Reinhold Scheck

THE rS1.METHOD

Planning, developing,
structuring, and dynamizing calculation models with

MICROSOFT EXCEL 2007 and beyond

(Descriptions apply analogously to Excel 97 and later versions)

Part 1: Application of the Method to Workbooks, Sheets, and Tables

The rS1.Method

Part 1
Application of the Method to Workbooks, Sheets, and Tables

This text provides supplementary information for a book. I have compiled most of the example files stored on the CD accompanying the book according to the rules of the rS1.Method. The book deals mainly with development of dynamic solutions that do not require any programming. That is why I have not included in this text those parts of the rS1.Method that describe specifications for programming Microsoft Excel.

Table of contents

1 GENERAL INFORMATION

The rS1.Method serves as a guideline for structuring and standardizing development in Microsoft Excel. This may at first sound boring, but the reality is far different, as you will soon learn and hopefully experience through the future success of your work. Whether the method is applied to simple table models or extensive, complex, and dynamic calculation solutions, it is hard to ignore its capabilities. It makes your work easier, your procedure clearer, and your results more attractive.

1.1 About the Development of the rS1.Method

I have been working intensively with Microsoft Excel since 1989 and have been annoyed time and again by the same problems that crop up right at the beginning of each project. Not so much by problems in Excel—of which there are only a few—but more by self-made ones. These involve quickly creating one's own solutions and continuously extending them functionally, step by step, bit by bit, always focused only on the results. To put it negatively, one could also call this "uncontrolled growth." The solutions usually worked well, some of them even very well, but because they lacked structure they had various disadvantages and deficiencies:

- The results were unclear and barely comprehensible to others.
- The solutions used methods that were very difficult to interpret after a while, even for the developer.
- Similar problems were solved unsystematically in different ways without selecting a "best method."
- Solutions could not be transferred easily to similar problems, or couldn't be transferred at all.

It was a lucky coincidence that at this time US macro programmers defined and published a method to arrange the so-called Excel macro template in a neat and useful manner. To summarize: The macro template, which is still available with Excel 2007, is a special sheet in table form that allowed efficient, visually attractive programs to be written in Excel, even in those early days. This method has been largely forgotten since 1993 due to the introduction of VBA, but the old programs created in this way still work. This is a tribute to the Excel developers who have persevered to retain downward compatibility in spite of numerous—and to some extent unnecessary—version updates. The basic concepts followed by these American colleagues were the consistent and functionally orientated assignment of names to macros, and the clearly structured notation of the program sequences and their modular division into various areas of the template. While incorporating these ideas, I have developed a similar structuring method for VBA projects. Even more important, I have, transferred my system to those Excel models and solutions that function without any program code but still create similar or even higher demands for an easily comprehensible body of rules. The results were astonishingly convenient and have remained so to this day. They include the following:

- You can search for something and find it.
- Even better: You know beforehand what *not* to search for and where not to search for it.
- Even months or years later, you can understand without great effort even highly complicated formulas that have not been used for a long time.
- Experts and users of the method can understand the contents and solutions in other people's models, without needing long explanations.
- Your solution can be easily and elegantly applied to similar problems.

Some parts of this system were published as the "LS1 method" in 1994. Given the very short innovation cycles of our industry, the rS1.Method is already ancient, though this is by no means an argument against it. In all those years it has been used in numerous independent projects, for contract work, and in complex business solutions, subject to constant minor changes and additions. I wasn't very interested in its distribution, mainly because I was concerned about protecting development expertise. It was fully published—as "rS1.Method"—for the first time in 2003.

1.2 General Advantages

As a user of the rS1.Method, you can meet a number of requirements that you'll find in the strictest professional programming guidelines. In addition, you can extend those high standards to models that have nothing to do with programming. This allows you to make each Excel solution clear and comprehensible, regardless of the type and task. You can benefit from such advantages not only during the design phase, but beforehand and especially afterwards. The mere existence of defined general conditions allows for considerations that otherwise wouldn't come into play. There is, for example, a sheet with the name *Parameters 1* in the rS1 workbook. When I open my rS1 *Excel template* in order to begin my work—or even before then—I reflect on what parameters are, whether my model already has parameters or needs them, which parameters of the customer would be useful for the solution, and other similar concerns. It is not uncommon—as I'm sure you've experienced—for ideas to appear before beginning work that would never have appeared without the existence of the parameter sheet. And once this has been internalized, planning parameters will soon become a natural part of your preparation.

The most impressive advantage of the rS1.Method *after* the development phase of a model is the considerable facilitation of administration and support. If, as "architect" of my own solution, I can follow and understand its total structure, its design, and its details at any time, this means I can also answer all questions, in particular those which I ask myself. Maybe I won't like a result as much as I used to do, or maybe I won't like it at all. Or, that I may be secretly surprised that the user *still* likes it. However, if I can easily and quickly comprehend what I did and why I did it months or even years later, whether the issue is "plaster" or "offals," I can also easily and quickly change or improve it. What's more, I've created solutions with easily replaceable design elements, because using the rS1.Method encourages (but does not force) you to follow certain procedures. This means—often to your great relief—that you can copy large parts from a calculation model, integrate them in a new project, adapt them there (*find and replace* is usually sufficient), and you're done.

All this applies to the “before” and “after” phases. At this point, we don’t need a detailed description of the advantages during the design phase itself. Since, if you work according to the specifications described here, you will find out and hopefully appreciate which direct and indirect benefits can be drawn for your daily work with Microsoft Excel by applying the rS1.Method. What initially appears to be slightly awkward and seems to take time to get used to opens up new and, in particular, secure ways of working successfully with Microsoft Excel.

2 BASIC RULES OF THE METHOD

Basic rule 1 Program only if necessary or useful

All options are made use of when creating dynamic Excel solutions with table functions. Programming is only performed if it is absolutely necessary or particularly useful.

Basic rule 2 Variables are preferable to constants

Constants are avoided whenever the same purpose can be achieved with variables. This applies also to table formulas.

Basic rule 3 Use redundancy whenever it is an advantage

Redundancy is exercised whenever it can be of help with subsequent verification and editing.

Regarding basic rule 1 (programming only if necessary or useful)

There are many efficient solutions that I call “non-program applications.” These are models that use dynamic components or are even highly dynamic (e.g., fully controllable by mouse clicks). They fulfill complete and complex tasks or provide specialized services for specialists, are well suited to the workstations used by Excel laymen, the computer on the CEO’s desk, or managers' notebooks, and may therefore can truly be called “applications” even if they do not contain a single line of program code. The rS1.Method does without programming if the tasks to be solved can be dealt with completely and efficiently with a table model. You can use control elements and table functions methodically and skillfully to push the limits of the directly formulated requirements. Simply said: Everything you can do without programming—which is a great deal—you *should* do without programming.

Regarding basic rule 2 (variables are preferable to constants)

Constants are to be avoided whenever the purpose can be achieved with variables. It may come as a surprise that this rule should also apply to sheets. It is important, however, especially in large calculation models that may consist of thousands of formulas, that most of the formulas take the values of their arguments from the outside. This applies even if it seems unlikely that a formula will ever require values other than those defined during its initial application. The required procedure is illustrated in *Figure 1*.

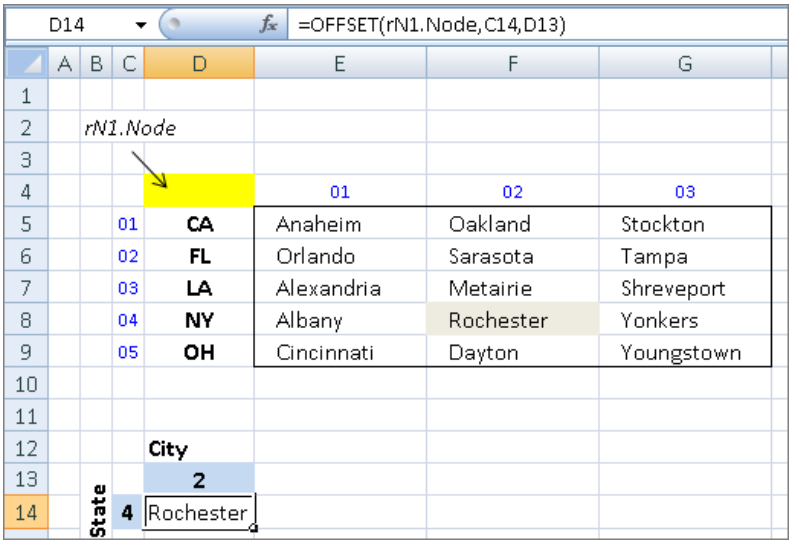


Fig. 1:
The formula in cell D14 takes the values of its arguments from the outside

The formula in cell *D14* is a typical rS1 formula. Starting at cell *D4*, with the range name *rN1.Node*, it uses the variables from *D13* and *C14* as arguments for determining the referenced point of intersection of the row and column (State = 4, City = 2) within the range *E5:G9* (details of the `OFFSET` function appear later in Section *3.2)*.

Regarding basic rule 3 (use redundancy whenever it is of advantage)

I find inappropriate and also absurd the argument that one can, should, or must "save paperwork" when developing an application. I simply cannot see who is supposed to benefit. What use are neatly abbreviated, elegantly shortened, and quickly written formulas or program lines to me if I (let alone those following me) may not be able to understand them in the future? And are software programs really made slower by redundant notation? Or is a table formula, or are 10,000 table formulas, really processed more quickly if I do without all unnecessary arguments (we're talking about time as perceived by the user, not about milliseconds)? In short, redundancy is exercised in the rS1.Method whenever it is of long-term benefit. This is the case whenever redundant notation is used to help search for, find or change something, or make it easier to understand. A small example of a formula is illustrated in *Figure 2*.

