YAC Data Builder

version 4.13

User Guide

Table of Contents

Chapter

1 Introduction

The YAC Data Builder program is used to prepare survey data for analysis under the YAC Data Analyzer application.

1.1 Requirements

YAC Data Builder works only on PC computers running the Microsoft Windows operating systems (Windows 9x, Windows NT, Windows 2000, Windows XP and newer versions).

1.2 Installation

The YAC Data Builder program needs no special installation - it's enough to copy the program and associated files to any folder on the computer. However, please note that to process protected surveys, YDB needs access to the license database on YAC Software's server.

1.3 Protection

YAC Data Builder is protected against illegal use - the full version of the application has to have access to the license database on YAC Software's server. Access details are provided after purchase.

Note
The Lite version is not protected.

1.4 Lite Version

YAC Data Builder - Lite is the freeware version of the application.

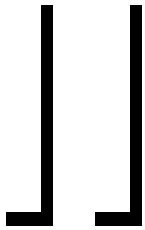
This version is limited to data files of at most 1100 records / cases and 100 columns / variables (both conditions must be met). Protected surveys cannot be processed as well as surveys with specialized radio and press analyses.

1.5 Contact and Support

In case of questions concerning YAC Data Builder or procedures described in this document, please contact:

YAC Software
support@yac.com.pl
www.yac.com.pl

Chapter



2 Functionality

YAC Data Builder converts input files into a format used by YAC Data Analyzer (output files).

This processing is required to:
- optimize data size and access time to the data,
- encrypt descriptions and data so that the original data cannot be easily accessed.

Moreover, during processing, YAC Data Builder checks the correctness and consistency of the descriptions and data itself.

2.1 Input Files

To process survey data, the following input files must be present:
- a script file describing the data with the `.dbs` extension,
- data files (split into waves) and column description files.

Moreover, the following optional files may be present:
- survey / company icon file,
- survey / company logo file,
- information files.

2.1.1 Survey Description File

A survey that will be processed by YAC Data Builder must be described in a script file with the `.dbs` extension.

This file must include only ASCII characters with no extra formatting codes. This means that if you are editing this file in Word, for instance, the file must be saved as a text file.

In this file, wave, weight, question, response, and other definitions will be documented. Data and column localization files are also described here (instruction `recordset`).

The full description of the language can be found in the chapter Survey Definition Language.

2.1.2 Data Files and Column Locations

Data files should be saved in the Fixed-ASCII format, meaning that data for consecutive cases is written in consecutive lines, and data for a variable is written always in the same columns (in each line). Moreover, each record is of the same length.

In one data file, a single survey wave may be saved.

This format does not describe what data is written in the consecutive columns. Thus, additional files are needed that describe which codes go into which columns. So, each data file needs a structure description file. If two consecutive data files have the same structure, it is enough to have a column locations file for the first data file.

Column locations can be documented in one of the two formats handled by YAC Data Builder:

Files in SPSS format, e.g.:

```
DATA LIST FILE='WAVE1.DAT'/
V1 1
V2 2-4
V3 5-24 (A)
V4 25
...
```

where `Vk` denotes the names of variables in the file. Variable formats, labels, value labels, missing values, etc. are ignored by YAC Data Builder.

Files in simple format, e.g.:

```
V1 L1
V2 L3
V3 L20
V4 L1
...
```

where `Vk` denotes the names of variables and `Ln` describes the number of columns used by a variable.

All data files and column location files should be placed in the same folder as the main script (`.dbs`) file.

The import utility in YAC Data Builder will automatically convert SPSS data files (`.sav`) into data and column locations files.

See also: instruction `recordset`.

2.1.3 Graphic Files

You can add additional graphic files to distributed data files. The survey's icon will be displayed in the YAC Data Analyzer application after a survey is opened. This icon will also be visible in Windows Explorer.

The file should have the `.ico` extension and has to be saved in the Windows icon format.

See also: instruction `icon`.

You can also distribute an image file (such as the company's logo). This image will be displayed in the YAC Data Analyzer's status bar or survey manager window after the survey is opened.

The file can be in any of the following formats: Windows bitmap, GIF, or JPEG.

See also: instructions `logo` and `logoPlacement`.

2.1.4 Information Files

You can also optionally distribute with the data additional information files in any, Windows recognizable, format. Thus, you can include Word, Excel, PowerPoint files, PDF files, or HTML files.

You can view these files in Survey Manager in YAC Data Analyzer.

See also: instruction `infopage`.

2.2 Licenses

Surveys may be protected against illegal distribution using YAC Software's licensing system.

2.2.1 Free and Protected Surveys

In YAC Data Builder, survey files may be processed in two ways:
- as free (non-protected) surveys,
- as protected surveys.

A free survey is a data file not protected against illegal distribution. Anybody who has access to the survey's `.das` file will be able to create analyses based on this data (YAC Data Analyzer is freely distributed).

However, surveys can be protected against illegal use - to open these surveys on a computer, this computer has to be registered in the surveys' license database (available as a separate purchase). It will be possible to open this survey on registered computers only.

The YAC Code Generator application is used to register computers in the database.

If new computers are added to the database, the data has to be processed by YAC Data Builder again (so that the survey will be accessible on those new computers).

See also: instruction `licenses`.

Note
Survey protection is not available in the Lite version.

2.3 Output Files

YAC Data Builder creates a `.das` file that contains:
- a converted `.dbs` file,
- converted data files,
- converted column locations files.

2.3.1 Errors, Warnings, Hints

If in documentation files, data files or column locations files errors are found, then:
- the application will report those errors to the user,

- script syntax and similar errors will be reported as comments in the `.dbs` file,
- the `.das` file will not be created / will be deleted.

The second point should be clarified a bit:
- The `.dbs` file is a text file that treats text after two slashes as comments - this text will be ignored by YAC Data Builder.
- If YAC Data Builder finds errors in this file (for instance, no closing **end** to a definition **def**), this information will be inserted into the `.dbs` file after the following characters: `//!` (so this will be a special type of comment). These comments can then be easily navigated (through the `F7` / `F8` keys or through standard search).
- When YAC Data Builder checks this file again, the first thing it does, it removes all special comments (so all previous errors, warnings and hints are removed so as not to come into conflict with any new errors). This also means that you should not create such special comments by hand - they will be automatically removed.

2.3.2 Distribution

The output `.das` file that is created by YAC Data Builder is ready for distribution - you can open this file in YAC Data Analyzer. It contains all data files and their structure, the converted script (`.dbs`) file, and (in case of protected surveys) licenses assigned to this survey.

The `.das` file does not include the survey's optional icon (it has to be visible to the operating system) or information files (these also have to be available to Windows as separate files).

Summing up, when sending the data to a client, the following files have to be included in the distribution:
- the `.das` file,
- the optional icon file,
- optional information files.

These files don't have to be installed in any special way - it's enough to copy these files to the user's computer.

2.4 Console Version

There are two versions of the application: `YDB.exe` (a standard GUI Windows application) and `YDBC.exe` - a command line / console version. The second program can be used for automatic processing of survey files.

`YDBC.exe` is best run in a console window (or MS-DOS mode).

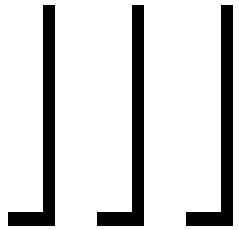
The program takes the following command line parameters:
- optional parameter `-d` that tells the application not to process data files but to check the syntax of the `.dbs` file only (useful when data files are large and their repeated processing takes too much time); as in the GUI version, any errors will be saved to the `.dbs` file as comments starting with `//!`
- mandatory parameter - a `.dbs` file name (supplied with the extension).

Examples:

Check documentation and process data files of `Survey1.dbs`:
```
YDBC.exe "C:\My Surveys\Survey1\Survey1.dbs"
```

Check the documentation file of `Survey2.dbs` only:
```
YDBC.exe -d Survey2.dbs
```

Chapter



3 Keyboard Shortcuts

Most functions in the program can be accomplished (and accelerated) through using keyboard shortcuts:

File navigation

move one line up / down	up / down arrow
move one character left / right	left / right arrow
scroll one line up / down	`Ctrl` + up / down arrow
move one word left / right	`Ctrl` + left / right arrow
move one page up / down	`PgUp` / `PgDn`
move to the start / end of the page	`Ctrl + PgUp` / `PgDn`
move to the start / end of the line	`Home` / `End`
move to the start / end of the file	`Ctrl + Home` / `End`

Note: if at the same time as you press the above keys, you hold down the `Shift` key, the block of text between the two cursor positions will be selected; this block can next be deleted, copied or moved.

Basic editing

change between insert and overwrite modes	`Ins`
copy to the clipboard	`Ctrl + C` or `Ctrl + Ins`
cut to the clipboard	`Ctrl + X` or `Shift + Del`
insert from the clipboard	`Ctrl + V` or `Shift + Ins`
delete a block	`Ctrl + Del`
delete the character after the insertion point	`Del`
delete the character before the insertion point	`Backspace`
delete the word after the insertion point	`Ctrl + T`
delete the word before the insertion point	`Ctrl + Backspace`
undo the last operation	`Ctrl + Z` or `Alt + Backspace`
redo undone operations	`Shift + Ctrl + Z` or `Shift + Alt + Backspace`
select the whole file as a block	`Ctrl + A`
indent a block	`Shift + Ctrl + I`
unindent a block	`Shift + Ctrl + U`

Basic editing, cont.

delete line	`Ctrl + Y`
delete to the end of line	`Shift + Ctrl + Y`
standard blocks	`Shift + Ctrl + N`
column blocks	`Shift + Ctrl + C`
line blocks	`Shift + Ctrl + L`
move to the previous window	`Shift + Ctrl + Tab`
move to the next window	`Ctrl + Tab`
show the list of open windows	`Alt + 0` (zero)
close window	`Ctrl + F4`
quit the application	`Alt + F4`

Search and replace functions

find	`Ctrl + F`
find and replace	`Ctrl + H`
repeat the last find / find and replace operation	`F3`
incremental search	`Ctrl + F3`
go to a line	`Ctrl + L`
insert bookmark n (where n is a digit)	`Shift + Ctrl + n` (e.g. `Shift + Ctrl + 2`)
go to bookmark n	`Ctrl + n` (e.g. `Ctrl + 5`)

Macro and processing operations

start / end macro recording	`Shift + Ctrl + R`
run macro	`Shift + Ctrl + P`
check script	`Ctrl + F9`
process data	`F9`
run YAC Data Analyzer with processed data	`F10`
move to the previous error	`F7`
move to the next error	`F8`

File operations

new file	`Ctrl + N`
import file	`Ctrl + I`
open file	`Ctrl + O` (the letter o)
save file	`Ctrl + S`

Chapter

IV

4 Survey Definition Language

In this topic, the survey definition language used in `.dbs` files is discussed in full detail.

