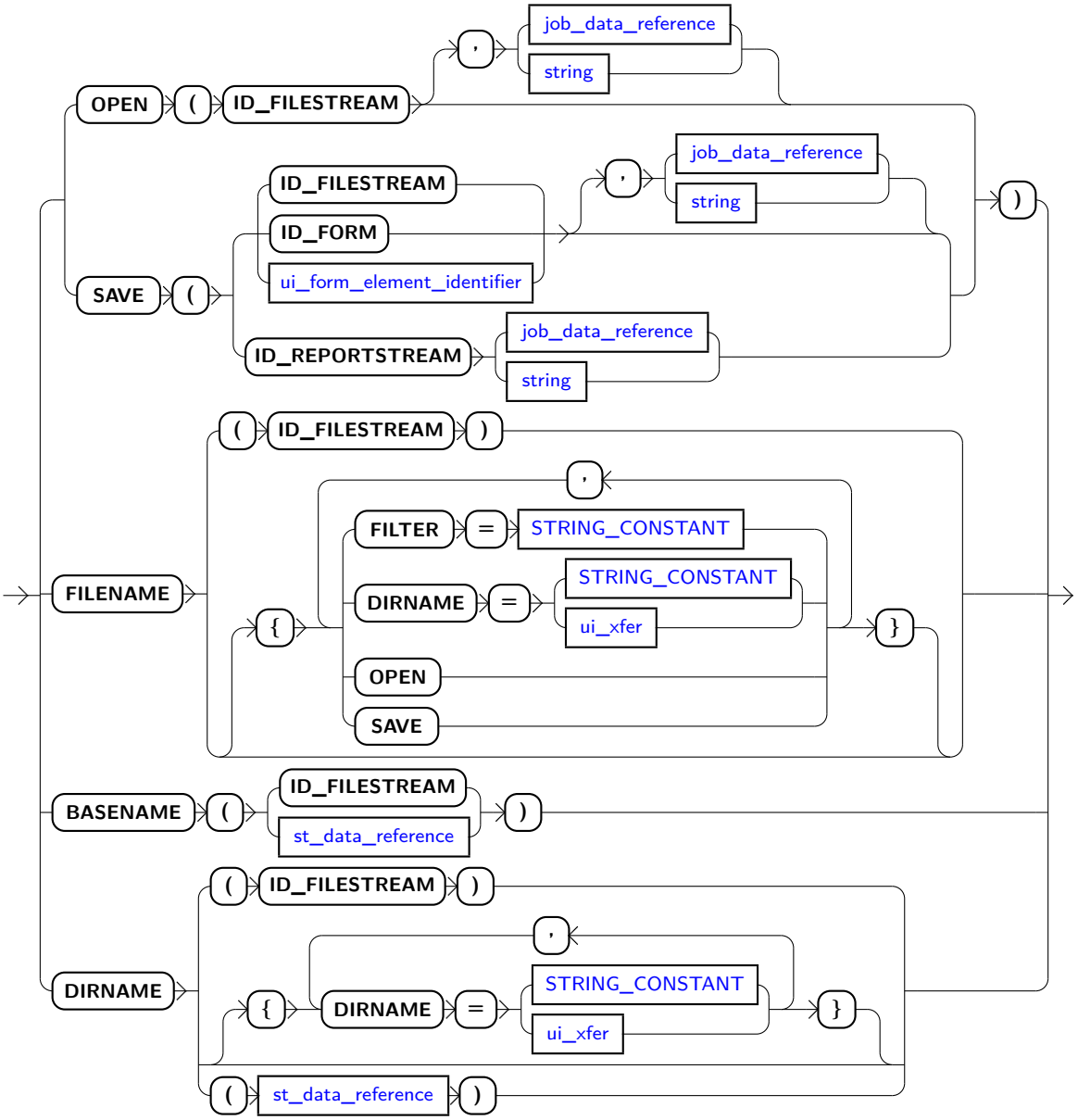
math_expression



math expression	description
ABS	returns the absolute-value of the expression
LENGTH	returns the string length of the expression
SIN	returns the sine-value of the expression (radians)
COS	returns the cosine-value of the expression (radians)