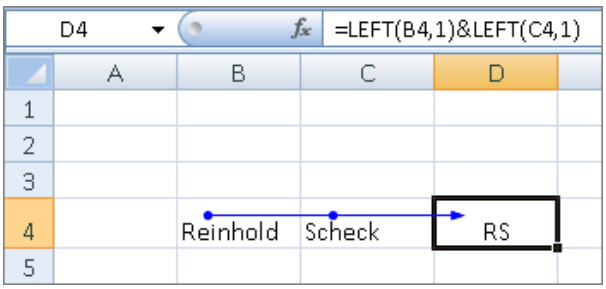


Fig. 2:
Though some formula arguments are unnecessary, they are useful

The formula `=LEFT(B4,1)&LEFT(C4,1)` in cell *D4* determines the initials of a name from the cells *B4* and *C4*. If the second argument—`num_chars`—is missing in the `LEFT` function,

Excel assumes that the value of this argument is *1*, which means that only one character is to be output. With the rS1.Method, the formula also includes such optional arguments even if, as in this case, they are not required by Excel. *You* may need them; if, for example, you are searching for an error in a much more complicated formula with 250 or more characters and numerous parentheses and need to understand what is calculated in detail there, argument by argument.

The `LEFT` function

The function `=LEFT(text,num_chars)` shows as a result as many `text` characters as correspond to the argument `num_chars`. If cell *K11* contains the text "Limerick" the result of the formula `=LEFT(K11,3)` is "Lim".

The text operator &

You can use an ampersand (i.e., the character `&`) to combine text. If *K11* contains the first name "Michael" and *J11* the surname "Friel" the result of the formula `=K11&" "&J11` is "Michael Friel". It links the content of *K11* with a space to the content of *J11*.

Basic rule 4

The fourth and basic rule, not yet mentioned, is that you should not think basic rules are more important than they actually are. If you come to terms with the specifications of the method and work successfully, that is excellent. If you implement the rules only partially and otherwise use them as basis for your own variants (i.e., implement your own mixed solution) that is also fine. Your models and solutions would ultimately also work without rS1.Method. Not as well arranged by far, hardly as easy to maintain, and certainly not with values as stable. But they would work. No matter how you decide to deal with this, the most important recommendation is that you always remain consistent in how you apply rules and methods. It is worth the effort.

3 RULES FOR WORKBOOKS AND SHEETS

This section deals with rules and recommendations for workbooks and sheets. You can find information on the following subjects:

- Workbooks and sheets and their design
- Range names and how to use them
- Object names and how to use them
- Particularities, recommendations, and notes
- Excel templates

3.1 Workbooks and Their Design

rS1 workbooks consist—at least during the development phase of models—of at least six sheets. You can find out more about their designation and arrangement further on. First, here are a few pieces of advice on the organization of work and designation of files.

3.1.1 File names and folders

Unfortunately, you can make many mistakes when developing an Excel model. A frequent problem is failure to save intermediate results. It is not uncommon for you to be forced to go back two, three, or four steps in order to correct faulty development. It is good if these steps can be made at all; i.e., if there are traces that you can follow. For this reason, I have imposed a system and procedure that already has helped prevent small problems from becoming disasters during numerous projects.

Here are some details of *Figure 3*, a Windows Explorer view of Windows Vista:

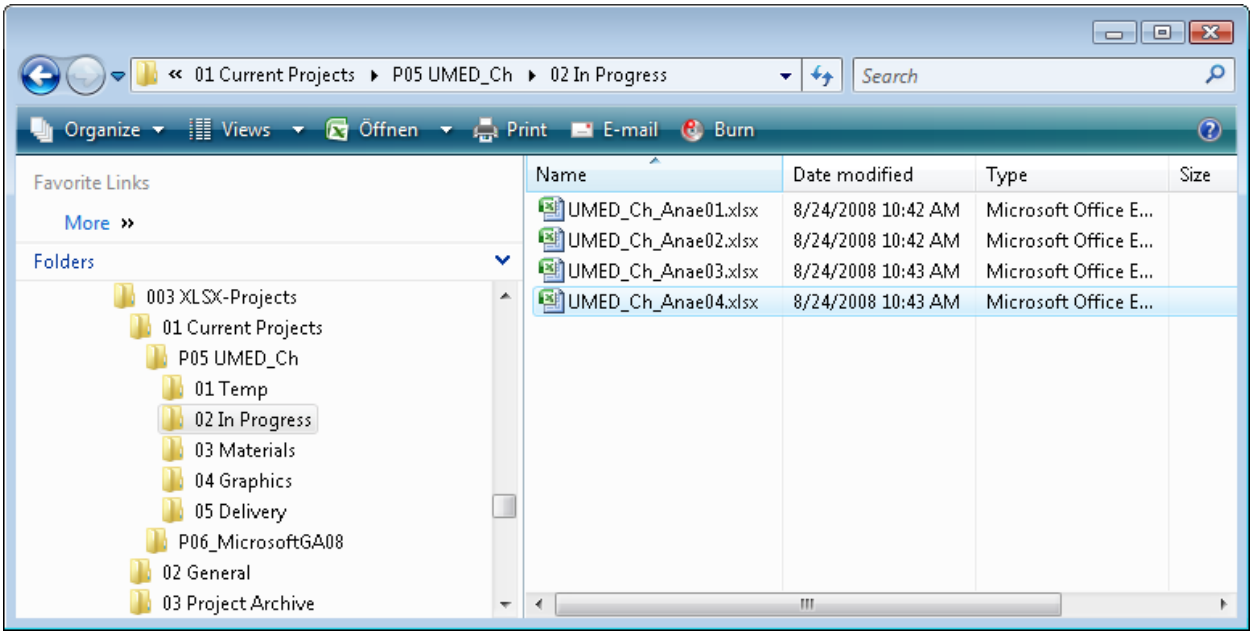


Fig. 3: The structure of the development folder

- At the top of the figure you see the path of the file that is currently marked.
- On the left side is the data-storage structure with the actual names of the folders.
- At the top right are the names of those files saved to the currently active folder.

You can see that all folder names begin with numeric prefixes. This has the great advantage that it enables you to create an access system without having to rely on giving names in alphabetical order.

Level 1 is a folder in which Excel 2007 projects are collected. The second level of the system comprises the following three folders:

- 01 Current projects
- 02 General
- 03 Project archive

01 Current Projects: This folder contains all projects currently being processed. Processing is regarded as the entire period of time from the first sketch of an idea to the handover of the usable product. Only when the product has been completely finished or distributed and dispatched are the corresponding files archived. However, they should then no longer be in *01 Current Projects*.

On the subordinate level 3, you can see two projects that are being worked on and are designated there as *P05 UMED_Ch* and *P06_MicrosoftGA08*. These designations follow a certain system: a prefix, such as *P05* refers to the consecutive number of the project (naturally, you often may have to work on several projects simultaneously). The prefix is followed by a name that assigns the project to a customer or subject as clearly as possible.

There are several standardized folders in a defined order on level 4 for each project:

- *01 Temp*: This is a sort of clipboard. I can deposit everything here that I currently cannot or do not want to assign in any other way or that I only need temporarily.
- *02 In Progress*: The project files are stored here in chronological order of development progress. In other words, different versions of the project are stored here. During complicated development work, it is useful not only to save (which is done anyway), but also to store any intermediate results judged to be useable as versions after each successful work step. The file names end, as illustrated at the top right in *Figure 3*, with a consecutive version number.
- *03 Materials*: This folder contains all usable items (provided they are not graphics) that belong to the project and its setup: planning data, raw data, preliminary studies, mock drafts, tables, texts, etc.
- *04 Graphics*: All graphics belonging to the project are stored here. In this folder I collect objects that I bring together from larger general stocks (e.g., collections of pictures), adapt project-specifically, and save under a new name.
- *05 Delivery*: The final version of the product or the version to be delivered is stored here (an internal transfer is also regarded as a "delivery" in this respect). The folder's job is to ensure that this, and only this, is the published final version. This provides a sense of security when potential conflicts arise.

02 General: This second-level folder contains in five further sub-categories, everything of a general nature, which is therefore not assigned to a current or archived project but should still be constantly accessible by the developer of an Excel solution.

03 Project Archive: In order to rely on good experiences and useable results, it is very important to keep successful work in a transferable condition and archive it. The folder structure should therefore be supplemented by an archive folder.

This procedure not only makes it easier to save attempts of any kind, but also helps to implement new and similar projects with considerably less effort. The saved preliminary or intermediate stage of the old project can spare you a great deal of preparatory work or at least reduce it considerably.

3.1.2 Sheets and their contents

Before Excel 2007 was released, the most innovative introduction of a new Excel version was the release of Excel 5. The existence of a workbook (there was previously only one table sheet per file) has resulted in a real explosion of options and considerable improvement in work security. The principle of "a separate sheet for each type of content" has applied in the rS1.Method ever since. Content in this respect is a certain category of data, which is described in the following paragraphs. During the development phase of models, rS1 workbooks consist of at least six sheets with sheet names as shown in *Figure. 4*.

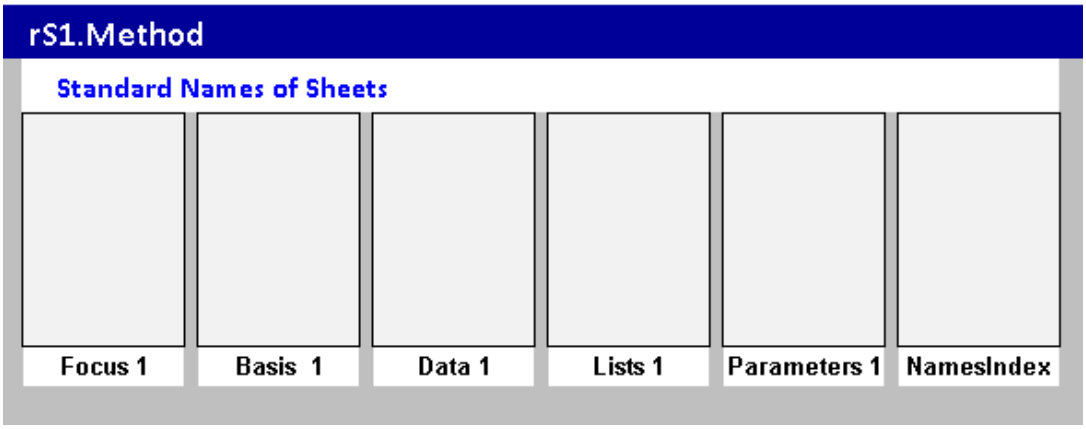


Fig 4: Standard Sheet Names of an rS1-Workbook

Sheet 1: *Focus 1*

Sheet 2: *Basis 1*

Sheet 3: *Data 1*

Sheet 4: *Lists 1*

Sheet 5: *Parameters 1*

Sheet 6: *NamesIndex*

The names of sheets 1 to 5 can be supplemented by meaningful suffixes, each separated by a space. Such a suffix may also consist of several character strings. Those sheet name combinations shown in Figure 5, for example, are also valid.