4.1 Notation

In this chapter, the following fonts are used for different types of information:
- `contents and names of files, as well as language examples use Courier New`,
- **`reserved words are in bold`**,
- *`comments in documentation examples are italicized`*,
- *`text literals are also italicized`*.

4.2 Grammar

In descriptions of various language constructs we will use the following notation:
- text that should be written literally (without any additional characters, spaces, etc.) is written in double quotes (the quotes should not be used in a documentation file),
- optional elements are placed between square brackets (brackets should not be used in a file),
- elements that can be repeated (but may be skipped altogether), are placed between curly brackets,
- all other elements are described using phrases (as a single character sequence, such as `file_name`); these elements are described in more detail in the following paragraphs.

Examples:

The following instruction definition
```
  "file" "=" file_name ";"
```
where `file_name` is a text literal, describes the following instruction:
```
  file = "c:\Doc.txt";
```

And
```
  "file" "=" file_name [ "," file_name ] ";"
```
describes the following instructions:
```
  file = "c:\Doc.txt";
  file = "c:\Doc.txt", "c:\Info.txt";
```
but not:
```
  file = "c:\Doc.txt", "c:\Info.txt", "c:\News.txt";
```

The last example is described by the following definition:
```
  "file" "=" file_name { "," file_name } ";"
```

4.3 General Rules

Before we start to describe specific instructions of a survey's definition file, let's first look at what types of elements might appear in that file.

4.3.1 Formatting

Survey documentation should be saved as a standard text file with no additional formatting characters. So, if you are writing this file in MS Word, for instance, remember to save it as a text file.

In a file, spaces and line breaks may appear anywhere, provided they don't split a documentation element into two parts (such as numbers or identifiers).

4.3.2 Comments

Comments in the documentation file are ignored by YAC Data Builder - they can be used to describe what was done and why.

C or Java type comments are used in survey documentation files, that is:
- comments that end with a line break are denoted by two slash characters `//`,
- multi-line comments start with `/*` and end with `*/`; comments of this type cannot be nested (comments inside comments).

Example:

```
def question
  id = sex;
  name = "Sex";
  // Use the column SEX not M1 -
  // missing data was filled in based on sample data.
  column = SEX;
  def responseList
    def response id = male;   code = 1; name = "men"; end;
    def response id = female; code = 2; name = "women"; end;
  end;
end;
```

In the example, a standard question about the respondent's gender is defined. The two commented lines will be ignored by the application when processing this file.

4.3.3 Identifiers

Identifiers are used to differentiate between various elements defined in a survey file, such as: waves, weights, record sets, questions, responses, etc.

Identifiers are defined by the file's author and should resemble as closely as possible the meaning of a definition. So, for instance, when defining a question's identifier, it's generally better to use identifiers such as `SPO` (for spontaneous awareness) instead of `P1` (for the question's number in the questionnaire).

Regardless of the author's usage of less or more meaningful identifiers, all identifiers must follow these rules:
- identifiers must start with a letter,
- consecutive characters can be from the set: letters, digits, underscore, period,
- non-ASCII characters are not allowed (for instance, language specific diacritic marks).

Thus, the following identifiers are valid: `age`, `user.heavy`, `trade_press` and `wave1`. And these identifiers are not valid:
- `płeć` (don't use diacritic marks),
- `user-heavy` (a dash is not allowed),
- `trade press` (an identifier must be a single sequence of characters excluding spaces and line breaks),
- `2wave` (the first character must be a letter).

Identifiers are not case sensitive. If we define the identifier `sex`, it can be later referred to using `Sex`. However, for better readability, it's better to keep identifier spelling consistent.

4.3.4 Texts

Most objects defined in a survey file are later visible in YAC Data Analyzer. So, a textual description of these elements should also be provided.

Two types of texts can appear in a survey definition file: single-line (e.g. response text) or multi-line (e.g. information pages).

Single-line texts should be places in double quotes. If you want to include a double quote in the text, repeat it:
- *`"A valid text without double quotes"`*,
- *`"An invalid text with " a double quote"`* (a quote in the text must be repeated),
- *`"A valid text with "" double quotes"`* (this will be displayed as: *`A valid text with " double quotes`*).

Multi-line texts should be placed between two less than characters << and two greater than characters >>.

Note
Multi-line strings are allowed only in certain places - in the following chapters these definitions will be marked as such. On the other hand, in all places where multi-line texts can be used, you can use single-line strings.

Note
Strings, unlike identifiers, are case sensitive.

4.3.5 Logic Values

In some places of the documentation file a logic value must be supplied. Enter 0 (zero) for false and 1 (one) for true.

Example:

```
def survey
  demo = 1;
end;
```

In the above example a demonstration version of the survey is defined.

4.3.6 Numbers

Some definitions require numbers - for instance, response codes.

Real (floating point) numbers should be entered with a dot as the decimal separator - this is independent of the computer's regional settings.

If numbers are placed inside texts, usually the decimal separator appropriate for the given language is used (see definition of available languages in the `environ` definition). For instance, weights in Polish texts would be entered with a comma, but in English text - with a period.

4.3.7 Definitions

Survey documentation consists of multiple definitions (such as modules, questions, responses, waves, weights, record sets, etc.).

There are two types of definitions:
- simple,
- complex.

4.3.7.1 Simple Definitions

Simple definitions are used to define single elements or lists of similar elements.

Syntax:

```
element_name "=" value { "," value } ";"
```

Examples:

```
demo = 1;
name = "Demonstration Survey";
wizards = general, press;
```

Note
We showed here the general syntax of simple definitions. However, most definitions allow only for single values and not lists of values (such as `demo` above).

4.3.7.2 Complex Definitions

Complex definitions are used to introduce more complicated elements into the documentation:

```
"def" element_name
  element_definition
"end" ";"
```

Where `element_definition` may consist of many simple and complex definitions.

Example:

```
def question
  id = sex;
  name = "Sex";
  column = v1;
  def responseList
    def response
      id = male;
      code = 1;
      name = "men";
    end;
    def response
      id = female;
      code = 2;
      name = "women";
    end;
  end;
end;
```

In the example above, indents are used to aid readability. Usually, a **def** "sees" its **end** (that is, these two words are written in the same column and all text between is indented). However, the above definition can be written as follows:

```
def question
  id = sex;
  name = "Sex";
  column = v1;
  def responseList
    def response id = male;   code = 1; name = "men"; end;
    def response id = female; code = 2; name = "women"; end;
  end;
end;
```

The type of notation (more or less expanded) depends only on the writer's preference - for the program, both of these definitions are equivalent.

4.4 Common Definitions

In many definitions several standard fields can be defined. These fields are described below.

- `id` (element's identifier - usually a mandatory definition)

  ```
  "id" "=" identifier ";"
  ```

  Identifiers are used by YAC Data Analyzer when saving or opening files (such as reports or parameter definitions).

- `name` (element's name - usually a mandatory definition)

  ```
  "name" "=" single_line_text ";"
  ```

  The name of the element displayed in YAC Data Analyzer (in dialog windows, analyses, etc.).

- `nick` (element's short name - optional definition)

  ```
  "nick" "=" single_line_text ";"
  ```

  When an element is displayed in tables and on charts, its full name can be too long to fit nicely into the analysis. In such cases YAC Data Analyzer will use the element's nickname. If the nickname is not defined, the element's name will be used.

- `info` (additional information about an element)

  ```
  "info" "=" multi_line_text ";"
  ```

  Additional information about an element displayed by YAC Data Analyzer on demand (this can be, for instance, the full text of a question).

In the following chapters, if an element supports the above mentioned definitions, we will just write that standard definitions can be used in the element's definition.

As an example, let's take the definition of a question:

```
id = sex;
name = "Sex";
info = "This question was not asked, data was entered from the sample file";
```

or

```
id = spo;
name = "Spontaneous Brand Awareness";
nick = "SPO";
info = <<
  Respondent was asked to mention all known to him/her brands.
  The first three responses were written down in the questionnaire.
>>;
```

4.5 Environ (environment)

This mandatory section is used to define some basic data about the survey.

For now, you have to define the list of languages in which the survey's texts will be displayed (such as questions, responses, etc.)

```
"languages" "=" language_id { "," language_id } ";"
```

Example 1:

```
def environ
  languages = enu;
end;
```

Example 2:

```
def environ
  languages = plk, enu;
end;
```

The first definition means that the survey is available in English only (USA version). The second defines a survey that supports two languages: Polish and English.

Language identifiers are listed in annex A.

The order of the identifiers is not important.

4.5.1 Defining Texts in Multilingual Surveys

When multiple languages are defined for a survey, all texts of the survey can be defined in such as way as to display a different string for each of the languages.

If we defined an element's name as:

```
name = "Demonstration Survey";
```

then it will be displayed as "Demonstration Survey" in all languages.

Let's now assume that for each language we would like to display this text in that language. Language tags are used for this:

```
name = "<enu>Demonstration Survey<plk>Badanie demonstracyjne";
```

Starting with `<enu>` to the next language tag, the English version is being defined. Starting with `<plk>` - the Polish version is being defined.

Please note that in language tags we use the same identifiers as those listed in the `languages` definition in the `environ` section.

Language texts can be used multiple times in a single text:

```
name = "<enu>Age<plk>Wiek<enu> 15-24<plk> 15-24";
```

The above example is a bit contrived, but shows one aspect of such definitions: sometimes we would like to include the same text in all language versions. This can be accomplished as follows:

```
name = "<enu>Age<plk>Wiek<*> 15-24";
```

So, starting with `<*>` to the next language tag, the text will be displayed in all language versions.