TAN	returns the tangent-value of the expression (radians)
ASIN	returns the arcsine-value (radians) of the expression
ACOS	returns the arccosine-value (radians) of the expression
ATAN	returns the arctangent-value (radians) of the expression
ATAN2	returns the arctangent-value (radians) of (y, x)
LENGTH	returns the string length of the expression
LOG	returns the natural logarithm of the expression
LOG10	returns the base 10 logarithm of the expression
SQRT	returns the nonnegative square root of the expression
ROUND	rounds the value (first expression) to n (second expression) digits after the decimal point
ROUND5	rounds the value to the next 20-th, i.E. to 5 cents
ROUND10	rounds the value to the next 10-th, i.E. to 10 cents
REAL	returns the real-value of the expression, or invalid if it is not real
IMAG	returns the imaginary quantity of a complex expression
ARG	returns the angle-value of the expression
COMPLEX	returns the complex-notation of 'expression, expression'

file_expression



math expression	description
OPEN (...)	read data from a file (see file_statement on page 198)
SAVE (...)	save to a file (see file_statement on page 198)
FILENAME (ID)	returns the entire file name (including directory) of previously used file stream.
BASENAME (ID)	returns the file name of previously used file stream.
DIRNAME (ID)	returns the directory path of previously used file stream.
FILENAME { ... }	open a file selection dialog and return the name of the selected file.
DIRNAME { ... }	open a directory selection dialog and return the name of the selected directory.

FILTER	specify the kind of files that should be shown. Only xml files: “XML (*.xml)” Image files (one filter): “Image files (*.png *.xpm *.jpg)” xml or json (two filters): “XML (*.xml);;JSON (*.json)”
DIRNAME = ...	default directory used in the select dialog
OPEN	use an open select dialog to get a filename
SAVE	use a save select dialog to get a filename
`ui xfer`	Data item (see section ui_xfer on page 69).

Reason

Functions may be used to handle input events on data fields. These functions will be invoked each time after a data item has been edited (see **FUNC** option in **DATAPOOL** data item declaration section Data Item on page 36).

Functions may also be used to handle **LIST** events. These functions will be invoked each time a line is selected, activated (mouse: double click or keyboard: return/enter) or unselected (see **FUNC** option in **LIST** declaration section List on page 92).

Functions may also be used to handle **FORM** events. These functions will be invoked each time a **FORM** is opened, closed or activated (see **FUNC** option in **FORM** declaration section Form on page 144).

Functions may also be used to handle **PLOT2D** events. These functions will be invoked each time a point or rectangle is selected in the **PLOT2D** (see graph option **FUNC** in **PLOT2D** declaration section Plot2d on page 110).

Data items may be displayed as a table by adding the **TABLESIZE** option to fieldgroups (**UI_MANAGER** fieldgroup options section Fieldgroup) or by defining a **TABLE** (**UI_MANAGER** table declaration Table).
Tables are shown with column- and row-titles. By pressing the right mouse button on it, a menu is popped up. Selecting an item will execute the data item assigned function first.

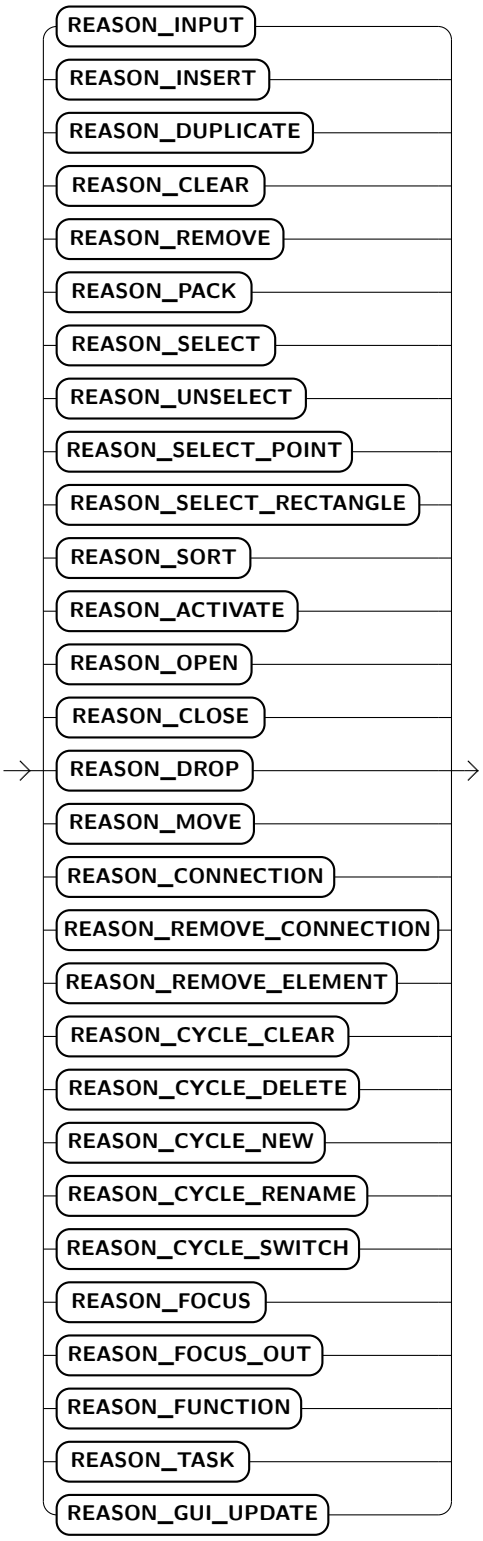
Inside a function, you can test the reason for calling it using **REASON** expressions:

```
FUNC a_func {
  ...
  IF(REASON_INPUT) {
    ...
  }
  ...
};
```

Sometimes, you may want to call a function with a different REASON. See set_func_statement on page 219 to know how to do so.