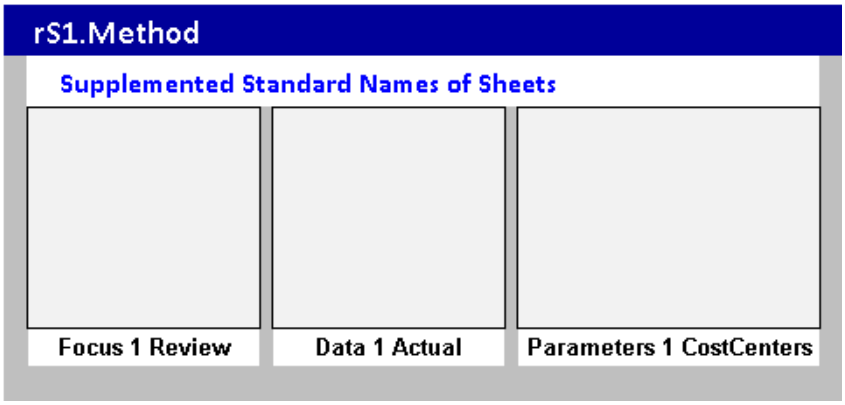


Fig. 5:
Supplements are allowed and helpful

The specified names always apply to the design phase of your model. The sheets can be subsequently renamed if other names prove to make more sense after you have completed the model and subjected it to a functional test. The significance of this name assignment and sub-division will become completely clear later when we explain the conventions for the range names. Unnecessary sheets are deleted once a development has been completed. Other sheets that are not to be viewed by the user are suppressed.

If several sheets of the same kind and type happen to be necessary, they are numbered consecutively; e.g., *Focus 2*, *Focus 3* etc. Naturally, this principle also applies to all other sheet names of the method.

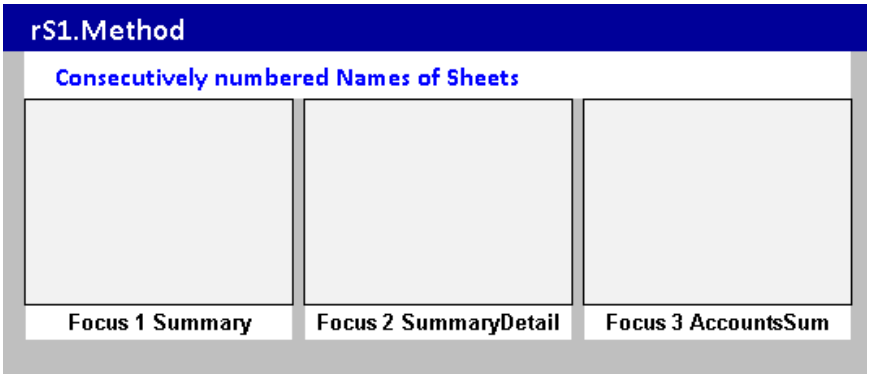


Fig. 6:
Sheets of the same basic type are numbered consecutively.

Focus 1 (display of selected data)

The *Focus 1* sheet (or other focus sheets following it) is the "face" of the application and incorporates all data directly viewed by the user or used for printing. It also contains charts and/or results structured in table form, such as periodic reports, and offers a dynamic, user-defined data selection in many cases.

Basis 1 (calculatory basis for the focus)

The *Basis 1* sheet contains the dynamizable data basis for the focus sheet. Data to be presented in the focus is compiled here with selective and extracting formulas. The formulas of this workbook often use variable arguments, which are generated by the user by means of controls (find more on controls in Section 3.4.4.)

Data 1 (Data container)

The *Data 1* sheet contains the source data of the solution; i.e., all data (either raw data or, better, as a structured sheet) of a certain category or group incorporated there by importing or other processes. If a model is to reference several source-data categories with its focus, several data sheets of the same kind are created according to the principle of "a separate sheet for each type of content" and are then assigned appropriate names with consecutive numbering.

Lists 1 (materials for controls)

The *Lists 1* sheet is a very essential element of the rS1.Method and usually contains all lists and other informations that serve the model as the basis for its dynamic elements. These are mainly definition ranges for controls. You can find out more about this later in the text.

Parameters 1 (master data and standards)

Data which plays a role as a constant or variable parameter in various parts of the model is stored on the *Parameters 1* sheet. This might include master data such as the designation and address of a company, names of responsible persons, and so on. This data is supplemented by standards such as percentages for dues and discounts or verified key data to be incorporated in a calculation of values. Finally, you can also store objects or define colors that will be used in another phase of the project.

NamesIndex (index of the range names)

Names play an extremely important role in the rS1.Method: They contribute the main structural element of the method. That is why it is particularly important always to have an overview of all the names used in the project. The *NamesIndex* sheet should not contain anything but a complete list of those range names (including their references) that currently exist in the workbook.

Other sheet names—options and limits

You certainly can set up even more sheets and name them according to the basic rules illustrated earlier. It is important that each letter of the alphabet is only used once as the first letter of a main term (such as *Focus, Graphics, Data, Basis*) in each workbook. A sheet with the name *Invoice 2* would be allowed only if there was no other *I* sheet. A sheet with the name *Chart 1* would not be allowed if *Center 1* exists. You can find the reason for this requirement in Section 3.2.

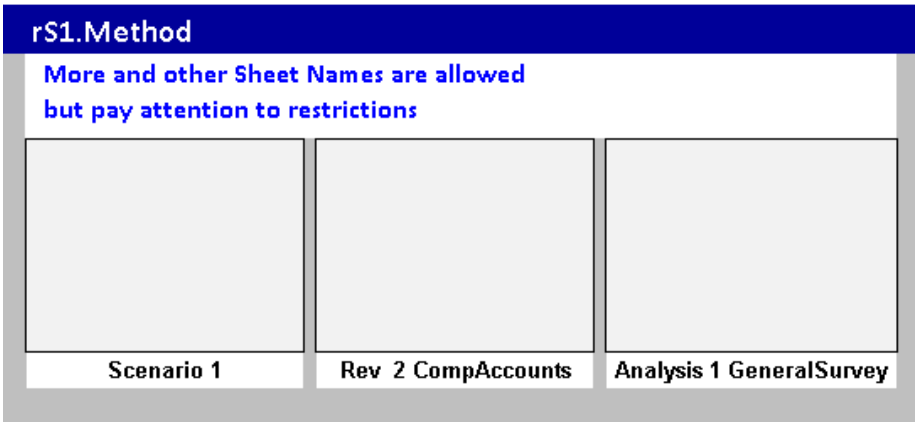


Fig. 7:
Other sheet names are also allowed
(under certain conditions)

3.2 Range Names and How to Use Them

Use of standardized range names plays a major role in the rS1.Method. Using such names will make it much easier to produce your Excel solutions, will improve the readability and interpretability of your application, and will therefore be a great help overall. Range names can be applied to everything that can be addressed as *range* in Excel; i.e., a group of related cells, a single cell, or a multiple selection of cells (a range that consists of several individual cells or cell groups).

rS1 range names consist of a prefix, a separator, and a suffix.

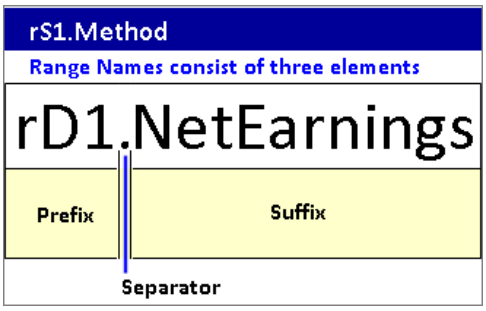


Fig. 8:
Syntax of range names

Prefix

The prefix is based on the name of the sheet that contains the named range. The prefix *rF1* confirms that this range is on the *Focus 1* sheet; the prefix *rD1* designates the localization of the name on the *Data 1* sheet. If the sheet is called *Parameters 1 Standards*, for example, all range names on this sheet begin with *rP1*. If a sheet is named *Statistics 2 LearningSuccess*, all of its range names begin with *rS2*. The leading lowercase *r* of the prefix indicates that it is a range, not a graphic object. Each sheet therefore has its own, sheet-specific prefixes for its range names. That is why the method requires that each letter of the alphabet should only be used once as first letter of a main term in the sheet name in each workbook. If the range name is assigned specifi-

cally to a certain sheet, I can tell by the prefix where to find this range, if necessary, in all name lists and in the program code. This is a huge advantage in many development steps.

Separator

A range name is separated by a point.

Suffix

The suffix, which is the actual name text, should designate the content of the named range as clearly and unambiguously as possible. If this meaning is to be expressed by two or more words or abbreviations (as in *rD2.CostsStaff*, *rP1.ProductNo* or *rB3.CountDuplex*), each new meaningful term begins with a capital letter but the letters are never separated by spaces or other characters. As an Excel rule, the range names must not contain any spaces or special characters.

Suffix extensions

The previously described procedure ensures that certain names can exist several times—but not ambiguously—in a workbook, since they can be distinguished clearly by their prefixes. But what if a range name is to be applied several times, not in a workbook but in a single sheet, which is not unusual with the rS1.Method? In this case, the suffix is supplemented by a consecutive number which usually has two digits. This number is not only a distinguishing feature; it is also a very useful sorting aid.