If we don't specify a language at the beginning of the text:

```
name = "Resp: <enu>sex<plk>płeć";
```

the program will treat this as starting with the `<*>` tag (thus, in the example above, the text "Resp: " will be displayed in all language versions).

4.6 Survey

This section describes the survey itself:

- `demo` (demonstration version) [optional]

  ```
  "demo" "=" logic_value ";"
  ```

  For demonstration versions of a survey, YAC Data Analyzer will display a warning message that data is not representative and no statistical reasoning should be based on it.

- `excludeSysMis` (exclude system missing data from calculations) [optional]

  ```
  "excludeSysMis" "=" logic_value ";"
  ```

  Excludes system missing data from calculations (and bases). For instance, having the responses: yes (3 times), no (once), missing data (once), percents with included missing data and excluded missing data will be respectively: 60%, 20% and 75%, 25% (in the first case percents don't sum up to 100 because missing data is included in the base of calculations).

Note 1
If the instruction is not included in the documentation, missing data will be included in calculations.

Note 2
In source data files, a system missing value is any text that is not convertable into a number (so, for instance, a string of blanks).

- `expires` [optional]

  ```
  "expires" "=" year [ "/" month [ "/" day ] ] ";"
  ```

  Defines the date up to which the survey will be accessible.

  Note
  `month` defaults to 1 (January); `day` also defaults to 1. Thus, if only `year` is specified, for instance 2010, this means January 1st, 2010.

- `fixedRecordsets` [optional]

  ```
  "fixedRecordsets" "=" logic_value ";"
  ```

  When set, YAC Data Analyzer will not allow the user to change the record set in an analysis.

- `hideRowsCols` [optional]

  ```
  "hideRowsCols" "=" "off" | "base" | "values" ";"
  ```

  Sets the default handling of empty rows and columns in YAC Data Analyzer:
  - `off` - empty rows and columns will not be hidden,
  - `base` - rows and columns with zero bases will be hidden,
  - `values` - rows and columns with zero values will be hidden.

- `icon` [optional]

  ```
  "icon" "=" file_name ";"
  ```

  This icon, if defined, will be displayed in YAC Data Analyzer in the Survey Manager. `file_name` should be a single-line text.

- `id` (survey identifier)

  ```
  "id" "=" identifier ";"
  ```

  This definition identifies a single survey. No two surveys (of the same company) should have equal identifiers.

  YAC Data Analyzer uses these two definitions when saving reports and parameter definitions.

- `logo` (the company's or survey's logo) [optional]

  ```
  "logo" "=" file_name ";"
  ```

  The specified image (it has to be either a Windows bitmap or a file in one of the formats: JPEG or GIF) will be displayed in YAC Data Analyzer in the status bar or in the survey manager window. This can be the company's logo, the survey's logo, or any other image that you want to show to the user.

  The status bar is currently 30 pixels high - images that are higher will be rescaled to fit the status bar.

  There are no limits for the image dimensions when it is placed in the survey manager window.

  The bottom left pixel (or less often, the top right - depends on the image format) defines the transparent color - this color will not be displayed (the background color will be visible - the color of the status bar or the survey manager).

- `logoPlacement` (placement of the company's or survey's logo) [optional]

  ```
  "logoPlacement" "=" "statusBar" | "surveyManager" ";"
  ```

  The image defined in the `logo` instruction will be placed respectively: in the application's status bar or in the survey manager window.

- `name`

  ```
  "name" "=" single_line_text ";"
  ```

  The name of the survey.

- `namespace`

  ```
  "namespace" "=" identifier ";"
  ```

  This definition is used to specify the company that is distributing this survey. This identifier should be the same for all surveys of a given research institute. Please contact YAC Software to determine a unique identifier for your company.

- `owner` [optional]

  ```
  "owner" "=" single_line_text ";"
  ```

  This optional text will be displayed in YAC Data Analyzer in the Survey Manager.

- `stdStats` (standard statistics) [optional]
  `advStats` (advanced statistics) [optional]

  ```
  "stdStats" "=" identifier { "," identifier } ";"
  "advStats" "=" identifier { "," identifier } ";"
  ```

  These fields define standard and advanced statistics - both groups will be displayed in YAC Data Analyzer in separate tabs of the statistics parameter dialog window.

  Valid identifiers for single statistics:
  - `cntA` - Count (actual) - the actual number of respondents in a cell.
  - `cntW` - Count (weighted) - the weighted number of respondents in a cell.
  - `pctLayer` - Layer percent - number of respondents in a cell divided by the number of respondents in the layer.
  - `pctRow` - Row percent - number of respondents in a cell divided by the number of respondents in the row.
  - `pctCol` - Column percent - number of respondents in a cell divided by the number of respondents in the column.
  - `index` - Affinity index - the number of respondents in a cell is divided by the expected number of respondents in the cell (multiplied by 100).
  - `est` - Population estimation - the result estimated to the surveyed population.
  - `meanRow` - Row mean - means in cells in a row, grouped by questions.
  - `meanCol` - Column mean - means in cells in a column, grouped by questions.
  - `varRow` - Row variance - variance in cells in a row, grouped by questions.
  - `varCol` - Column variance - variance in cells in a column, grouped by questions.
  - `stdDevRow` - Row standard deviation - standard deviation in cells in a row, grouped by questions.
  - `stdDevCol` - Column standard deviation - standard deviation in cells in a column, grouped by questions.
  - `sumRow` - Row sum - sum of cells in a row, grouped by questions.
  - `sumCol` - Column sum - sum of cells in a column, grouped by questions.
  - `shareRow` - Row share - share in the row of the sum of cells in the cell's column, grouped by questions.
  - `shareCol` - Column share - share in the column of the sum of cells in the cell's row, grouped by questions.
  - `indLayer` - Layer indicator - values of media indicators in layers.
  - `indRow` - Row indicator - values of media indicators in rows.
  - `indCol` - Column indicator - values of media indicators in columns.

You can also use the following identifiers for groups of statistics:

- `cnt` - counts.
- `pct` - percents.
- `mean` - means.
- `var` - variances.
- `stdDev` - standard deviations.
- `sum` - sums.
- `share` - shares.
- `ind` - media indicators.

Note
If these field are not defined, all statistics will be shown and all will be treated as standard.

Note
If only the standard statistics are defined, only these statistics will be available in YAC Data Analyzer; if only the advanced statistics are defined, all other statistics will be treated as standard.

- `wizards`

```
"wizards" "=" identifier { "," identifier } ";"
```

This field defines which analysis creation wizards will be available for the survey. Currently, the following wizards can be specified (their identifiers are listed in the first column):

General analyses:

`table1d`	frequency tables,
`table2d`	cross-tabs,
`cmpWaves`	trends (compare results between waves),
`cmpGroups2`	compare two target groups,
`cmpGroups`	compare target groups (two or more),
`genComplex`	create an analysis based on complex questions (multi-dimensional),
`genCrossBann`	cross-tabs that include percentages and means.

If you want to provide all of the above wizards, you can use the following identifier: `general`.

Examples:

```
wizards = table1d, table2d, cmpGroups;
wizards = general;
```

Example of a definition with all of the above fields:

```
def survey
  namespace = yac.com.pl;
  id = demo;
  owner = "YAC Software";
  name = "<enu>Demonstration Survey<plk>Badanie demonstracyjne";
  demo = 1;
  icon = "Demo.ico";
  logo = "CompanyImage.jpg";
  logoPlacement = surveyManager;
  wizards = general;
  excludeSysMis = 1;
  hideRowsCols = values;
end;
```

4.6.1 Infopage (Information Pages)

In the `survey` section one more definition is optional - `infopage`. This definition can be used to add information pages to the survey. You can describe here things such as sampling method, sample structure, questionnaire, additional materials, project's history, etc.

This is a complex definition that can include the following fields:

- `name`

  ```
  "name" "=" single_line_text ";"
  ```

  The page's name is displayed in YAC Data Analyzer in the table of contents in Survey Manager.

- `text`

  ```
  "text" "=" multi_line_text ";"
  ```

  This text will be displayed after a page is selected in the table of contents. It can use HTML tags to format content.

- `file`

  ```
  "file" "=" single_line_text ";"
  ```

  Instead of entering the page's text into the survey file, this text can be read from an external file (specified in the definition). This file can be in any format recognizable by Windows (that is, the file's extension must be registered in the system). This includes: HTML files, text files, files of standard applications (such as Word, Excel, or Acrobat Reader).

- `addr`

  ```
  "addr" "=" single_line_text ";"
  ```

  File's distributed with the survey can be difficult to update in a timely manner. In cases when this data has to be available as quickly as possible, it's possible to define a page pointing to any internet address. When this page is selected in YAC Data Analyzer, the program will display the referenced www page.

  When processing the survey file, YAC Data Builder will check whether the page exists (if you're on-line).

- `openInNewWindow`

  ```
  "openInNewWindow" "=" logic_value ";"
  ```

  By default, an information page is displayed in an internal window of YAC Data Analyzer (in Survey Manager). If you want to open this page in a separate window (the window of the appropriate application), add the above definition with the value of 1.

An example of the whole definition:

```
def survey
  . . .
  def infopage
    name = "<enu>Survey Calendar<plk>Harmonogram badania";
    file = "<enu>Calendar.xls<plk>Kalendarz.xls";
    openInNewWindow = 1;
    def infopage
      name = "<enu>Vacations<plk>Wakacje";
      text = <<
        <enu> During summer vacations <b>no</b> interviews will be
              conducted.
        <plk> W czasie letnich wakacji <b>nie</b> będą przeprowadzane
              żadne wywiady.
      >>;
    end;
  end;
  def infopage
    name = "<enu>Current Wave Status<plk>Stan realizacji obecnej fali";
    addr = "yac.com.pl/<enu>curWaveStatus<plk>stanAktFali<*>.html";
  end;
end;
```

Notes
- Information pages may be nested (sub-pages).
- File names, page texts and addresses can be versioned for different languages.
- Multiple blanks and new lines in multi-line texts are ignored by YAC Data Analyzer. Indenting used in the example above helps readability, but has no impact on how it will be displayed.
- In multi-line texts you can use HTML tags - in the example above, the word "no" on page "Vacations" will be in bold.
- To view the survey's calendar, the MS Excel application will be executed, in which the appropriate file will be opened. All other pages will be displayed in Survey Manager's window.
- For a given page, you can use only one of the definitions: `text`, `file`, `addr`.