You may use one or more of the following REASON expressions:

reason_expression



reason expression	description
REASON_INPUT	the function was called after the assigned data item has been edited.
REASON_INSERT	insert has been choosen from table popup menu

REASON_DUPLICATE	duplicate has been choosen from table popup menu
REASON_CLEAR	clear has been choosen from table popup menu
REASON_REMOVE	remove has been choosen from table popup menu
REASON_PACK	pack has been choosen from table popup menu
REASON_SELECT	a list or navigator line was selected by a mouseclick (see section List List page 92) (see section Navigator Navigator page 122)
REASON_UNSELECT	a list or navigator line was unselected by a mouseclick (see section List List page 92) (see section Navigator Navigator page 122)
REASON_SELECT_POINT	a plot2d point is selected by a mouseclick (see paragraph UI Mode in section Plot2d on page 108)
REASON_SELECT_RECTANGLE	a plot2d rectangle is selected by a mouseclick (see paragraph UI Mode in section Plot2d on page 108)
REASON_ACTIVATE	a list line was activated by pressing the enter key (see section List page 92). a navigator entry was doubleclicked (see section Navigator page 122) a form is activated (becomes the active window) (see section Form page 140).
REASON_SORT	a list was sorted differently (see section List page 92).
REASON_OPEN	a form is opened (see section Form page 140).
REASON_CLOSE	a form is closed (see section Form page 140).
REASON_DROP	a navigator entry was drag'n'dropped (see section Navigator page 122) (see section Navigator Diagram page 131)
REASON_MOVE	a navigator (diagram) element was moved (see section Navigator Diagram page 131)
REASON_CONNECTION	two navigator (diagram) elements were connected (see section Navigator Diagram page 131)
REASON_REMOVE_CONNECTION	a navigator (diagram) connection was removed (see section Navigator Diagram page 131)
REASON_REMOVE_ELEMENT	a navigator (diagram) element was removed (see section Navigator Diagram page 131)

REASON_CYCLE_CLEAR	cycle was cleared. Used in **FUNCTION ON_CYCLE_EVENT** (see section job_function on page 190)
REASON_CYCLE_DELETE	cycle was deleted. Used in **FUNCTION ON_CYCLE_EVENT** (see section job_function on page 190)
REASON_CYCLE_NEW	cycle was new. Used in **FUNCTION ON_CYCLE_EVENT** (see section job_function on page 190)
REASON_CYCLE_RENAME	cycle was renamed. Used in **FUNCTION ON_CYCLE_EVENT** (see section job_function on page 190)
REASON_CYCLE_SWITCH	cycle was switched. Used in **FUNCTION ON_CYCLE_EVENT** (see section job_function on page 190)
REASON_FUNCTION	the function was called directly (i.E. from a menu)
REASON_TASK	the function was called from a **TASK** that was called directly (i.E. using its button)
REASON_GUI_UPDATE	the function was called by a gui update

4.8.4 Examples

Example 1

```
FUNCTIONS
  FUNC testminmax {
    IF( VALID(SnMin) && VALID(SnMax) ){
      IF( SnMin > SnMax ){
        SET_ERROR("Minimum value must be less than maximum!"
                  , EOLN );
      }
    }
  }
;
END FUNCTIONS;
```

Example 2

```
DESCRIPTION "Example of THIS";

DATAPOOL
  SET Set_identifier ("Pass" = 1, "Fail" = 2 );
  STRUCT Structure {
    REAL {EDITABLE}
      Data_item_choice { SET = Set_identifier,
                         FUNC = save_test_results };
    STRING
      Data_item_test_result;
  };
  Structure TestResults;
END DATAPOOL;

UI_MANAGER
  FIELDGROUP fieldgroup_identifier_1{TABLESIZE=5} (
    "Choose restult of each test-run:"
    TestResults [*].Data_item_choice
  );
  FORM form_identifier_1 {MAIN} (
    ( fieldgroup_identifier_1 )
  );
END UI_MANAGER;

FUNCTIONS
  FUNC save_test_results {
    IF ( THIS.Data_item_choice == 1 ) {
      THIS.Data_item_test_result = "Test result was OK";
    }
    ELSE {
      THIS.Data_item_test_result = "Test failed";
    }
  }
;
END FUNCTIONS;

END.
```

Example 3 When **INTENS** is started with the option --withInputStructFunc and an input variable does not have a function, the function of its struct (or the first parent of it that has a function) is called when the value is changed on the GUI. Inside that function, the following expressions can be used:

BASE references the structure with the called function. It is not set if the input variable has a function.