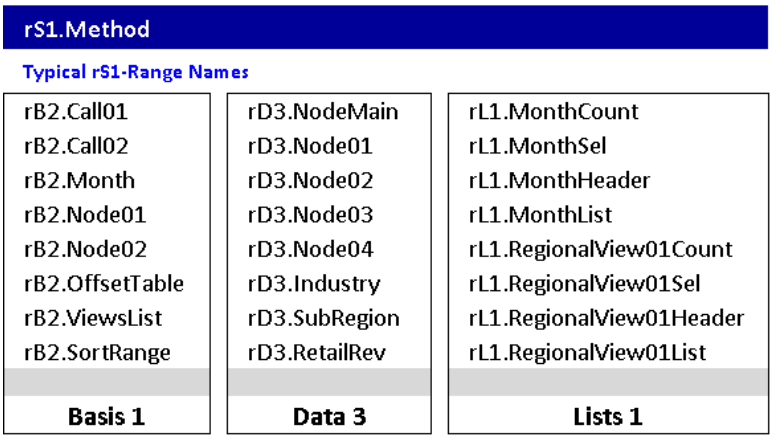


Fig. 9:
Typical rS1 range names

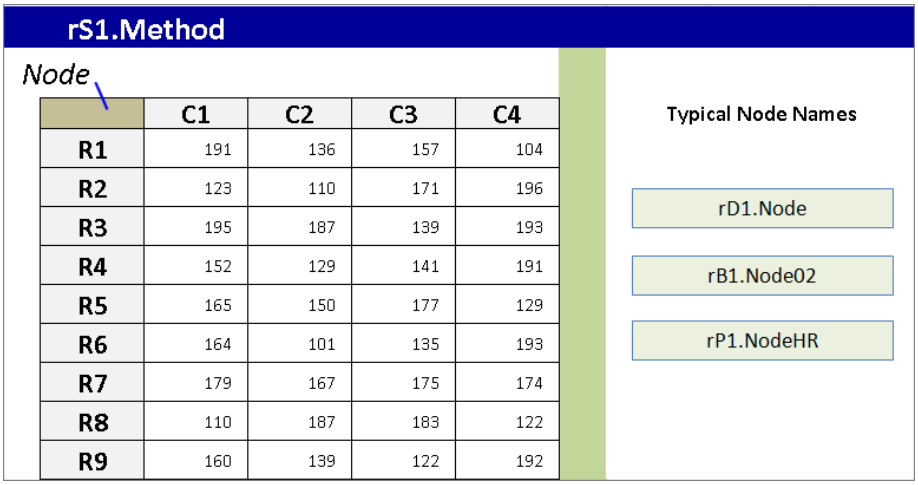
Special names 1: nodes

In rS1-models, names which exist several times include so-called *nodes*. From a functional point of view nodes are standardized starting or anchor points, from which you can control certain cells or cell ranges. They always refer to single cells, from which the `OFFSET` function, which is frequently used in the rS1.Method, can address or read out any other cell or cell range of any size on the sheet in any direction. The `reference` argument for the arachnoid access options of the `OFFSET` formula is the *node*.

The OFFSET function

`=OFFSET(reference,rows,cols,height,width)` returns a reference (or the value there), which is offset a certain number of rows and columns in relation to the specified reference argument. The `reference` argument therefore defines the starting point for access to a different cell or cell range. You can use this function not only to address other cells from a single cell in any direction, but also to grasp ranges of any `height` (= number of rows) and `width` (= number of columns). This lets you easily solve very difficult tasks on the basis of very simple structures.

In many sheets you do not have to assign any name other than that of the node. If the node exists, it does not matter to the calculation model whether the referenceable data range of a sheet subsequently becomes larger or smaller.



	C1	C2	C3	C4
R1	191	136	157	104
R2	123	110	171	196
R3	195	187	139	193
R4	152	129	141	191
R5	165	150	177	129
R6	164	101	135	193
R7	179	167	175	174
R8	110	187	183	122
R9	160	139	122	192

Fig. 10: Optimal position of the node and typical names of nodes

There should always be a node in the top left corner of structures in the table form; i.e., at the point of intersection of the existing or imaginary row and column headings.

Special names 2: selection cells

Cells in which the user makes an entry or to which the result of an input action is transferred are referred to as “selection cells”. The user makes a selection, which is often a number if the rS1.Method is applied consistently. He may enter the digit 5 directly in a cell, for example, but it is more likely that he will click on the fifth text entry of a *List Box* or a *Combo Box* (Form Controls), whereupon the digit 5 appears in the cell link.

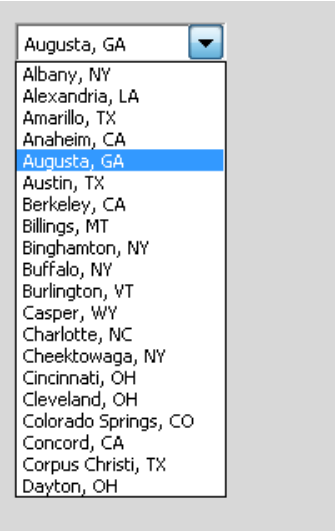


Fig. 11:
A click on the fifth list entry generates the number 5 in a linked cell

In each of these cases, the linked cell has the suffix extension *Sel*. The advantage is that you can tell at any time in appropriate formulas and in a program code whether a value to be further processed applies to a selection made by the user.

rL1.Account02Sel	rL2.DataTypeSel
rL1.AreaSel	rL2.MonthsSel
rL1.CommSectorSel	rL2.TerritorySel
rL1.Compare01Sel	rL2.GlobalParentSel
Lists 1	**Lists 2**

Fig. 12:
The abbreviation "Sel" always stands for a selection made by the user

Among the many advantages of these name conventions is the creation of a versatile system and an outstanding overview. In every Excel dialog box that manages names (*Name Manager*, *Paste Name* and *Go to*, see *Figure 13*) everything that belongs together appears together. On top of that, you can easily tell in most cases what the name stands for.

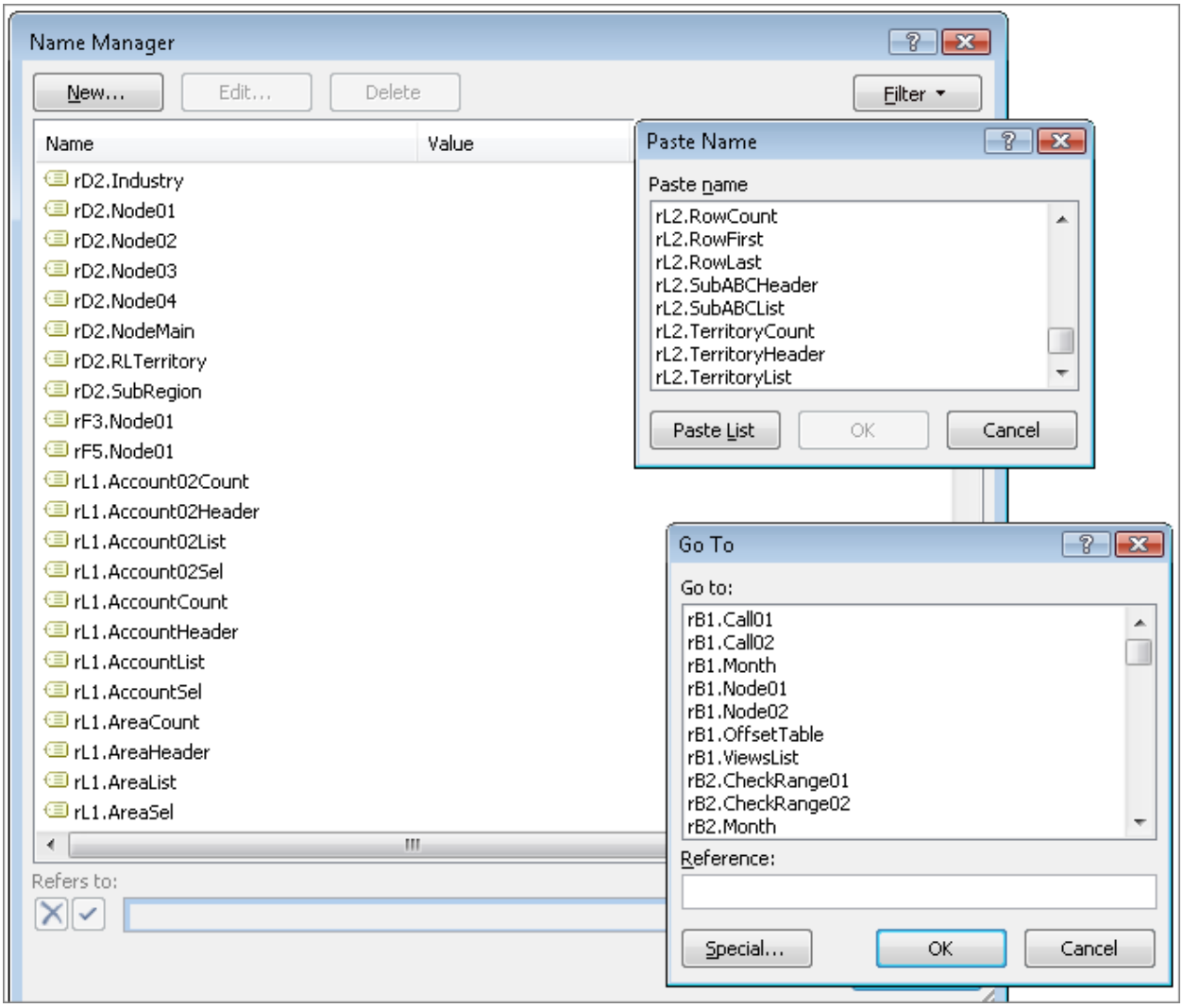


Fig. 13: Clarity and overview in name lists

3.3 Object Names and How to Use Them

Occasionally, controls and graphic elements are also given specific names. These are significant, however, only if such objects are to be drawn upon for a solution with program code in the event of subsequent supplements or extensions.

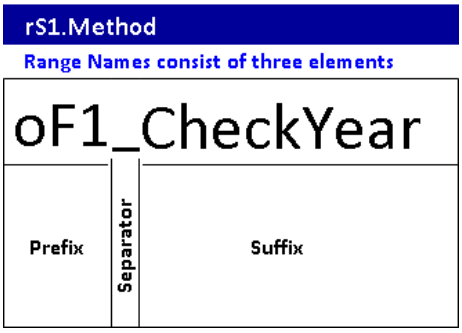


Fig. 14:
Syntax of object names
(Example: Check Box on the *Focus 1* sheet)

As with range names, the syntax of object names is: prefix—separator—name text. Examples of typical object names are:

- *oF1_OptWarning* (with the abbreviation "Opt" standing for an OptionButton)
- *oF2_BoxCc* (with the abbreviation "Cc" standing for cost centers)
- *oG3_SpinRegions* (with the abbreviation "Spin" standing for a SpinButton). See also *Figure 15* and the text pertaining to *Figure 15*.