4.7 Licenses

This definition turns on survey protection - only registered users will be able to open the data.

Note
This option is not available by default. This extension to YAC Data Builder may be purchased separately from YAC Software.

Licenses are listed in additional complex definitions. In definitions of licenses, standard fields are not defined.

Note
The whole section is optional. If it's not defined, the data will not be protected.

Example:

```
def licenses
  batchCode = "DEMOMA";
  def license
    key = "29E2-4F60-083E-055E-BC23-36C5-1D74-B933-0857";
    type = standAlone;
    expires = 2006/12/20;
  end;
  def license
    key = "4C74-F80F-4E3D-2A37-7F7F-2EE8-21C5-CD58-0936";
    type = network;
    count = 3;
  end;
  def license
```

```
    key = "23728059";
    type = hardwareKey;
    expires = 2007/01/20;
  end;
end;
```

Notes

- The definition above defines three licenses.
- The first license is a stand-alone, single seat license that expires on 2006-12-20.
- The second license is a network license for 3 simultaneous users that never expires.
- The third license is a hardware key based license that expires on 2007-01-20.
- The first two licenses are based on YAC License Kit software; the last license is based on Aladdin's HASP hardware keys.
- If you want to define licenses based on hardware keys, your Batch Code (supplied by Aladdin) must be defined in a separate instruction `batchCode`. This code will be used to extract your Vendor Code from a `.hvc` file; the file with this code must be present in Aladdin's default directory with Vendor Code files or in the same directory as the survey's `.dbs` file. This file must be named `<batchCode>.hvc` (`DEMOMA.hvc` in the example above).

4.8 Recordset

Record sets define source data for the survey.

This definition includes standard fields (`id`, `name`, `nick`, `info`).

The following fields can also be defined:

- `hidden` [optional]

  ```
  "hidden" "=" logic_value ";"
  ```

  When set, YAC Data Analyzer will not allow the user to select this record set as a base for analyses (but questions from multiple recordsets and 1-to-many data structures are still allowed).

- `stats` (statistics) [optional]

  ```
  "stats" "=" identifier { "," identifier } ";"
  ```

  Limits the available statistics (to the specified list) in analyses based on this record set.

  For a list of valid identifiers, refer to the `stdStats / advStats` documentation in the `survey` section.

Note
The `nick` field is used only in surveys that use more than one record set; the shortened version of the name is displayed in tables in the **Calculation base** parameter.

This definition lists the source data files that make up the record set.

4.8.1 Files

This is a list of definitions of the type:

```
"colSpecFormat" "=" format_identifier ";"
"colSpec" "=" column_file_name ";"
"dataFile" "=" data_file_name ";"
```

In the column specification file (`colSpec`) the column names, locations (positions and widths), and types are listed. In the data files (`dataFile`), the data itself is placed. File names should be given in double quotes.

The `colSpecFormat` instruction defines the column specification format (in files defined in `colSpec` instructions). In the `colSpecFormat` instructions one of the following identifiers needs to be used:

- `simple` (simple column specification),
- `sps` (column specification compatible with SPSS Syntax instructions).

Examples of column specification files in the above formats are given in annex B.

The `files` definition can include all of the above definitions multiple times (`colSpecFormat`, `colSpec`, `dataFile`). A definition of column specification format describes this format for all following column specification files (unless a new format is specified). A column specification file defines the columns for all following data files (unless a new column specification file is specified). Thus the `files` definition should start with a `colSpecFormat` instruction, then a `colSpec` instruction.

If the column specification format is not defined, it is set to `simple` by default.

Example:

```
def recordset
  id = respondents;
  name = "<enu>respondents<plk>respondenci";
  nick = "Resp:";
  info = "<enu>Respondents' background data<plk>Metryczka respondenta";
  def files
    colSpecFormat = sps;
    colspec = "fdemo001.var";
    datafile = "fdemo001.dat";
    datafile = "fdemo002.dat";
    colspec = "fdemo003.var";
    datafile = "fdemo003.dat";
    datafile = "fdemo004.dat";
  end;
end;
```

Notes

- Column definitions for the first two data files are the same and are given in the file `fdemo001.var`. Similarly, the two consecutive data files have the same column specification (defined in file `fdemo003.var`).
- All column specification files are in SPSS Syntax format.
- The text in the `info` field will be displayed in the first window of analysis creators.
- Data files need to be in Fixed-ASCII format.

4.8.2 Join

In surveys based on several record sets, the `join` instruction needs to be used to define how these record sets are merged / joined.

```
def recordset
  . . .
  def files
    . . .
  end;
  join = id;
end;
```

This means that the above record set will be joined with the main record set (defined as the first one) on column `id`.

Notes
- The `id` column must exist in both this record set and in the first recordset; it must be defined in all data files.
- The `join` instruction has no sense for the first record set, for all consecutive record sets it is mandatory.

4.9 Weights

This instruction defines weights in the survey.

Availability of calculations on non-weighted data is defined by the instruction:
`"noweight" "=" logic_value ";"`

Descriptions of weights are defined in additional complex definitions. In weight definitions standard fields can be defined. Moreover, the source column for the weight needs to be defined:
`"column" "=" column_name ";"`

Base for population estimations can be defined for weighted and non-weighted data. If it is not defined, estimations will not be available. Use the `population` instruction to define this base.

Note
This whole section is optional. If it's not defined, calculations on non-weighted data will be available by default. No other weights will be defined.

Example:

```
def weights
  noweight = 1;
  def weight
    id = population;
    name = "<enu>population<plk>populacyjna";
    column = v17;
    population = 29957930;
  end;
  def weight
    id = households;
    name = "<enu>households<plk>gosp. domowych";
    column = v18;
    population = 13582900;
  end;
end;
```

Notes
- The above example defines two weights and allows for calculations on non-weighted data. Weight values are taken from columns `v17` and `v18` of the data files (respectively).
- Population estimations will be available for weighted data only.
- The text defined in the `info` instruction (if defined) will be displayed in the first window of analysis creators.

4.10 Waves

In `waves` definition, consecutive waves may be defined as well as the following field:

- `minSelCount` [optional]

  `"minSelCount" "=" integer ";"`

  Defines the minimum number of waves needed for analyses; if the user selected less waves in YAC Data Analyzer, the results will not be calculated.

In `wave` definitions, standard fields may be defined. Moreover, the following fields are valid:

- `dateStart` [mandatory]

  `"dateStart" "=" year [ "/" month [ "/" day ] ]  ";"`

  Defines the starting date of the wave - for information purposes only.

  `year`, `month`, and `day` are numbers from the standard ranges. The year must be defined using 4 digit notation. If end date is not defined, it will be equal to the start date. Month and day are optional (but if you want to specify the day, you need to define the month, too).

  Date format is independent of the current regional settings in Windows. However, YAC Data Analyzer will display these dates using current Windows settings.

- `dateEnd` [optional]

  `"dateEnd" "=" year [ "/" month [ "/" day ] ] ";"`

  Defines the ending date of the wave - for information purposes only.

  Note
  If defined, it must specify a later date than `dateStart`.

Example:

```
def waves
  minSelCount = 2;
  def wave
    name = "Pilot study";
    dateStart = 2002/01;
  end;
  def wave
    name = "pre-test";
    dateStart = 2002/02/01;
    dateEnd = 2002/02/14
  end;
  def wave
    name = "post-test";
    dateStart = 2002/06/01;
    dateEnd = 2002/06/14;
  end;
end;
```

Notes

- If the wave identifier is not defined, it is assumed that waves will have identifiers of the form `waveX`, where `X` is the number of the wave.
- The wave's name is displayed in YAC Data Analyzer in the wave selection dialog, for instance. If it's not specified, it is assumed to be of the form Wave X, where X is the wave's number.
- The number of defined waves must be the same as the number of defined files in each recordset. This is a temporary solution and hopefully will be removed in one of the future versions of the program.

4.11 Data

In this complex definition questions and responses, as well as question modules can be defined:

```
def data
  // here go question and module definitions
end;
```

4.11.1 Module

Modules are used to organize survey data into a folder-like structure. Standard fields may be defined, as well as other modules and questions.

Modules, sub-modules and questions define a hierarchy similar to folders, sub-folders and files on a computer. Modules should be introduced to help the users find data (a given question) in the survey. Thus modules should be defined with that in mind, and should not be over used.

An example definition of modules and sub-modules:

```
def module
  name = "Demographics";
  def module
    name = "Respondent";
    info = "Respondent's Demographic Data";
    // next go questions and / or sub-modules
  end;
  def module
    name = "Household";
    info = "Household's Demographic Data";
    // next go questions and / or sub-modules
  end;
end;
```

4.11.2 Question

Questions are defined in modules. Questions can also be defined right under the `data` definition outside of any module - these questions will then be available in YAC Data Analyzer on the top / root level.

A question definition example:

```
def question
  id = sex;
  name = "Sex";
  info = "Data taken from sample records";
  // here go response definitions
end;
```

Except for standard fields, `response`, `responseList`, and `responseGrid` definitions can be placed in a question's definitions.

Measurement scales
Starting with version 3.03, you can also define measurement scales in questions:

```
scale = <scale_identifier>;
```

where the scale identifier is one of the following: `nominal`, `ordinal`, or `interval`. For instance:

```
def question
  id = educ;
  name = "Education";
  scale = ordinal;
```

```
  // here go response definitions
end;
```

Note 1
If the scale is not defined, interval is assumed.

Note 2
YAC Data Analyzer will not allow for calculating means, variances and standard deviations for questions with scales other than interval.

4.11.3 ResponseList

A list of responses must appear in every question. These can be defined in the following complex instruction:

```
def responseList
  // here go definitions of responses
end;
```

Standard fields are not used here. In the response list, we can have definitions of responses (described later) as well as the following elements:

- `column`

  `"column" "=" identifier ";"`

  that describes the column of the data on which the responses will be based (a definition most often used in single-choice questions). Identifier is the name of a column that must appear in at least one column specification file (`recordset`).