NODE(INPUT) can be used to know the name of the variable that was changed.

NODE(THIS) can be used to know the name of the structure of the variable that was changed.

NODE(BASE) can be used to know the name of the structure with the called function.

The following table shows the called functions, the references of **THIS** and **BASE** as well as the values of different **NODE**() expressions of the example, when variable is changed on the GUI:

				NODE()		
variable	function	**THIS**	**BASE**	**INPUT**	**THIS**	**BASE**
data1.p1.x	x_func	data1.p1	-	x	p1	-
data1.p1.y	p1_func	data1.p1	data1.p1	y	p1	p1
data1.p2.x	x_func	data1.p2	-	x	p2	-
data1.p2.y	data1_func	data1.p2	data1	y	p2	data1
data2.p1.x	x_func	data2.p1	-	x	p1	-
data2.p1.y	p1_func	data2.p1	data2.p1	y	p1	p1
data2.p2.x	x_func	data2.p2	-	x	p2	-
data2.p2.y	-	-	-	-	-	-

```
DESCRIPTION "Example of Input Struct Func and NODE, BASE";

DATAPOOL
  STRUCT Point {
    REAL {EDITABLE, SCALAR}
      x {FUNC=x_func}
    , y
    ;
  };

  STRUCT Data {
    Point
      p1 {FUNC=p1_func}
    , p2
    ;
  };

  Data
    data1 {FUNC=data1_func}
  , data2
  ;
END DATAPOOL;

UI_MANAGER
  FIELDGROUP main_fg (
    VOID         "x"          "y"
  , "data1.p1"   data1.p1.x   data1.p1.y
  , "data1.p2"   data1.p2.x   data1.p2.y
  , "data2.p1"   data2.p1.x   data2.p1.y
  , "data2.p2"   data2.p2.x   data2.p2.y
  );

  FORM main_form {MAIN, HIDE_CYCLE} (main_fg);
END UI_MANAGER;

...
```

```
...
FUNCTIONS
  FUNC print_func {
    PRINT("NODE(INPUT): ", NODE(INPUT),
          ", NODE(THIS): ", NODE(THIS),
          ", NODE(BASE): ", NODE(BASE), EOLN, EOLN);
  };

  FUNC x_func {
    PRINT("x_func:", EOLN);
    RUN(print_func);
  };

  FUNC p1_func {
    PRINT("p1_func:", EOLN);
    RUN(print_func);
  };

  FUNC data1_func {
    PRINT("data1_func:", EOLN);
    PRINT("BASE.p1:", BASE.p1, ", BASE.p1.x:", BASE.p1.x, EOLN);
    RUN(print_func);
  };
END FUNCTIONS;

END.
```

5 Unit Manager

The Unit Manager is an INTENS component that handles the scaling of values in the user interface automatically.

We recommend having values in their base units in the datapool, and then scale them for the user interface. This way, the values passed to calculation programs, stored in the database or in files are in base units. They are only scaled for the user interface.

5.1 Example

Throughout this section, let's use an example variable motor.Lls that was defined this way:

```
DATAPOOL
  STRUCT Motor: Component {
    ...
    REAL {EDITABLE, SCALAR}
      Lls {
        LABEL=_("Stator Leakage"),
        HELPTEXT=_("Stator leakage inductance"),
        UNIT="mH"
      },
    ...
  };
  Motor
    motor {
      LABEL=_("Motor"),
      PERSISTENT, SCALAR
    }
  ;
```

It has the value 0.001 H, shown as 1 mH in the user interface.

			DATAPOOL
Pole pairs			
Nom. Flux		Vs	
Stator Resistance		Ω	
Rotor Resistance		Ω	
Stator Leakage	1	mH	0.001
Magnetizing Inductance		mH	
Iron Resistance		Ω	
Rotor Leakage		mH	
Moment of Inertia		kg*m²	

Figure 21: Test application using Unit Manager

5.2 Without Unit Manager

Without the Unit Manager, the variable motor.Lls needs a scale factor *1e3 when shown in the user interface:

```
FIELDGROUP motor_properties_fg(
  ...
  LABEL(motor.Lls)       motor.Lls*1e3   UNIT(motor.Lls),
  ...
);
```

INTENS reads the value motor.Lls (0.001), multiplies it by 1000 (1.0) and shows this in the user interface. When the user enters a value, INTENS takes that value (1.0), divides it by 1000 (0.001) and stores it. See figure 22 on page 243.