Prefix

The prefix, in turn, is based on the name of the sheet that contains the object. The leading lower-case *o* of the prefix indicates that it is indeed a (graphic) object, not a range.

Separator

Object names are separated by an underscore.

Don't use a point as separator in object names:

You must not use points as separators in object names, as this will cause certain mistakes in the program if Excel programming is to be subsequently performed!

Name text

The actual name text should express the type, meaning, or content of the object.

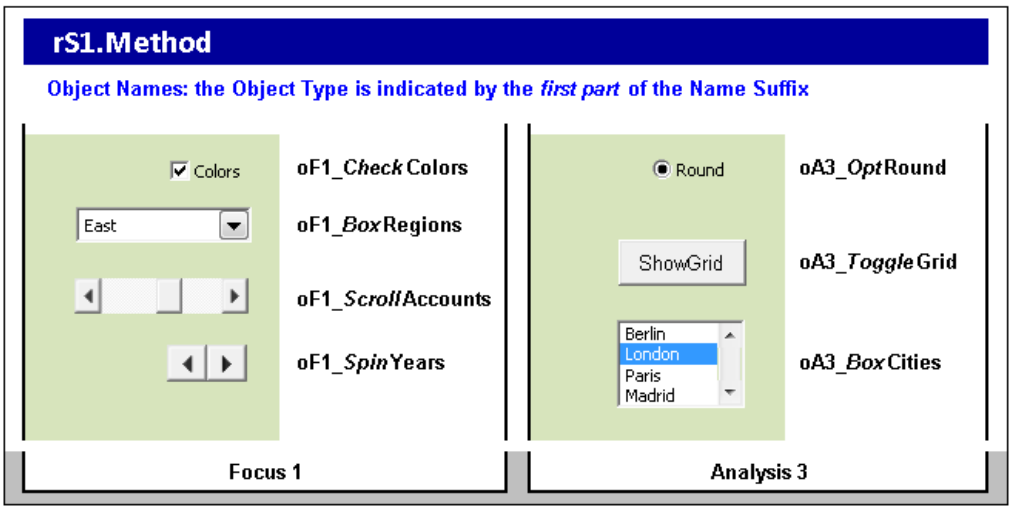


Fig. 15: The type of object is also indicated by the object name

The type of objects is designated in the first part of the suffix. *Figure 15* contains seven controls with their exemplary object names to the right of them (more on controls in Section 3.4.4.). The first part of the name suffix refers to the *type* of object:

Left, from top to bottom:

- *Check Box*, the suffix begins with *Check*
- *Combo Box*, the suffix begins with *Box*
- *Scroll Bar*, the suffix begins with *Scroll*
- *Spin Button*, the suffix begins with *Spin*

Right, from top to bottom:

- *Option Button*, the suffix begins with *Opt*
- *Toggle Button*, the suffix begins with *Toggle*
- *List Box*, the suffix begins with *Box*

Combo Box* and *List Box

Because these two elements are managed and programmed in an almost identical manner, no distinction has to be made for their object names. Both suffixes therefore begin with *Box*.

3.4 Particularities, Recommendations and Notes

You can find out a number of certain details and particularities of the rS1.Method in this section, which continues our discussion of "sheets". You also can find information about useful procedures.

3.4.1 K11 as top-left cell/auxiliary rows and auxiliary columns

All sheets of rS1 standard workbooks have auxiliary rows and columns. These are ranges that receive functional, supporting structures during and also after the development phase. You'll find more information on this later. The height of the auxiliary rows 1 to 10 is *8*. The width of the auxiliary columns A to J is *1*. As a result of this structure, the top left cell of the actual function range of a table is cell *K11*.

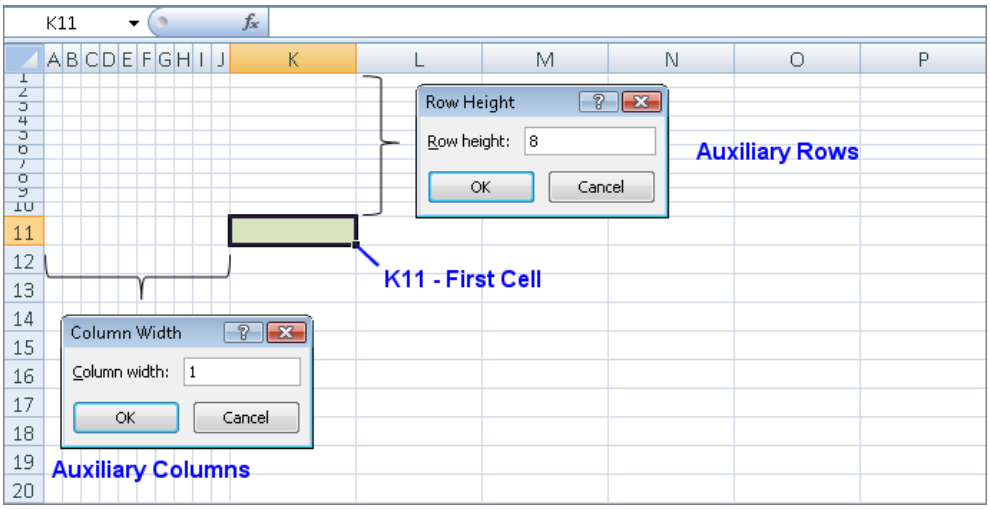


Fig. 16: Auxiliary rows and auxiliary columns / K11 is the top left cell

I have already made the important point that—especially in large calculation models—most formulas obtain the values of their arguments from the outside. The "outside" is, for example, also the auxiliary columns or auxiliary rows. The formulas of the calculation model are below and to the right of cell *K11*. Values required by the formulas in order to change their variable arguments are above and to the left of *K11*. The auxiliary rows and auxiliary columns are no longer visible at the end of the development. They are, if not in hidden sheets, suppressed, made invisible with identical letter and background colors, and occasionally also covered with other structures; e.g., with text boxes. In *Figure 17* you can see an exemplary screenshot from a Basis 1 sheet, showing formulas and some traces to their auxiliary cells.

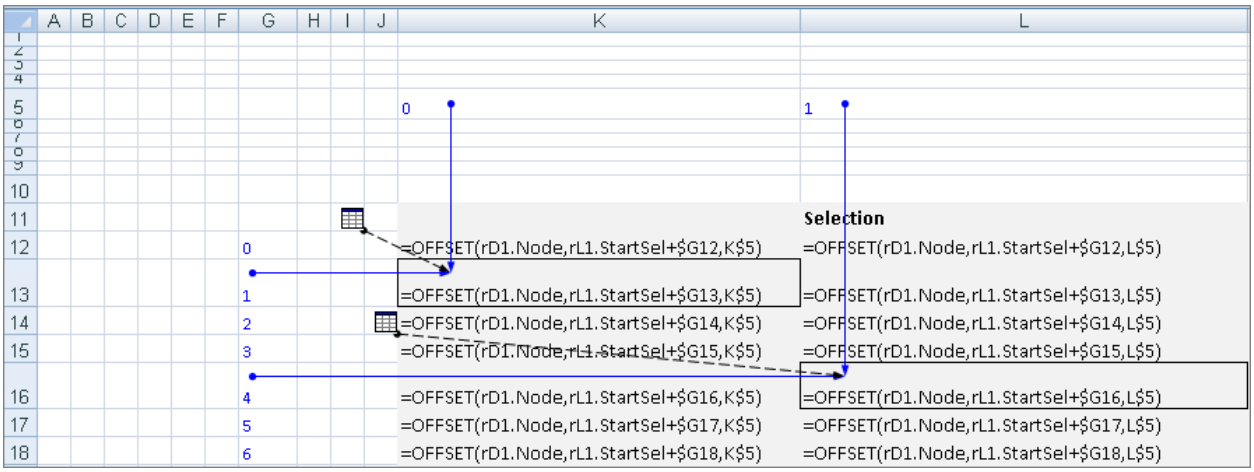


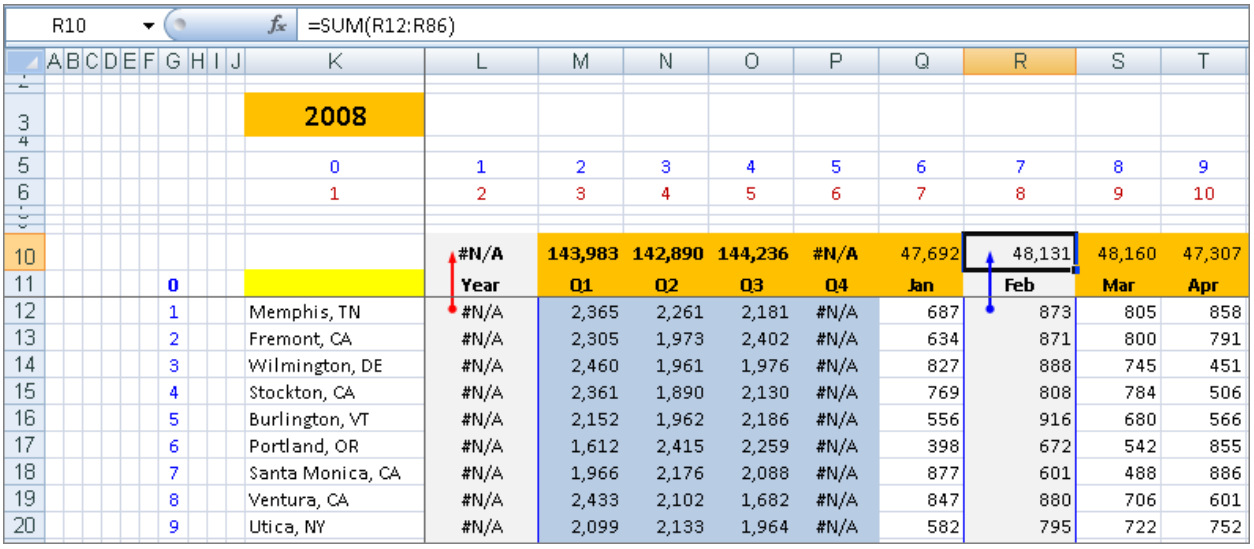
Fig. 17: Formulas using auxiliary ranges

Very few solutions will actually need *all* auxiliary rows and auxiliary columns. In many models, you will not need any at all. If I only need one at a time, I often use row 5 and column G. In most cases, I do not get any closer to the function range, as I often insert headings and other information ranges after the subsequent tests and use the columns from J or rows from 10 upwards for this purpose. This is no rule, it is a habit. Habits are often quite tedious, but can occasionally also

be quite useful. This one is useful because it allows me to work without much thinking (row 5 and column G are always free and are not used for anything else) and because I often can find what I am searching for later on (if there are no auxiliary structures for this routine type of usage in row 5 and/or in column G, this means there are none at all).