- `columnList`

  `"columnList" "=" identifier { "," identifier } ";"`

  that describes the list of columns on which response codes were written.

- `code`

  `"code" "=" number ";"`

  that describes the code that all responses in the response list share (most often used in dichotomous multi-choice responses). Number can be prefixed by a minus sign for negative values.

- `range`

  `"range" "=" number_list ";"`

  that defines codes that will be counted as a single code.

- `numeric`

  `"numeric" "=" number_list ";"`

  that defines a numeric question; for each code in `number_list`, counts will be calculated independently.

  `number_list` consists of single codes and code ranges, separated by commas. To define a code range, separate the minimum value from the maximum value with a colon.

  Examples:

  ```
  range = 3, 6, 8:9;
  range = 1:2;
  . . .
  numeric = 15:75;
  ```

  The first instruction defines a single response that will sum up responses with the codes 3, 6, and 8 thru 9 (so, for instance, the middle of a scale, no response, and missing data). The second instruction defines, for instance, top two boxes. The third instruction defines a numeric response that will calculate values from 15 to 75 independently.

  In range definitions, the words `min` (no lower limit) and `max` (no upper limit) can also be used:

  ```
  numeric = -15:max;   // responses from -15 an up
  numeric = min:max;   // no limits on response codes (report counts for all found codes)
  ```

- `attr` (response attributes)

  `"attr" "=" identifier_list ";"`

  This instruction defines additional response attributes. The following can be used:

  - `estIndependent`: population estimations will be carried out for each response independently; this attribute is usually used for questions that divide the sample into groups that should be estimated for the whole population (such as the day of week of the interview),

  - `allowOverlaps`: turns off control of response codes; in the example below the program would generate a warning that codes for different responses overlap:

    ```
    def response ... numeric = 0:100; end;
    def response ... name = "not applicable"; code = 98; end;
    ```

    if one of these responses would be defined with the `allowOverlaps` attribute, this warning would not be generated,

  - `noMean`: exclude the code from calculating means; let's assume that we have the following scale: *I definitely agree* to *I definitely disagree* and the response *not applicable*:

    ```
    def response ... name = "I definitely agree"; code = 1; end;
    ...
    def response ... name = "I definitely disagree"; code = 5; end;
    def response ... name = "not applicable"; code = 9; attr = noMean; end;
    ```

    Since `noMean` was added to the attribute list of the last response, means for this question will be calculated for values 1 to 5 only.

    Note 1
    This attribute cannot be used in numeric responses (`numeric`).

    Note 2
    If in a single question two responses include the same code, these responses have to be consistent as to the `noMean` attribute, for instance:

```
def response ... name = "middle three boxes"; range = 2:4; end;
def response ... name = "neither yes nor no"; code = 3; attr = noMean; end;
```

Both responses include the code 3, but their `noMean` definitions are not consistent - should the value 3 be calculated in means, or should it not?

Note 3
This attribute should not be used in questions with a non-interval measurement scale.

- `substStats`: automatically changes results in YDA from percents to means.

4.11.4 Response

In response lists (`responseList`), responses are described in complex definitions, as the example below shows:

```
def response
  id = user;
  name = "User";
  info = "user of brands mentioned in question Q1";
  column = v2;
  code = 1;
end;
```

Standard fields can be defined here, as well as fields already described in response lists: `attr`, `column`, `code`, `range` and `numeric`. Fields `id` and `name` are mandatory. One of the definitions: columns or codes (`code`, `range`, `numeric`) must be present in a response definition.

If the `attr` field, `column` or definition of codes is not used in the response's definition, the appropriate definition is used from the parent response list. Definition of columns and codes must appear in either the response list or the response.

If the `attr` field, `column` or definition of codes is used in both definitions - response list and response - the second one will be used (thus, the response's definition overrides response list's definition). This can be useful when responses from different questions are merged in a single question (see the example of a combined question).

If in the response list there are multiple definitions of `attr`, `code`, `range` and `numeric`, the last one before the definition of the response is taken.

4.11.5 ResponseGrid (Multi-dimensional Question)

The `responseGrid` complex definitions are used to describe multi-dimensional questions. Below we give several examples of this type of questions:

Please indicate if you agree with the following statements:

	I definitely agree	*I agree*	*I disagree*	*I definitely disagree*
I love pizza				
I'm afraid of spiders				
Radio & TV payments should not be obligatory				

Another such question:

Which brands fit the following statements (multiple brands may be indicated for a single statement)?

	brand A	*brand B*	*brand C*
nice foam			
nice color			
strong enough			
good price			

Some more examples are presented in the Multi-dimensional Questions chapter.

So, in general, multi-dimensional questions consist of several axes (statements x scale and statements x brands in the examples above). There can even be questions with 3 axes - take the second example and add a scale describing how strongly the respondent agrees that a brand fits a statement.

Please note that in multi-dimensional questions at most one axis is a response axis (as scales), but this axis might not be present (as in the second example above).

Definitions of multi-dimensional questions were introduced for two reasons:
- easier access to such questions in YAC Data Analyzer (where responses can be presented in tables similar in layout to a questionnaire),
- easier and more concise definition of such questions in YAC Data Builder scripts.

These definitions require that column names follow a certain scheme. Elements of all non-scale axes must be assigned to strings that, together with a mask, make up column names for all columns in the question.

For instance, let's say we have a statements x brands question that is saved on dichotomous columns:

```
P10a1 - brand A: statement 1
P10a2 - brand A: statement 2
P10b1 - brand B: statement 1
P10b2 - brand B: statement 2
```

Then the following assignments of strings to axis elements would be possible:

```
brand A: "a"
brand B: "b"
statement 1: "1"
statement 2: "2"
```

And the column names are generated from the above string and the mask `P10**`. The first asterisk will be replaced by brand identifiers, the second asterisk - by statement identifiers.

If the labels of these columns can also be generated in a similar fashion (so they follow the scheme `<brand>: <statement>`), they will be automatically used for axis names (this will be done by the import utility in YAC Data Builder).

If one of the axes is a response axis, the above scheme must be followed by all other axes (the response axis defines the codes that are found in the columns, and not column names).

A multi-dimensional question starts with the definition:

```
def responseGrid
  . . .
end;
```

that can include the following fields:

- `columnMask`

  `"columnMask" "=" single_line_text ";"`

  defines the mask for column names in the question; it must include as many asterisks (`*`) as there are non-response axes

- `code` and `range` - these definitions are the same as in response lists.

Moreover, axes should be defined here (including the optional response axis).

4.11.6 ResponseAxis

Axes are used to define the dimensions of a multi-dimensional question:

```
def responseAxis
  . . .
end;
```

Standard fields can be defined in an axis definition as well as response and responseList definitions, just like in questions. However:
- only responses of the last axis can include definitions `range`, `code` and `numeric`,
- if `code` was defined on the question level, then none of the responses in any of the axes can define the above fields,
- all axes, except the last (if it defines a response scale), must define the field `columnText`,
- none of the responses can use define `column` or `columnList`.

The `columnText` instruction is used to assign response texts that will be used to build column names:

```
"columnText" "=" single_line_text ";"
```

so in the example of columns `P10a1..P10b2`, for the response "brand A" you would add the definition:

```
columnText = "a";
```

In Question Definition Examples - Multi-dimensional Question some more examples are discussed.

4.11.7 Question Definition Examples

Here we will show some examples for the question types described earlier.

For greater readability, questions and responses were defined in a single language version.

4.11.7.1 Single-choice Question

Responses to this question are based on a single column (`v3`) and its two responses (1 - men and 2 - women).

```
def question
  id = sex;
  name = "Sex";
  def responseList
    column = v3;
    def response id = m; code = 1; name = "men"; end;
    def response id = f; code = 2; name = "women"; end;
  end;
end;
```

It's a single-choice question, since a single value excludes all other values.

4.11.7.2 Multi-choice Question

The responses to the following questions are saved on consecutive columns (`nprinter`, `iprinter`, `lprinter`) as the value 1 (has the device) and all other values (doesn't have the device).

```
def question
  id = peripherals;
  name = "Peripheral Devices";
  def responseList
    // The following code identifies positive responses:
    code = 1;
    def response
      id = pp_nprinter;
      column = nprinter;
      name = "Dot-matrix printer";
    end;
    def response
      id = pp_iprinter;
      column = iprinter;
      name = "Ink-jet printer";
    end;
    def response
      id = pp_lprinter;
      column = lprinter;
      name = "Laser printer";
    end;
  end;
end;
```

This is a multi-choice question since a positive answer to one of the responses doesn't exclude a positive answer to other responses.

Multi-choice questions can also be saved in another way: on consecutive columns codes of chosen responses are saved (so, these are not dichotomous variables; this format is often used when coding open-ended questions):

```
def question
  id = peripherals;
  name = "Peripheral Devices";
  def responseList
    // On these columns, response codes of given answers are saved:
    columnList = per_1, per_2, per_3;
    def response
      id = pp_nprinter;
      // The printer's code can appear on any of the given columns.
      // The same goes for the remaining responses.
      code = 1;
      name = "Dot-matrix printer";
    end;
    def response
      id = pp_iprinter;
      code = 2;
      name = "Ink-jet printer";
    end;
    def response
      id = pp_lprinter;
      code = 3;
      name = "Laser printer";
    end;
  end;
end;
```

4.11.7.3 Combined Question

Combined questions can be used to merge responses from several columns with different codes. In many situations this may be more useful to the user of your data.

```
def question
  id = user;
  name = "User";
  def responseList
    // Most of the data will be taken from column v1:
    column = v1;
    def response
      id = non.user;
      code = 0;
      name = "non-user";
    end;
    def response
      id = user;
      // Note - this data is taken from another column:
      column = v2;
      code = 1;
      name = "user";
    end;
    def response id = light;  code = 1; name = "light";  end;
    def response id = medium; code = 2; name = "medium"; end;
    def response id = heavy;  code = 3; name = "heavy";  end;
  end;
end;
```

In the above example, column `v1` contains the data on usage level (0 - non-user, 1 - light, 2 - medium, 3 - heavy). In column `v2` we just have the user / non-user data. When this data is combined into a single response, it might be easier in analysis.