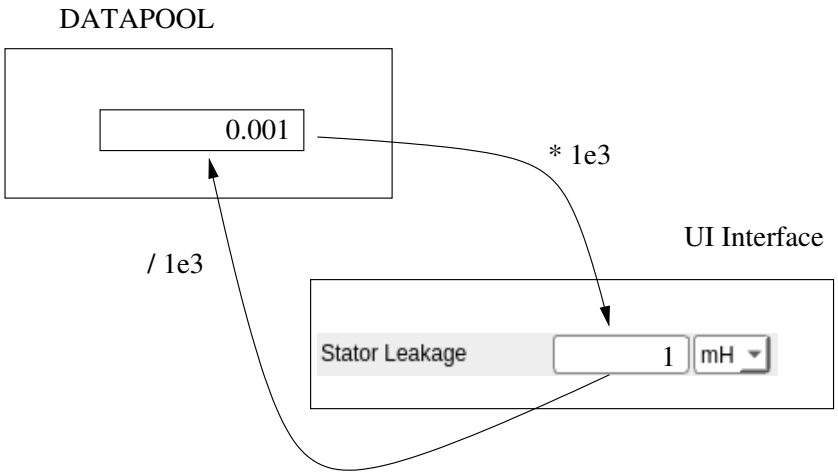


Figure 22: Unit Scaling

That brings the risk of forgetting to provide the (correct) scale factors in the UI_MANAGER, while the UNIT is defined in the DATAPOOL. This is where the Unit Manager comes in.

5.3 With Unit Manager

You activate the Unit Manager using the INTENS command line option `--unitManager`.

As mentioned, the Unit Manager handles the scaling of values in the user interface automatically.

You no longer provide the scale factor:

```
FIELDGROUP motor_properties_fg(
  ...
  LABEL(motor.Lls)       motor.Lls   UNIT(motor.Lls),
  ...
);
```

In fact, you could even keep the scale factors, as they are all ignored in Unit Manager mode.

As you defined the UNIT “mH” in the variable definition, INTENS knows that you want to see motor.Lls in “mH”.

5.4 Configuring the Units

INTENS knows a set of standard units. For example

```
[...,
 {"name": "H", "factor": 1.0,
 "derived": [
     {"name": "mH", "factor": 1e3},
     {"name": "μH", "factor": 1e6}
 ]},
 ...
]
```

"H" is the base unit. It has the scale factor 1.0 - as most base units do. And there are two derived units, "mH" and "μH", with their scaling factors. That information is enough for INTENS to automatically show motor.Lls (0.001 H) as 1 mH.

Now, your application likely needs more units than the small set of units known by INTENS. You can extend or change the units for the application using the built in Unit Manager form (menu Options > Unit Manager, see figure 23 on page 244.