3.4.2 Summing up at the top

Some people may need to get used to finding a total above a value range instead of below it. Nevertheless, there is much in favor of such a design because tables can be structured variably in different levels, either entirely or partially. People from our culture usually scan images—and Excel tables are basically nothing else—with their eyes from left to right and from top to bottom. If you take business reports as examples, the following perception aspect applies: Regardless of the state of the report's structure, you should find information on the way from the top to the bottom of a hierarchic structure that corresponds to our perceptual requirements. The most important thing is first (top left), less important things below it (and possibly offset to the right), even less important things below that, and finally the most unimportant thing at the very bottom. By *no* means, therefore, should the total be at the bottom.



	G	K	L	M	N	O	P	Q	R	S	T
5		0	1	2	3	4	5	6	7	8	9
6		1	2	3	4	5	6	7	8	9	10
10			#N/A	143,983	142,890	144,236	#N/A	47,692	48,131	48,160	47,307
11	0		Year	Q1	Q2	Q3	Q4	Jan	Feb	Mar	Apr
12	1	Memphis, TN	#N/A	2,365	2,261	2,181	#N/A	687	873	805	858
13	2	Fremont, CA	#N/A	2,305	1,973	2,402	#N/A	634	871	800	791
14	3	Wilmington, DE	#N/A	2,460	1,961	1,976	#N/A	827	888	745	451
15	4	Stockton, CA	#N/A	2,361	1,890	2,130	#N/A	769	808	784	506
16	5	Burlington, VT	#N/A	2,152	1,962	2,186	#N/A	556	916	680	566
17	6	Portland, OR	#N/A	1,612	2,415	2,259	#N/A	398	672	542	855
18	7	Santa Monica, CA	#N/A	1,966	2,176	2,088	#N/A	877	601	488	886
19	8	Ventura, CA	#N/A	2,433	2,102	1,682	#N/A	847	880	706	601
20	9	Utica, NY	#N/A	2,099	2,133	1,964	#N/A	582	795	722	752

Fig. 18: Totals are displayed at the top

3.4.3 Calculation mode

For a number of actions, Excel recalculates the entire workbook, even if this does not (initially) appear to be logical. Each recalculation takes time; even a few seconds in the event of models with thousands of formulas. Nevertheless, this automatism should not be suppressed, as most rS1 calculation models require constant (i.e., automatic) calculation because of the cross-linking of its formulas, which is sometimes quite extensive.

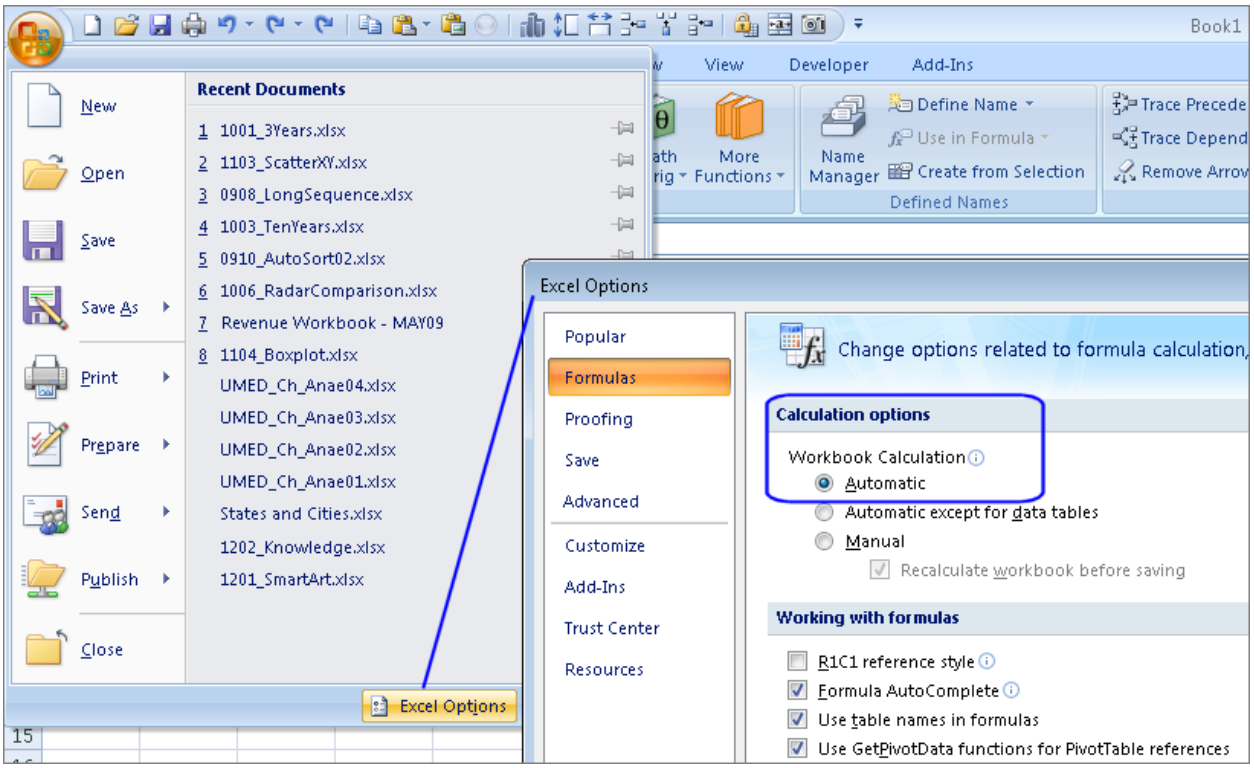


Fig. 19: In most cases automatic calculation is necessary

You can make the definition, as indicated in *Figure 19*, with the commands *Office Button/Excel Options/Formulas* tab/*Calculation Options*/*Automatic* section. You should then test your model several times to see whether the computation times are acceptable. Even 5,000 or more formulas in complicated structures pose no real time problem for modern computers with quick processors, sufficient working memory, and high-capacity graphics cards. If you do decide that your solution is too slow due to the constant recalculations, two things can help. First, ditch any unnecessary data. However, because you have followed every trick in the book and, on top of that, have worked according to the rS1.Method, you will find barely any unnecessary data. The second option involves programming: You can switch off the automatic calculation and define by program code what is to be calculated and when.

3.4.4 Controls

If you work according to the rules of the rS1.Method, the controls available in Excel will play a major role in the dynamization of your solutions. The principle used is quite simple. You make a control selection with a mouse click and, in this way, create an output value which you transfer to a cell of your choice. Examples of such output values are:

- Variable numbers that you can use directly or indirectly as arguments of a formula *or*
- Logical values (`TRUE` or `FALSE`) that you can process directly or indirectly, for example with an `IF` formula, *or*
- Text which you can use directly or indirectly, for example as the `search criteria` argument of a formula (such as in `VLOOKUP` or `SUMIF`).

You already learned about these excellent tools earlier in connection with object names (*Figure 16*). Here are further details.

Access to controls

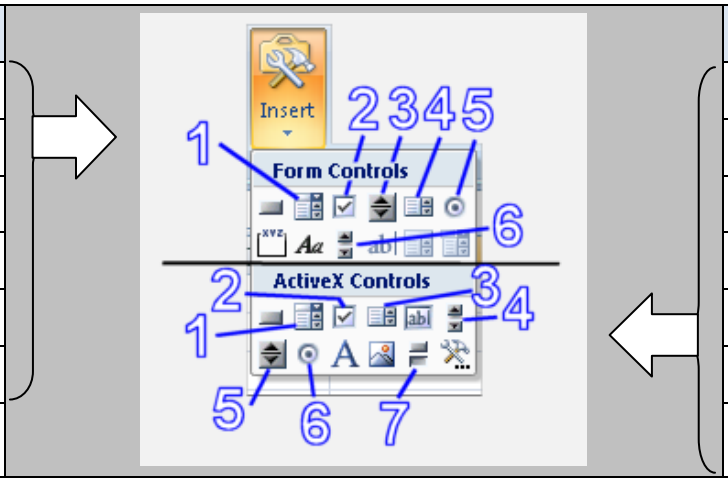
The commands for setting up and designing controls can be found in the ribbon/*Developer* tab/*Controls* group.

The *Developer* tab is not automatically displayed after the installation of Excel 2007. It can be made permanently available, however, with the following commands: Office Button/*Excel Options*/*Popular* tab/*Show ... Developer tab*.

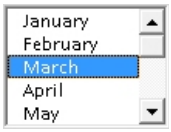
In general, you can use two different types of controls to "vitalize" your applications. Both of them are used in the rS1.Method:

- Use plain and barely formattable *form controls* for simple tasks in solutions that do not need complex and formally sophisticated design. These elements are relatively undemanding in any respect, and few if any problems are expected during their functionalization and use.
- Use the extensively designable *ActiveX controls* in those solutions that place high demands on appearance and therefore also require various formatting options from controls. Working with these elements is relatively demanding and may result in minor display errors in some system environments under certain conditions. In so-called "non-program applications" however, they provide widespread options of design and can be enhanced greatly by program code (*ActiveX controls* were created primarily for use in programmed models. Their use for dynamization of non-programmed solutions belongs to the characteristics of the rS1.Method).

Only those controls used in non-programmed solutions created with the rS1.Method are listed in the following table.

Form controls		*ActiveX controls*
1 Combo Box		1 Combo Box
2 Check Box		2 Check Box
3 Spin Button		3 List Box
4 List Box		4 Scroll Bar
5 Option Button		5 Spin Button
6 Scroll Bar		6 Option Button
		7 Toggle Button

Naturally, it is up to you to decide which control to use and for what purpose. You should observe certain basic rules, however, and take into account the type of design of these tools. Here are a few notes:

Control		*Actions*
	Combo Box	Select an element from a list of elements. It is a text field with dropdown list. Manual input is optionally possible as selection specification.
	Check Box	Switch an option on or off by activating or deactivating it.
	List Box	Select an element from a list of elements (all elements are visible).
	Scroll Bar	Increase or reduce a value in single steps by clicking on the arrows or scroll quickly through complex value ranges with the mouse button pressed down.
	Spin Button	Increase or reduce a value in single steps by clicking the arrows.
	Option Button	Select an option from a group of options.
	Toggle Button	Switch an option on or off.