The above question can also be defined with the use of the `range` instruction:

```
def question
  id = user;
  name = "User";
  def responseList
    column = v1;
    def response id = non.user; code = 0;    name = "non-user"; end;
    def response id = user;     range = 1:3; name = "user";     end;
    def response id = light;    code = 1;    name = "light";    end;
    def response id = medium;   code = 2;    name = "medium";   end;
    def response id = heavy;    code = 3;    name = "heavy";    end;
  end;
end;
```

The above example shows how net-counts can be constructed in any type of question.

And one more example from the General Social Survey, where ranges were used to define age groups (the first question is also available to define exact target groups, for example):

```
def question
  id = agewed;
  name = "Age when first married";
  def responseList
    column = AGEWED;
    def response id = r1; code = 0;       name = "NAP"; attr = noMean; end;
    def response id = rg; numeric = 1:97; name = "age in years"; end;
    def response id = r2; code = 98;      name = "DK"; attr = noMean; end;
    def response id = r3; code = 99;      name = "NA"; attr = noMean; end;
  end;
end;

def question
```

```
  id = agewedg;
  name = "Age when first married (grouped)";
  def responseList
    column = AGEWED;
    def response id = r1;     code = 0;      name = "NAP"; attr = noMean; end;
    def response id = r16m;   range =  1:17; name = "-17"; end;
    def response id = r18_19; range = 18:19; name = "18-19"; end;
    def response id = r20_21; range = 20:21; name = "20-21"; end;
    def response id = r22_25; range = 22:25; name = "22-25"; end;
    def response id = r26_30; range = 26:30; name = "26-30"; end;
    def response id = r31p;   range = 31:97; name = "31-"; end;
    def response id = r2;     code = 98;     name = "DK"; attr = noMean; end;
    def response id = r3;     code = 99;     name = "NA"; attr = noMean; end;
  end;
end;
```

4.11.7.4 Numeric Question

In this question, the `numeric` keyword is used to define a numeric question; an additional response, "refused to answer", was also defined.

```
def question
  id = age;
  name = "Age in years";
  def responseList
    column = v2;
    def response id = age;     numeric = 15:75; name = "age";     end;
    def response id = refused; code = 97;       name = "refused"; end;
  end;
end;
```

4.11.7.5 Multi-dimensional Question

A multi-dimensional question statements x brands (which statements fit which brands; multi-choice version):

```
def question
  id = grid1;
  name = "Statements x Brands";
  def responseGrid
    columnMask = "**";
    code = 1;
    def responseAxis
      id = ax1;
      name = "Statements";
      def response id = r1; columnText = "p1";
        name = "Products of this brand are especially delicious"; end;
      def response id = r2; columnText = "p2";
        name = "Products of this brand are of high quality"; end;
    end;
    def responseAxis
      id = ax2;
      name = "Brands";
      def response id = brandA; columnText = "a"; name = "Brand A"; end;
      def response id = brandB; columnText = "b"; name = "Brand B"; end;
    end;
  end;
end;
```

Columns that this question refers to are the following: `p1a`, `p1b`, `p2a`, `p2b`. Positive responses are denoted by 1's (instruction `code = 1;`). If the instruction `code = 1;` is changed to `range = 1:5;` then positive responses will be denoted by all values from 1 to 5.

Multi-dimensional question statements x scale (does the respondent agree with the given statements):

```
def question
  id = grid2;
  name = "Statements x Scale";
  def responseGrid
    columnMask = "*";
    def responseAxis
      id = ax1;
      name = "Statements";
      def response id = r1; columnText = "p1";
        name = "UPR is the best political party in Poland"; end;
      def response id = r2; columnText = "p2";
        name = "There are too many bureaucrats"; end;
    end;
    def responseAxis
      id = ax2;
      name = "Scale";
      def response id = r1; code = 1; name = "I definitely agree"; end;
      def response id = r2; code = 2; name = "I agree"; end;
      def response id = r3; code = 3; name = "I neither agree nor disagree"; end;
      def response id = r4; code = 4; name = "I disagree"; end;
      def response id = r5; code = 5; name = "I definitely disagree"; end;
    end;
  end;
end;
```

`columnText` is defined for the first axis only, since the second axis describes codes in columns. The question references the following columns: `p1`, `p2`. `columnMask` defines a mask with only one asterisk - there's only one column dimension. In such cases, the `code` instruction before axis definitions is not allowed.

A battery of scales (evaluation on a 11 point scale):

```
def question
  id = grid3;
  name = "Statements x Evaluation";
  def responseGrid
    columnMask = "*";
    def responseAxis
      id = ax1;
      name = "Statements";
      def response id = r1; columnText = "p1"; name = "The product is very dense"; end;
      def response id = r2; columnText = "p2"; name = "The product is very sweet"; end;
    end;
    def responseAxis
      id = ax2;
      name = "Evaluation";
      def response id = r1; numeric = 1:11; name = "scale"; end;
      def response id = r2; range = 98,99; name = "hard to say"; end;
    end;
  end;
end;
```

Opinions on how well various brands fit various statements (for each pair: brand - statement, the respondent can say how well the two go together):

```
def question
  id = grid4;
  name = "Statements x Brands x Scale";
  def responseGrid
```

```
    columnMask = "**";
    def responseAxis
      id = ax1;
      name = "Statements";
      def response id = r1; columnText = "p1";
        name = "Products of this brand are especially delicious"; end;
      def response id = r2; columnText = "p2";
        name = "Products of this brand are of high quality"; end;
    end;
    def responseAxis
      id = ax2;
      name = "Brands";
      def response id = brandA; columnText = "a"; name = "Brand A"; end;
      def response id = brandB; columnText = "b"; name = "Brand B"; end;
    end;
    def responseAxis
      id = ax3;
      name = "Scale";
      def response id = r1; code = 1; name = "fits well"; end;
      def response id = r2; code = 2; name = "fits"; end;
      def response id = r3; code = 3; name = "doesn't fit"; end;
      def response id = r4; code = 4; name = "doesn't fit at all"; end;
    end;
  end;
end;
```

4.12 Press

Press data is defined in the following complex definition:

```
def press
  . . .
end;
```

Moreover, the following additional wizards are available:

- `pressReadership` readership of selected titles, sorted alphabetically,
- `pressRanking` readership of selected titles, sorted by the first readership indicator,
- `pressCmpWaves` readership trends,
- `pressCmpGroups2` compares readership in two target groups (a simpler version of the `pressCmpGroups` wizard),
- `pressCmpGroups` compares readership in several target groups,
- `pressCoreadership` coreadership of selected titles,
- `pressStructure` readership structure,
- `pressStructureRow` readership in selected groups of respondents,
- `pressPrices` analysis of price lists,
- `pressMediaPlan` media-plan,
- `pressMediaPlanOpt` media-plan optimizer.

To include all of the above wizards, a single wizard identifier suffices: `press`.

Note
Press definitions are not allowed in the Lite version of YAC Data Builder.

4.12.1 Initial definitions

At the start of press definitions, the following field should be defined first:

- `hook`

  `"hook" "=" module_name ";"`

  Defines the module in the hierarchy of questions where the automatically generated press questions will be placed.

4.12.2 Indicators

In the definition **`def`** `indicators . . .` **`end`**`;` place the definitions of the consecutive indicators according to the following template:

```
def indicator
  id = spo;
  name = "Spontaneous awareness (%)";
  info = "Spontaneous awareness of the press title (of the group of press titles)";
end;
```

In the example above we have standard fields, however the indicator's identifier has a predefined meaning (thus, the above definitions allow you to specify which indicators are actually available in the data and what texts should be shown when displaying the indicators or information about them).

The following identifiers are supported:
- `tom` - top of mind,
- `spo` - spontaneous awareness,
- `awa` - prompted awareness,
- `distrib` - circulation,
- `er` - ever read,
- `scr` - season cycle readership,
- `rr` - regular reader,
- `lir` - last issue readership,
- `nir` - number of issues read in the season cycle,
- `mnir` - mean number of issues read in the season cycle,
- `ar` - average issue readership (average reach),
- `cppg`, `cppr`, `cpper` - Cost Per Point (GRP, Reach, Effective Reach),
- `cpmg`, `cpmr`, `cpmer` - Cost Per Thousand (GRP, Reach, Effective Reach),
- `cpi` - Cost Per Insertion,
- `coi` - Cost Of Insertions,
- `noi` - Number Of Insertions,
- `fd` - Opportunities To See (frequency distribution),
- `fdplus` - Opportunities To See (plus),
- `af` - Average Frequency (average issue readership),
- `grp` - Gross Rating Points,
- `gi` - Gross Impressions ('000),
- `nci` - number of contacts with an issue,
- `mnci` - mean number of contacts with an issue,
- `rpc` - readers per copy,
- `mrpc` - mean number of readers per copy,
- `pir` - part of issue read.

Other identifiers are not currently supported.

Since indicators need the information on which columns their data is saved, the following fields should also be defined:

- `column`

  `"column" "=" column_name ";"`

  Defines the column in the data for the indicator (used when the indicator is based on a single column).

- `columnList`

  `"columnList" "=" column_name [ "," column_name ] ";"`

  Defines a set of columns in the data for the indicator (used when the indicator is based on a set of columns).

- `numeric`

  `"numeric" "=" code_list ";"`

  Defines the set of valid values for the indicator.

- `columnMask`

  `"columnMask" "=" column_mask ";"`

  Defines a set of columns in the data for the indicator; used when the data for each press title is kept in a separate column.

  In the `column_mask` specification, the "*" (asterisk) character must be used. When creating the column name for a title with the identifier A, the identifier A is placed in the `column_mask` in place of the asterisk. For instance, the mask can be defined as follows:

  ```
  columnMask = "q1_*" ;
  ```

  Then, for the title with the identifier A, the column name will become q1_A.

- `independentColumns`

  `"independentColumns" "=" ( 0 | 1 ) ";"`

  Independent columns define a single column for each title in a multi-choice indicator (dichotomous variables). Dependent columns define a set of columns where the codes of the titles are saved.

4.12.3 Regions

In this section you can define regions that can later be used to specify regions where titles are circulated.

In the definition **`def`** `regions . . .` **`end`**`;` place the definitions of the consecutive regions using the following template:

```
def region
  id = r01;
  name = "Warsaw";
end;
```

Identifiers defined here will be used in the `regions` instruction when defining press titles. Region names will be displayed in YAC Data Analyzer and can be used there to search the list of press titles by regions.