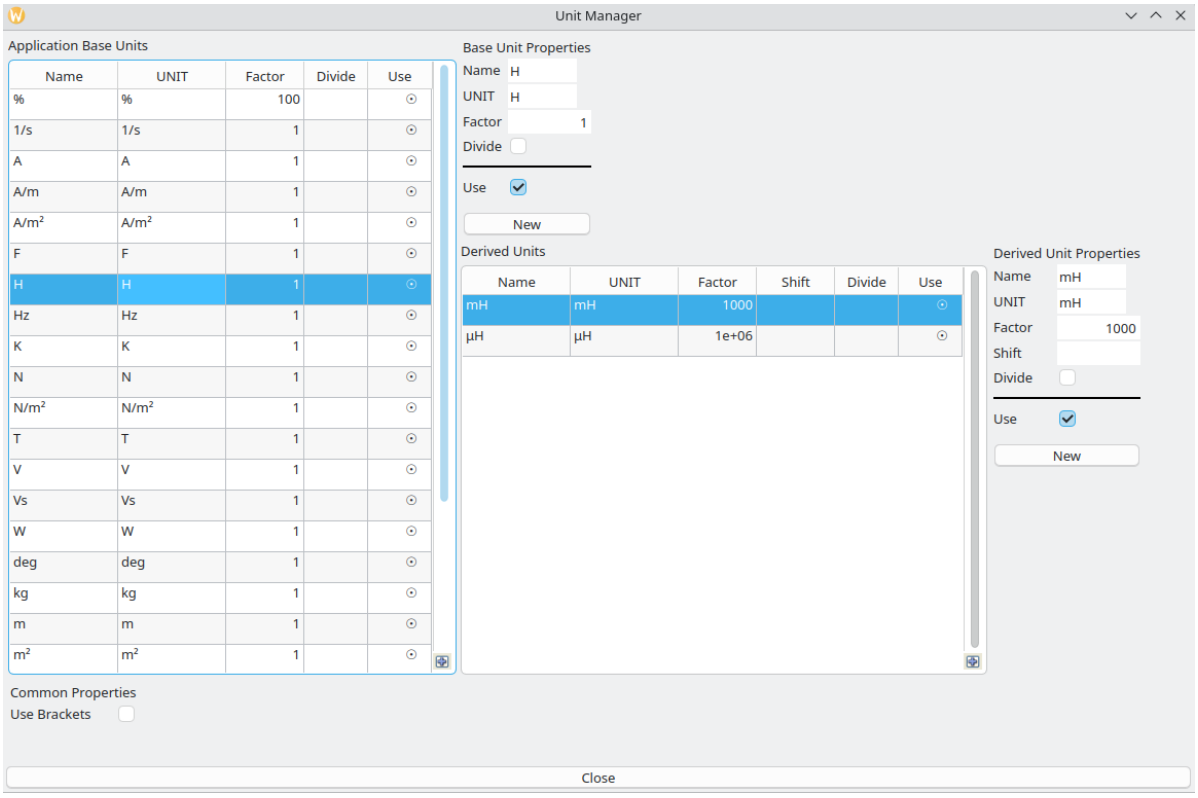


Figure 23: Unit Manager: Editor

You can add new base units and / or derived units as needed. When you close the form, the information is stored in the file <APPHOME>/etc/application_unit.json. You can also extend or change the units for the application by modifying this file directly. Just keep in mind that the file is read when the application starts, so you need to restart the application to see the effect of your modifications. All of this is normally done during the application development. Remember to distribute the file etc/application_unit.json with your application.

If a single value “mH” is preferred (i.e. no combobox for inductance units), the following configuration can be written to the file

`etc/application_unit.json`:

```
{"name": "H", "factor": 1.0, "use_set": 0,
"derived": [
    {"name": "mH", "factor": 1e3}
]}
```

The value “`use_set`” controls the content of the set. Only units with “`use_set`” value 1 (which is default) are included. use_set is labelled “Use” in the Unit Manager form.

5.5 Dynamically changing a Unit

Maybe a user wants to see different units in the user interface? You can tell INTENS to show the units as comboboxes instead of as labels. To do so, provide one of two values to the command line option unitManager:

- `--`unitManager=comboBox_hide_single
- `--`unitManager=comboBox_always

As mentioned above, there are two derivates, “mH” and “μH”, for the base unit “H”. So, INTENS will show a combobox with the three entries: H, mh, μH, and H as the base unit (ie. the unit of the values in the DATAPOOL).

“mH” will be selected by default, as that is what you defined in the variable definition.

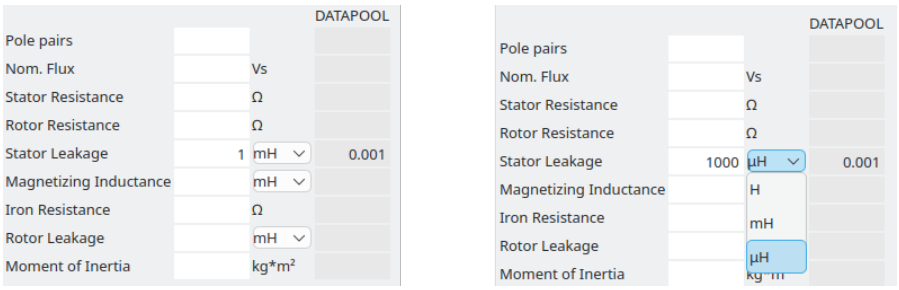


Figure 24: –unitManager=comboBox_hide_single

When you select a different unit, INTENS knows the new unit name and its scale factor and now shows motor.Lls as 0.001 H or 1000 μH.

This choice is remembered for you and when you start the INTENS application the next time, motor.Lls will be shown in the chosen unit. (INTENS stores these choices in the user resource file.)

If you prefer to see a combobox for all (known) units, use the command line option `--`unitManager=comboBox_always (see figure 25 on page 246.

Now, the unit “Vs” is also shown as a combobox, even though there is no other choice. The unit “Ω” is still shown as a label, because this unit is not configured yet.

You can use this mode if you prefer the look or to see what units you have not yet defined.

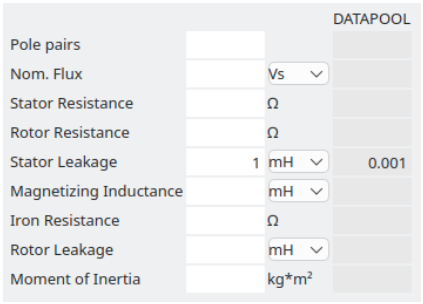


Figure 25: –unitManager=comboBox_always

6 Commandline options

An INTENS application is started calling intens followed by the name of the description file:
`intens description.des`
In addition, INTENS knows the following options. The list of INTENS commandline options can be printed using the command:
`intens --help`