Further information

You can learn much more about setting up and working with controls in the book for which this document is a supplement.

3.4.5 Names on the Lists 1 Sheet

The preceding short description of the controls will help make the details in the following paragraphs easier to understand. Depending on their significance, the controls have their own definition ranges in each workbook designed according to the rS1.Method. It's best to create these ranges on the *Lists 1* sheet. The following conventions apply.

Fig. 20:
Controls and their object names

As explained in the section on *object names* (Section 3.3), specific rules apply to the designation of controls. You can see three of the elements in *Figure 20* together with the corresponding names. We can assume that the controls were created on a sheet with the name *Focus 1*, which is the reason for the name prefix *oF1*.

- *A*: The user clicks through several years with a *SpinButton* named *oF1_SpinYear*. As a result, budget data of different financial years—or similar data—can be addressed in the background and read into the focus. The output value of this control is changed step by step by clicking on the arrows. The output value is a number that is transferred to a definable cell. This cell is referred to as a *cell link* when using the *form controls*, whereas a so-called *LinkedCell* is used for the *ActiveX controls*.
- *B*: The *oF1_CheckColors CheckBox* is used by the user to decide, for example, whether its data is to be displayed in the focus with traffic light colors in order to indicate certain problems. The output value of this control is changed by clicking on the box. The output value is a logical value (i.e., `TRUE` or `FALSE`) that is transferred to the *cell link* or to the *LinkedCell*.
- *C*: The *oF1_BoxCity ListBox* is used by the user to make a selection from a list of different towns. As a result, regional sales revenues of a product, for example, could be addressed in the background and read into the focus. The output value of this control is changed by clicking on an element of the list. The output value is (provided nothing else is determined when using ActiveX controls) a number that is transferred to the *cell link* or to the *LinkedCell*.

The user therefore makes a decision and *selects* an item by means of a mouse click. The result of his selection is transferred to the *cell link* or *LinkedCell* and then used by formulas. As described above, such cells should always have the suffix extension *Sel* (for *selection*) and normally can be found on the *Lists 1* sheet.

The *cell links* or *LinkedCells* can be found in column L of the *Lists 1* sheet (or, if necessary, in *Lists 2*, and so on) for all controls other than the *ListBox* or *ComboBox*. In most cases, the name of the control to which this *cell link* or *LinkedCell* belongs is displayed above the cell for information. The following syntax rule applies to the designation of the cell itself:

- Prefix from sheet name; i.e., usually *rL1*, followed by a point as separator
- Suffix from designation of the related control, followed by the character string *Sel* (see also the information texts in *Figure 21*, column K)

This sounds more complicated than it actually is. A *LinkedCell* with the name *rL1.SpinYearSel* belongs to the object *oF1_SpinYear*. The selection of the *oF1_CheckColors* control is transferred to a *LinkedCell* with the name *rL1.CheckColorsSel*. And a *ToggleButton* with the name *oF1_ToggleMode* would have to be connected to a *LinkedCell* with the name *rL1.ToggleModeSel*. You will soon understand the logic of these name assignments during your daily work and will then be able to benefit from their advantages in many situations, especially when designing—or *re*designing—your solution.

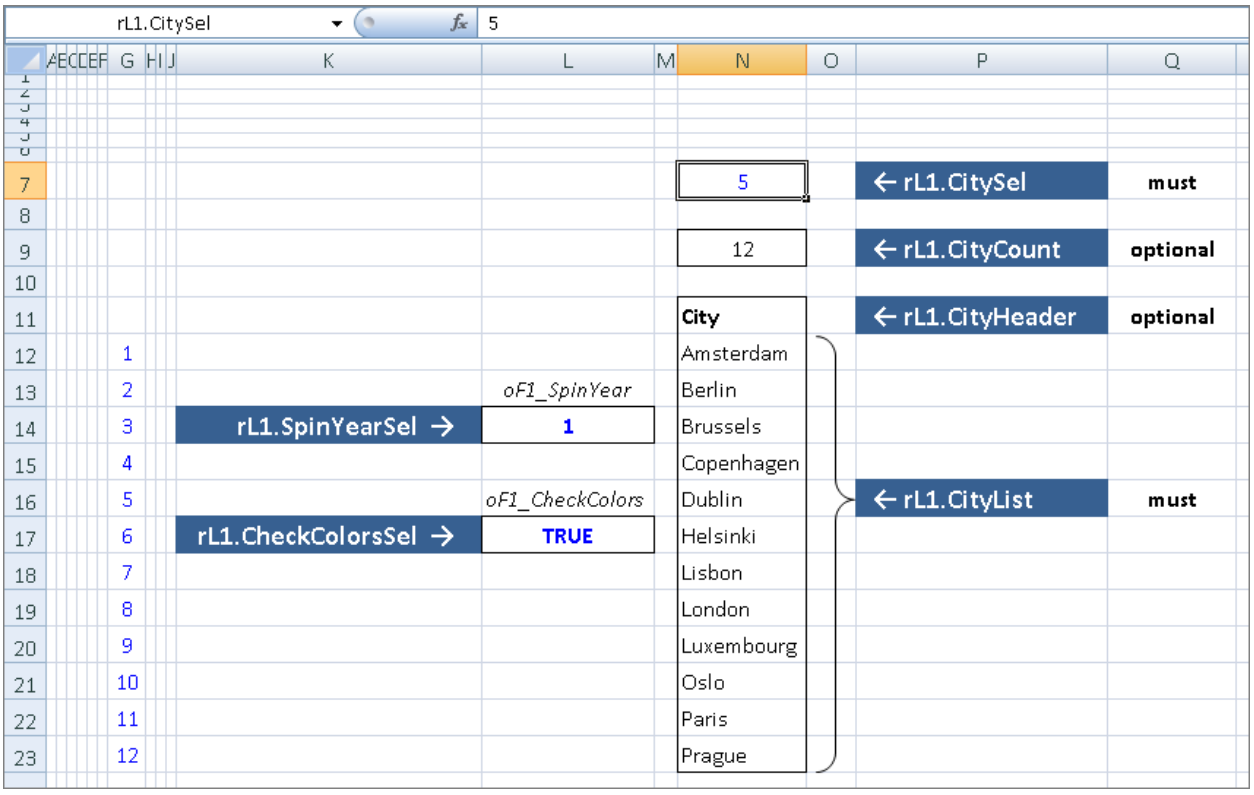


Fig. 21: rS1 range names on the *Lists 1* sheet

Let's now look at the list definition ranges on the right side of *Figure 21*. Such list definitions are required for the *ComboBox* and *ListBox* controls.

- There are single-column and multi-column list definition ranges. The figure shows the single-column standard. The title of the list should be as short as possible and is typed into row 11 of the respective column without spaces or special characters. This cell forms the list header and (optionally) contains the standard prefix as range name (i.e., *rL1*), followed by the standard separator, followed by the text in the cell, followed by the *Header* suffix extension. The cell *N11* with the text *City* is therefore given the name *rL1.CityHeader*. If such a list header cell contained the text *Months02*, for example, it would be named *rL1.Months02Header*.
  This name assignment for the list header is OPTIONAL. As a rule, the name is mostly not necessary in non-program applications, but it may be useful in programmed solutions.

- The actual content of the list, the *input range* (as it is called with the *form controls*) or the *ListFillRange* (as it is less ambiguously called when using *ActiveX controls*) follows in a row of lines without interruption below the header. This range receives the text content of the list header as name text behind the prefix and separator, followed by the *List* character string. In the illustrated example, this definition range is therefore quite logically called *rL1.CityList*. If the list header cell contained *Months02*, the *ListFillRange* would be named *rL1.Months02List*. The name assignment for a *ListFill-Range* is MANDATORY if the rS1.Method is applied. The name is used in non-program applications as well as in programmed solutions.

- In row 9 of the respective column, a formula is used to count the number of elements in the list; i.e., the number of entries in the *ListFillRange*. The `COUNTA` function is used for this purpose. The corresponding formula in the illustrated example is `=COUNTA(rL1.CityList)`. Cell *N9* receives the text content of the list header as name text, followed by the *Count* character string; i.e., *rL1.CityCount*. This name assignment is OPTIONAL. You won't need the name in non-program applications, but it may be useful in programmed solutions.

The COUNTA function

The `=COUNTA(reference)` function counts the amount of data (cell contents) in the `reference`, regardless of the type of content. Any text strings and numbers are counted. However, the related `COUNT` function only counts the amount of *numbers* only within the reference.

- Finally, row 7 of the respective column contains the *cell link* or *LinkedCell* of the list. A number corresponding to the user's selection appears each time the user clicks on an entry in the *ComboBox* or *ListBox*. Cell *N7* receives the text content of the list header as name text, followed by the character string *Sel*: *rL1.CitySel* in the example. The name assignment for this cell is MANDATORY if the rS1.Method is applied.

Recommendations for the heading text in the list header

The text entry in the list header is the main element for all designations and further procedures, whether they are performed manually or automatically. The entry should be short. Use abbreviations, however, only if they are clearly understandable. The text should still designate the list concisely and not contain any spaces or special characters. In the book you will see from a few examples that it may be necessary to keep two lists of identical content. This might happen, for instance, if data from different months has to be placed side by side in comparison reports by clicking the mouse. For this purpose, there need to be two boxes with the name of the month and also two definition ranges for the respective lists. In such cases, a number is appended to the texts of the list header. The required distinctions can be made with *Months01*, *Months02*, etc.