4.12.4 Titles

Radio stations are defined in complex definitions `module` and `title`.

Modules can be used just the same way as in question definitions to define hierarchies. Press titles can be defined inside modules, as well as other modules can be defined inside modules. Titles can also be defined outside of modules (directly in the `press` definition).

Module definitions are the same as question module definitions, so let's go straight to press title definitions:

```
def title
  id = TA;
  name = "Title A";
  regions = all;
end;
def title
  id = TB;
  name = "Title B";
  regions = r01,r19,r35,r63,r69,r93;
end;
```

Two titles are defined in the example above. Next to standard fields you can define regions where these titles are circulated.

The definition `regions = all;` means that the title is circulated in all regions (this is equivalent to not specifying the regions at all).

The definition of the second title lists a set of regions (here we use GUS numbering of the old 49 voivodeships). Identifiers used here must be the same as the identifiers of regions used in the **`def`** `regions` `. . .` **`end`**`;` definition.

4.13 Radio

Radio data is defined in the following complex definition:

```
def radio
  . . .
end;
```

Moreover, the following additional wizards are available:

- `radioAudience` audience of selected stations, sorted alphabetically,
- `radioRanking` audience of selected stations, sorted by the first indicator,
- `radioCmpWaves` audience trends,
- `radioCmpGroups2` compares audience in two target groups (a simpler version of the `radioCmpGroups` wizard),
- `radioCmpGroups` compares audience in several target groups,
- `radioColistening` colistening of selected stations,
- `radioStructure` audience structure,
- `radioStructureRow` audience in selected groups of respondents,
- `radioDays` audience in selected days of the week,
- `radioQs` audience in selected quarters of an hour,
- `radioPlaces` audience by selected places of listening,
- `radioSources` audience by selected sources of radio signal.

To include all of the above wizards, a single wizard identifier suffices: `radio`.

Note
Radio definitions are not allowed in the Lite version of YAC Data Builder.

4.13.1 Initial definitions

At the start of radio definitions, the following fields should be defined first:

- `hook`

  `"hook" "=" module_name ";"`

  Defines the module in the hierarchy of questions where the automatically generated radio questions will be placed.

- `dayColumn`

  `"dayColumn" "=" column_identifier ";"`

  Specifies the column in the data file where the data on the day of the interview is saved.

4.13.2 Indicators

In the definition **`def`** `indicators . . .` **`end`**`;` place the definitions of the consecutive indicators according to the following template:

```
def indicator
  id = spo;
  name = "Spontaneous awareness (%)";
  info = <<
    Spontaneous awareness of the station (of the group of stations) R
    is equal to the number of respondents who,
    without hearing the list of radio stations,
    declared awareness of the station
    (of at least one station in the group) R.
  >>;
end;
```

In the example above we have standard fields, however the indicator's identifier has a predefined meaning (thus, the above definitions allow you to specify which indicators are actually available in the data and what texts should be shown when displaying the indicators or information about them).

The following identifiers are supported:
- `spo` - spontaneous awareness,
- `awa` - prompted awareness,
- `week` - weekly reach,
- `day` - daily reach,
- `qs` - reach in quarters,
- `share` - market share,
- `meanTime` - mean audience time,
- `avgQ` - average quarter audience.

Other identifiers are not currently supported.

4.13.3 Sources

In the **`def`** `sources . . .` **`end`**`;` definition list the definitions of the consecutive sources of signal using the following template:

```
def source
  id = rso;
  name = "Aerial antenna";
end;
```

In the example above we have standard fields, however the identifiers have predefined meanings:
- `rso` - aerial,
- `web` - Internet,
- `sat` - satellite,
- `cab` - cable.

Other identifiers are not currently supported.

4.13.4 Places

In the **`def`** `places . . .` **`end`**`;` definition list the definitions of the consecutive places of listening using the following template:

```
def place
  id = home;
  name = "At home";
end;
```

In the example above we have standard fields, however the identifiers have predefined meanings:
- `home` - at home,
- `work` - at work,
- `car` - in a core,
- `other` - in other places.

Other identifiers are not currently supported.

4.13.5 Regions

In this section you can define regions that can later be used to specify regions where stations are available in.

In the definition **`def`** `regions . . .` **`end`**`;` place the definitions of the consecutive regions using the following template:

```
def region
  id = r01;
  name = "Warsaw";
end;
```

Identifiers defined here will be used in the `regions` instruction when defining radio stations. Region names will be displayed in YAC Data Analyzer and can be used there to search the list of radio stations by regions.

4.13.6 Stations

Radio stations are defined in complex definitions `module` and `station`.

Modules can be used just the same way as in question definitions to define hierarchies. Radio stations can be defined inside modules, as well as other modules can be defined inside modules. Stations can also be defined outside of modules (directly in the `radio` definition).

Module definitions are the same as question module definitions, so let's go straight to radio station definitions:

```
def station
  id = S0A;
  name = "Polish Radio 1";
  regions = all;
end;
def station
  id = S1E;
  name = "RADIOSTATION";
  regions = r01,r19,r35,r63,r69,r93;
end;
```

Two stations are defined in the example above. Next to standard fields you can define regions where these stations are available.

The definition `regions = all;` means that the station is available in all regions (this is equivalent to not specifying the regions at all).

The definition of the second station lists the set of regions (here we use GUS numbering of the old 49 voivodeships). Identifiers used here must be the same as the identifiers of regions used in the **`def`** `regions` `. . .` **`end`**; definition.

4.14 TV

TV data is defined in the following complex definition:

```
def tv
  . . .
end;
```

Moreover, the following additional wizards are available:

- `tvAudience` audience of selected stations, sorted alphabetically,
- `tvRanking` audience of selected stations, sorted by the first indicator,
- `tvCmpWaves` audience trends,
- `tvCmpGroups2` compares audience in two target groups (a simpler version of the `tvCmpGroups` wizard),
- `tvCmpGroups` compares audience in several target groups,
- `tvCoviewing` coviewing of selected stations,
- `tvStructure` audience structure,
- `tvStructureRow` audience in selected groups of respondents,
- `tvDays` audience in selected days of the week,
- `tvQs` audience in selected quarters of an hour,
- `tvDaysQs` audience in selected days of the week and quarters of an hour.

To include all of the above wizards, a single wizard identifier suffices: `tv`.

Note
TV definitions are not allowed in the Lite version of YAC Data Builder.

4.14.1 Initial definitions

At the start of TV definitions, the following fields should be defined first:

- `daysCol`

  `"daysCol" "=" column_identifier ";"`

  Specifies the column in the data file where the data on the day of the interview is saved (with values from 1 - Monday to 7 - Sunday).

- `minutesCol`

  `"minutesCol" "=" column_identifier ";"`

  Specifies the column in the data file where the data on the number of minutes spent watching a TV station is saved.

- `qsCol`

  `"qsCol" "=" column_identifier ";"`

  Specifies the column in the data file with the identifier of the quarter of an hour is saved (with values from 1 - the first quarter in the day to 96 - the last quarter in the day).

- `stationsCol`

  `"stationsCol" "=" column_identifier ";"`

  Specifies the column in the data file with the TV stations' code is saved.

4.14.2 Indicators

In the definition **`def`** `indicators . . .` **`end`**`;` place the definitions of the consecutive indicators according to the following template:

```
def indicator
  id = amr;
  name = "Average Minute Rating";
  nick = "AMR";
  info = <<
    Average Minute Rating\n
    \n
    The description of the indicator goes here...>>;
end;
```

In the example above we have standard fields, however the indicator's identifier has a predefined meaning (thus, the above definitions allow you to specify which indicators are actually available in the data and what texts should be shown when displaying the indicators or information about them).

The following identifiers are supported:
- `amr` - Average Minute Rating,
- `atv` - Average Time Viewing,
- `shr` - share,
- `rch` - reach.

Other identifiers are not currently supported.

4.14.3 Stations

TV stations are defined in complex definitions `module` and `station`.

Modules can be used just the same way as in question definitions to define hierarchies. TV stations can be defined inside modules, as well as other modules can be defined inside modules. Stations can also be defined outside of modules (directly in the `tv` definition).

Module definitions are the same as question module definitions, so let's go straight to radio station definitions:

```
def station
  id = tvp1;
  name = "TVP1";
  code = 1;
end;
def station
  id = tv4;
  name = "TV 4";
  code = 6
end;
```

Two stations are defined in the example above.

Next to standard fields you can must define the station's code (that is saved in the `stationsCol` column in the data file).

Chapter V

5 Annexes

5.1 Annex A - Language Identifiers

afk	Afrikaans
sqi	Albanian
arg	Arabic (Algeria)
arh	Arabic (Bahrain)
are	Arabic (Egypt)
ari	Arabic (Iraq)
arj	Arabic (Jordan)
ark	Arabic (Kuwait)
arb	Arabic (Lebanon)
arl	Arabic (Libya)
arm	Arabic (Morocco)
aro	Arabic (Oman)
arq	Arabic (Qatar)
ara	Arabic (Saudi Arabia)
ars	Arabic (Syria)
art	Arabic (Tunisia)
aru	Arabic (U.A.E.)
ary	Arabic (Yemen)
hye	Armenian
aze	Azeri (Cyrillic)
aze	Azeri (Latin)
euq	Basque
bel	Belarusian
bgr	Bulgarian
cat	Catalan
zhh	Chinese (Hong Kong S.A.R.)
zhm	Chinese (Macau S.A.R.)
chs	Chinese (PRC)
zhi	Chinese (Singapore)
cht	Chinese (Taiwan)
hrv	Croatian
csy	Czech
dan	Danish
div	Divehi
nlb	Dutch (Belgium)
nld	Dutch (Netherlands)
ena	English (Australia)
enl	English (Belize)
enc	English (Canada)
enb	English (Caribbean)
eni	English (Ireland)
enj	English (Jamaica)
enz	English (New Zealand)
enp	English (Philippines)
ens	English (South Africa)
ent	English (Trinidad)
eng	English (United Kingdom)
enu	English (United States)
enw	English (Zimbabwe)
eti	Estonian
fos	Faeroese
far	Farsi
fin	Finnish
frb	French (Belgium)
frc	French (Canada)
fra	French (France)