--geometry <argument> defines the geometry of the main window (on linux only).
<argument> is <width>x<height>[+<xOffset>+<yOffset>]
obsolete / not implemented

--name <argument> obsolete / not implemented

--display <argument> obsolete / not implemented

--matlabnode <argument> TODO

--fontname <argument> TODO

--background <argument> obsolete / not implemented

--xfontlist <argument> TODO

--listFonts obsolete / not implemented

--init <argument> defines the filename of a datapool init file. That file is read at startup to initialize the datapool.
The file has one value per line:

Scalar speed 0.0

List # speed 1 1.0

Matrix # # speed 1 2 2.0

Struct points[3].x 3.0

--notitle hides the titlebar at startup. The titlebar shows

- a title (apptitle.text from the resource file)
- a subtitle (appsubtitle.text from the resource file)
- a left bitmap
- a right bitmap

(see Minimal Example on page 30). It is shown at startup unless this option is given. It can be shown and hidden later using the menu Options > Titlebar.

--shortMainTitle The operating systems title bar (Maintitle in Minimal Example on page 30) of the main window shows

- the application title (string after DESCRIPTION)
- the path of INTENS
- the hostname
- the INTENS version

With this option, however, it only shows the application title.

--toolbar obsolete

--undo enables undo/redo in INTENS.
The Edit menu is added with two entries:

Undo Ctrl-Z Use this menu or the keyboard shortcut Ctrl-Z to undo a changed value.

Redo Ctrl-Y Use this menu or the keyboard shortcut Ctrl-Y to undo an undo.

You can undo at most five changes.

--helpmsg enables help messages, shown in the Statusbar. The help message mode can be changed later using the menu Options > Help messages.
INTENS can show a help message of the content under the mouse-pointer. The help message is defined using **HELPTEXT**. INTENS knows three types of help messages:

Statusbar shows the help message in the Statusbar (see Minimal Example on page 30).

Tooltip shows the help message as a tooltip.

Disabled does not show help messages.

--resfile <argument> INTENS reads resources (Colors, fonts, plot line styles, etc.) from the file provided with this option.

--createRes <argument> asks INTENS to write a default resource file. <argument> is the filename.
No description file is needed.

--qtGuiStyle <argument> sets the Qt Style.
Use --listQtGuiStyles to get a list of available styles.

--listQtGuiStyles prints the available Qt Styles.
No description file is needed.

--localeDomain <argument> INTENS uses gettext for translation. There are two locale domains:

INTENS itself is translated using the locale domain name "intens" and the locale directory "<INTENS_HOME>/share/locale". The mo file for the german translation is "<INTENS_HOME>/share/locale/de/LC_MESSAGES/intens.mo".

The Application is translated using a second locale domain. The default locale domain name for the application is the name of the description file, without the extension. You can, however, provide the locale domain name using this commandline option. This is needed i.E. when you want multiple applications to use the same translation files.
The locale directory is "<APPHOME>/share/locale". The mo file for the german translation is "<APPHOME>/share/locale/de/LC_MESSAGES/<localeDomainName>.mo".

--maxoptions <argument> sets the maximal number of options shown in a **COMBOBOX**. The default value is 25.

--maxlines <argument> sets the maximal number of lines of the **LOG_WINDOW** and the **STD_WINDOW**. The default value is 500. When the content exeed this maximum, the oldest 20% lines are deleted.

--toolTipDuration <argument> sets the maximal number of seconds a help message is shown. The default value is 10.

--leftIcon <argument> defines the name of the left bitmap in the titlebar. The default value is “semafor” and shows the Semafor logo. See Minimal Example on page 30.

--rightIcon <argument> defines the name of the right bitmap in the titlebar. The default value is “” and shows no right bitmap in the titlebar. See Minimal Example on page 30.

--startupImage <argument> defines the name of the splash screen bitmap. When this option is given and the bitmap is found, it is shown as a splash screen during application startup.

--rolefile <argument> TODO

--includePath <argument> defines the path where INCLUDE files are searched for. Multiple paths are separated by ':'.
An **INTENS** description file can be split into multiple files. These files are included using
INCLUDE file.inc (see **MESSAGE_QUEUE PUBLISH - SUBSCRIBE** example with **TIMER** on page 184).
The INCLUDE statement can include the path, a subpath or no path at all.
INCLUDE files are (also) searched in the current directory:

- without this option
- when <argument> is an empty string
- when it is included in <argument> (as '.' or ':': foo:, :foo, foo:., .:foo, foo::bar, foo:.:bar)

Examples (- -includePath include:/tmp/include):

INCLUDE common.inc (filename only):

- ./include/common.inc
- /tmp/include/common.inc

INCLUDE etc/common.inc (filename with subpath):

- ./include/etc/common.inc
- /tmp/include/etc/common.inc

--pspreviewer <argument> TODO

--disableSVGSupport TODO

--printerConfig <argument> TODO

--jsb <argument> TODO

--xml <argument> TODO

--xmlPath <argument> TODO

--apprunPath <argument> TODO

--dbdriver <argument> TODO

--dbautologon TODO

--debug <argument> TODO

--persistfile <filename> INTENS parses the description file, writes information about persistent components into <filename> (using XML) and exits.