frl	French (Luxembourg)
frm	French (Monaco)
frs	French (Switzerland)
mki	FYRO Macedonian
glc	Galician
kat	Georgian
dea	German (Austria)
deu	German (Germany)
dec	German (Liechtenstein)
del	German (Luxembourg)
des	German (Switzerland)
ell	Greek
guj	Gujarati
heb	Hebrew
hin	Hindi
hun	Hungarian
isl	Icelandic
ind	Indonesian
ita	Italian (Italy)
its	Italian (Switzerland)
jpn	Japanese
kan	Kannada
kkz	Kazakh
knk	Konkani
kor	Korean
kyr	Kyrgyz (Cyrillic)
lvi	Latvian
lth	Lithuanian
msb	Malay (Brunei Darussalam)
msl	Malay (Malaysia)
mar	Marathi
mon	Mongolian (Cyrillic)
nor	Norwegian (Bokmal)
non	Norwegian (Nynorsk)
plk	Polish
ptb	Portuguese (Brazil)
ptg	Portuguese (Portugal)
pan	Punjabi
rom	Romanian
rus	Russian
san	Sanskrit
srb	Serbian (Cyrillic)
srl	Serbian (Latin)
sky	Slovak
slv	Slovenian
ess	Spanish (Argentina)
esb	Spanish (Bolivia)
esl	Spanish (Chile)
eso	Spanish (Colombia)
esc	Spanish (Costa Rica)
esd	Spanish (Dominican Republic)
esf	Spanish (Ecuador)
ese	Spanish (El Salvador)
esg	Spanish (Guatemala)
esh	Spanish (Honduras)
esn	Spanish (International Sort)
esm	Spanish (Mexico)
esi	Spanish (Nicaragua)
esa	Spanish (Panama)
esz	Spanish (Paraguay)
esr	Spanish (Peru)
esu	Spanish (Puerto Rico)

esp	Spanish (Traditional Sort)
esy	Spanish (Uruguay)
esv	Spanish (Venezuela)
swk	Swahili
svf	Swedish (Finland)
sve	Swedish
syr	Syriac
tam	Tamil
ttt	Tatar
tel	Telugu
tha	Thai
trk	Turkish
ukr	Ukrainian
urd	Urdu
uzb	Uzbek (Cyrillic)
uzb	Uzbek (Latin)
vit	Vietnamese

5.2 Annex B - Examples of Column Definitions

5.2.1 Simple Defintions

```
WAVE L3
NRANK L4
SEX L1
AGE L1
EDU L1
HHSIZE L1
HHBABY L2
INCOME L2
VOIVOD L2
MOTHER L1
MB_AUD L1
MB_VGA L1
MB_LAN L1
PCCORE L1
PCSPEED L2
PCMEM L1
HDBRAND L1
HDSIZE L1
CDBRAND L1
CDKIND L1
MONBRAND L1
MONSSIZE L1
NPRINTER L1
IPRINTER L1
LPRINTER L1
SCANNER L1
WEIGHT L10
```

5.2.2 SPSS Compatible Definitions

```
FILE HANDLE DEMO /NAME='Demo.dat' /LRECL=45.
DATA LIST FILE=DEMO RECORD=1/
WAVE 1-3
NRANK 4-7
SEX 8
AGE 9
EDU 10
HHSIZE 11
HHBABY 12-13
INCOME 14-15
VOIVOD 16-17
MOTHER 18
MB_AUD 19
MB_VGA 20
MB_LAN 21
PCCORE 22
PCSPEED 23-24
PCMEM 25
HDBRAND 26
HDSIZE 27
CDBRAND 28
CDKIND 29
MONBRAND 30
MONSSIZE 31
NPRINTER 32
IPRINTER 33
LPRINTER 34
SCANNER 35
WEIGHT 36-45 (F)
.
```

5.3 Annex C - Changes

Changes from previous versions are described below.

Version 4.13.a released 2012-05-29

Changes in the YAC Data Analyzer application only.

Version 4.13 released 2011-02-03

Fixed `excludeSysMis` handling.
Added `hideRowsCols` definition that sets the default handling of empty rows and columns in YDA.

Version 4.12 released 2010-09-19

Added response attribute `substStats` that automatically changes results in YDA from percents to means.

Version 4.11 released 2010-08-15

Added:
- `fixedRecordsets` directive that, when set, blocks changing of the record set in an existing analysis.
- `stdStats and advStats` directives for defining standard and advanced statistics.
- `stats` directive in record set definition for limiting available statistics for analyses based on that record set.
- `hidden` directive in record set definition for blocking analyses based on that record set.

Version 4.10 released 2010-03-07

Changes in the YAC Data Analyzer application only.

Version 4.03 released 2010-01-14

Changes in the YAC Data Analyzer application only.

Version 4.02 released 2009-10-04

Added:
- `expires` instruction in the `survey` section to define the date up to which the data will be accessible.
- `minSelCount` instruction in the `waves` section to define the minimum number of waves for analyses.

Version 4.01 released 2009-07-24

Added license control for survey files based on hardware keys.

Version 4.00 released 2009-06-01

Support for TV audience data.

Version 3.04 released 2008-12-15

- Registering file extension associations.
- Apart from that, changes in the YAC Data Analyzer application only.

Versions 3.03.a - 3.03.d released between 2008-05-01 and 2008-08-24

Small fixes only.

Versions 3.00 - 3.03 released between 2007-09-29 and 2008-04-15

- Right margin added that displays positions of all issues (errors, warnings, and hints).
- Added sections on press and radio definitions in the English version of this document.
- Measurement scales can be defined for questions.
- Measurement scales and noMean attributes can be automatically imported from SPSS data files.
- Measurement scales are displayed in the import dialog window (for each variable).
- More examples of question definitions (based on the General Social Survey).
- Various small fixes.

Versions 2.30 - 2.30.b released between 2006-07-31 and 2006-09-02

Changes in the YAC Data Analyzer application only.

Version 2.29 released 2006-06-04

- Changes in survey protection and license management; new definition `licenses`.
- New definition `logoPlacement`.
- Instruction `logo` now also handles JPEG and GIF formats.
- Translated YAC Data Builder documentation into English.

Version 2.28 released 2005-09-26

- A free Lite version is available for download; handles files of up to 1100 cases and 100 columns.
- A demo data file is distributed with the program.

Version 2.27 released 2005-08-03

- Handling of long variable names in SPSS data - up to 64 bytes.
- Changes to licensing of YAC Data Builder and YAC Data Analyzer.
- Changes in survey data protection.

Version 2.26 released 2005-03-16

Additional menu **File | Append block...** adds the highlighted text in the editor to an existing file (**File | Save block...** replaces the contents of a file with the highlighted text).

Version 2.25 released 2004-11-25

New wizard: `pressReadership` (instruction `wizards`).

Version 2.24 released 2004-10-26

New wizards: `cmpGroups2`, `genComplex`, `pressCmpGroups2`, `pressStructureRow` (instruction `wizards`).

Version 2.23

In this version, most changes were made to the YAC Data Analyzer application. Only small changes were made to YAC Data Builder.

Version 2.22

Language:
- The response attribute `noMean` was added that excludes given responses from the calculation of means.

Version 2.21

Language:
- The instruction `excludeSysMis` was added to the `survey` definition; it defines how missing values are treated by default in calculations.

Version 2.20

Language:
- Definitions of multi-dimensional questions.
- New response attribute: `allowOverlaps`.

Data import:
- Automatic naming of questions, provided that variables have similar labels.
- Copying of modules, questions, and columns.
- Changing the order of elements.
- Inclusion / exclusion of elements into / out of existing definitions.
- Saving / opening import definitions.
- Variable information preview (value labels, missing data).
- Automatic definitions of multi-dimensional questions.
- Automatic generation of the survey name based on the imported file name.
- Semi-automatic selection of variables into questions.
- Automatic saving of import settings into a `.dbi` file.

Fixes / enhancements:
- The function **Processor | Run YDA...** was fixed - YAC Data Analyzer didn't open a survey if the survey's file name contained blanks.

Version 2.10

New:
- Integration of data processing with data import, fully functional editor and automatic execution of YAC Data Analyzer with the processed data.
- Instruction `independentColumns` in press indicator definitions.

Fixes / enhancements:
- Added the description of an undocumented key `attr`.

Version 2.00

New definitions:
- New instruction `logo` in the `survey` definition that defines the logo of the company that is distributing the data; shown in YAC Data Analyzer in the status bar.
- New wizard: press media plan optimizer (`pressMediaPlanOpt` value in the `wizards` key in the `survey` definition).

Fixes / enhancements:
- The key `demo` in the `survey` definition is used to display a warning message in YAC Data Analyzer that data is not representative.
- Version is controlled in the database of licenses:
  - `press.media.plan` implies `general`,
  - `press.media.plan.opt` implies `press.media.plan`.

  If the definition is not consistent with the above rules, an error will be reported.
- Added the description of an undocumented format of multi-choice questions. If codes of the selected responses are used, and not dichotomous variables, the `columnList` key may be used.

Version 1.20

- Handling of press data (instruction `press`).
- Handling of protected surveys (licenses).
- New instructions: `version` and `protected` in the `survey` definition.

Version 1.10

- Handling of radio data (instruction `radio`).
- Additional field added for defining the format of the column specification in data files (`colSpecFormat`).

Version 1.00

- Values can be grouped inside a single response (instruction `range`).
- Numeric responses added (instruction `numeric`).

Chapter VI

6 Dialog Windows

6.1 File | Import...

This is a tool to support importing data from other formats and applications.

Currently, importing from SPSS is supported (files with the `.sav` extension).

Imported files are converted to the Fixed-ASCII format that later will be converted by YAC Data Builder into a format readable by YAC Data Analyzer.

6.1.1 Options

This tab defines additional options for data import.

Define scales is used to automatically define measurement scales based on variable definitions in the data being imported.

Define noMean attributes automatically defines this attribute for missing values defined in the data being imported:
- if **Define scales** is checked, this attribute will be defined for interval questions only,
- if **Define scales** is unchecked, this attribute will be defined for all missing values.

Index

Q

R

S

T

W

Y