--persistfileREST <filename> Same as --persistfile, but **LABEL** and **HELPTEXT** are not written to <filename>. This allows the usage of html tags in **LABEL** and **HELPTEXT**.

--logconfig <argument> is obsolete. It used to be the configuration filename for log4cxx.

--log4cplusPropertiesFile <argument> sets the filename of the log4cplus properties file. The default is APPHOME/config/log4cplus.properties.

--startToken <argument> TODO

--reflistfile <argument> TODO

--helpdir <argument> TODO

--version prints the INTENS version and exits.
No description file is needed.

--help prints the list of INTENS commandline options and exists.
No description file is needed.

--withTargetStreamInfo <argument> When INTENS detects a dependency violation between input and output values, if automatically clears the output (or target) values (**AUTOCLEAR_DEPENDENCIES**) or it shows a confirm dialog:

The results are not consistent anymore.
Do you want to delete them?
Yes / No

With this commandline option, the dialog also shows the name(s) of the target stream(s) (output values) involved.

--whichGui prints the GUI type (QT) and exists.
No description file is needed.

--withWheelEvent enables the mouse wheel event for:

Number increase/decrease a number using the mouse wheel (unless --withoutArrowKeys is given)
The mouse wheel event can be enabled for single numeric data items using the datapool option **WHEEL_EVENT**(see data_item_more_option on page 41).

COMBOBOX select different entries using the mouse wheel

INDEX increase/decrease a GUI **INDEX** using the mouse wheel

--withInputStructFunc enables the call of a struct (or parent) function when an input variable is changed that does not have a function.
See example Example 3 on page 239.

--withoutArrowKeys disables the “arrow keys”.
When enabled (default), you can use the up/down keys to increment/decrement the digit of a number before the cursor.

--withKeypadDecimalPoint <argument> TODO

--defaultScaleFactor1 sets the default scale factor to 1.0. See section Scale Factors on page 20.

--withoutEditableComboBox disables the completer of **COMBOBOX**es.
The completer makes it possible to type some characters into the **COMBOBOX** and the options are filtered to the values that start with the typed characters.

--withoutTextPopupMenu disables the right click menu of multiline text input fields.

--withoutRangeCheck <argument> TODO

--test <argument> enables TestMode:

- The given function is called after the INIT function. End the function with the **EXIT** statement to close the application automatically.
- The QUIT function is not run when the application is closed.
- **MESSAGEBOX**es are not shown. Their content is printed to stderr.

--testmode enables TestMode.

--noInitFunc Don't run the INIT function.

--replyPort <argument> An INTENS application can run as a desktop program or in the browser (webtens). When run in the browser, INTENS must be started with this commandline option. INTENS will listen on this tcp port for zmq requests. Webtens sends all user inputs to INTENS as zmq requests, on this port.

This can also be used for testing an INTENS application: a test software, i.E. python behave, can set or get values, call functions, etc. using the same zmq request / reply pattern.

--sendMessageQueueWithMetadata TODO

--defaultMessageQueueDependencies <argument> A **MESSAGE_QUEUE REQUEST** may define dependencies between input (**REQUEST**) and output (**RESPONSE**) **STREAM**s (see paragraph Dependency on page 52).
Each **STREAM** can have an explicit option **DEPENDENCIES** or **NO_DEPENDENCIES** (see job_message_queue_option on page 214 or job_plugin_option on page 214).
Without such an option, the default **MESSAGE_QUEUE** dependencies are used.
This commandline option can change that default:

- 1 or "true" (=default): the **STREAM**s are used with dependencies by default
- 0 or "false": the **STREAM**s are not used with dependencies by default

Note: the reason for this commandline option was the introduction of **MESSAGE_QUEUE REQUEST** dependencies. This option allows existing applications to behave as before, without changing the application code:
`--defaultMessageQueueDependencies=0`.

--oauth <argument> TODO

--oauthAccessTokenUrl <argument> TODO

--opentelemetryMetadata TODO

--lspWorker TODO

--unitManager[=<argument>] Activate the unit manager (see section Unit Manager on page 242).
The optional argument causes the units that are known by the unit manager to be shown as comboboxes:

- comboBox_hide_single: (known) units with only one choice are not shown as comboboxes
- comboBox_always: all (known) units are shown as comboboxes

Index

Symbols

A

B

C

D

E

F

S