3.4.6 Defining names

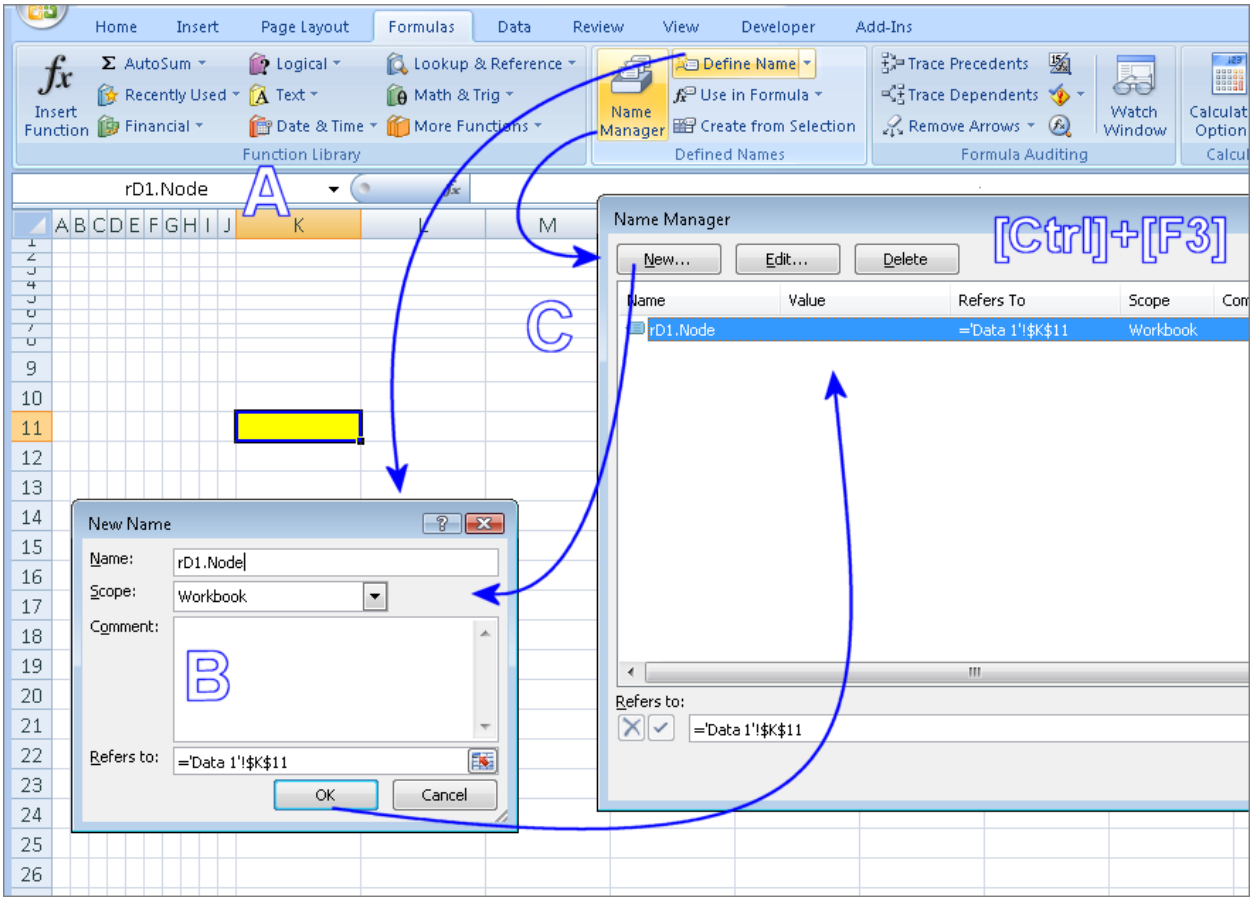


Fig. 22: Defining a name for a cell or cell range

We've discussed name conventions at length already. In addition, here are a few notes on the name-definition methods used in the example in *Figure 22*.

Cell *K11* is to be given the name *rD1.Node*. Mark the cell and then proceed as follows.

- Variant *A*: Enter the name in the *name box* on the left side of the formula bar and confirm it by pressing the [Enter] key.
- Variant *B*: Activate the *Formulas* tab in the ribbon and click the *Defined Names* group on the *Define Name* command button. Then enter the name in the *New name* dialog box. You can use this dialog box to make further decisions on the management of this name.
- Variant *C*: Activate the *Formulas* tab in the ribbon and click the *Defined Names* group on the *Name Manager* command button. Alternatively, you can use the key combination [Ctrl]+[F3].
  Then click on the *New* button in the *Name Manager* dialog and continue as with variant B.
  You can also make complex decisions with regard to the management of the names of your workbooks in the *Name Manager* dialog.

For flexibility and work security, variants *B* and *C* should be preferred over variant *A* when defining range names.

When defining object names (for the designation of controls for example), use variant *A* after marking the object (entering the name in the name box and then pressing the [Enter] key)

3.4.7 Inserting names in formulas

The rS1.Method requires the consistent use of range names as formula arguments. It is very easy and convenient to insert such names in formulas, as shown in *Figure 23*:

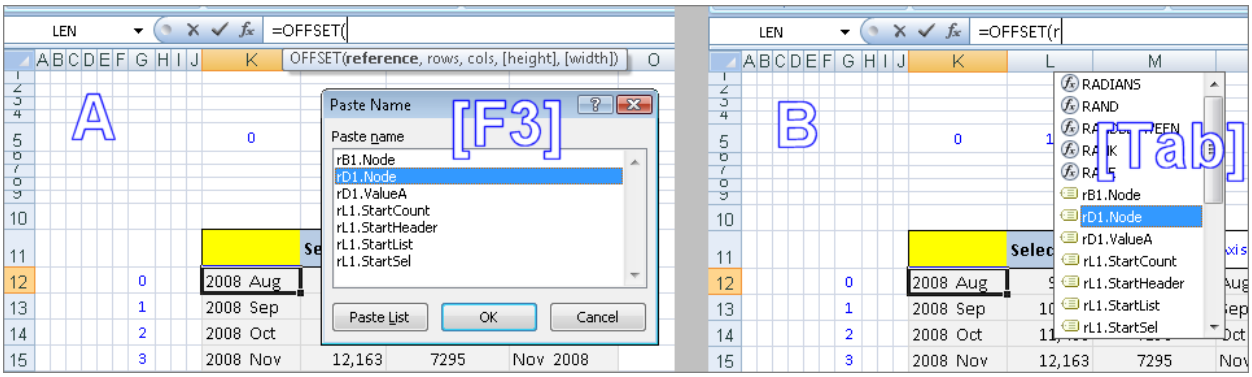


Fig. 23: Inserting names in formulas

- Variant *A*: Write the formula up to the point where a range name is to be inserted. Then press the [F3] key. A dialog appears with a list of all assigned names. Because a very clear overview is established by applying the rS1.Method with its name prefixes, it is easy to find the required name and insert it in the formula by double-clicking on it or selecting it and then selecting *OK*.

- Variant *B*: Do you know the first letter of the required name by heart? If so, you can ensure consistency when applying the rS1.Method! Write the formula up to the point where a range name is to be inserted. Write the first letters of the name, such as *rd*. A selection list (via the *Formula AutoComplete* feature) including all the names which begin with *rD* now appears below the formula bar (if you, as shown in *Figure 23*, had only entered an *r*, all formulas *and* all names which begin with *R* would have been listed). Double-click on the name to be inserted or mark this name in the list and then press the [Tab] key.

The **Formula AutoComplete** feature is made available with the following commands: Office Button/*Excel Options*/*Formulas* tab/*Working with formulas* section/*Formula AutoComplete* option.

3.4.8 Moving to named ranges

Range names are also very useful navigation aids. Open the dropdown list of the names defined in the workbook in the *name box* to the left of the *formula bar*. The named range is moved to and marked by clicking on the desired name. You switch automatically to the respective sheet because the names, if not defined otherwise, apply to the entire workbook.

Another navigation method is the *Go to* dialog, which you can open with [Ctrl]+[G] or display even faster with the [F5] key.

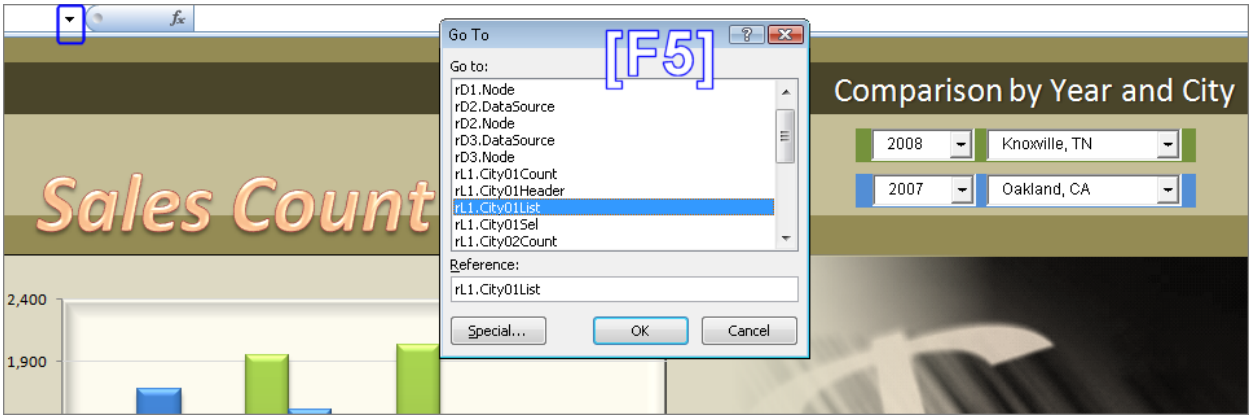


Fig. 24: Moving to ranges with *Go to* ([F5] key)

You also can find all range names in the list of this dialog and move to a range quickly by clicking *OK* or by double-clicking on an item.

3.5 Excel Templates

A standardized procedure, as required by the rS1.Method, is more efficient if you can draw on workbooks already prepared accordingly (see Section 3.1.2). The user-defined, configurable *Excel template* is a specific Excel storage form that corresponds to the *document template* in Word in its form and in the way it is used. Make sure you maintain creative freedom when using such templates. Define the "main details", but avoid any specific structural specifications. If you specify too many details, you are more likely to restrict your work than make it easier, unless you create a very special template for a very specific recurring purpose. Your Excel template should normally contain only a few or no function elements, but should contain all basic structures that facilitate development using the rS1.Method (named sheets, column widths and row heights of the auxiliary columns and auxiliary rows, standard formatting on the *Lists 1* sheet). The *Excel template* is therefore a workbook which can be reproduced at any time without much effort, freeing you from having to create the structure of a standardized basic frame time and again.

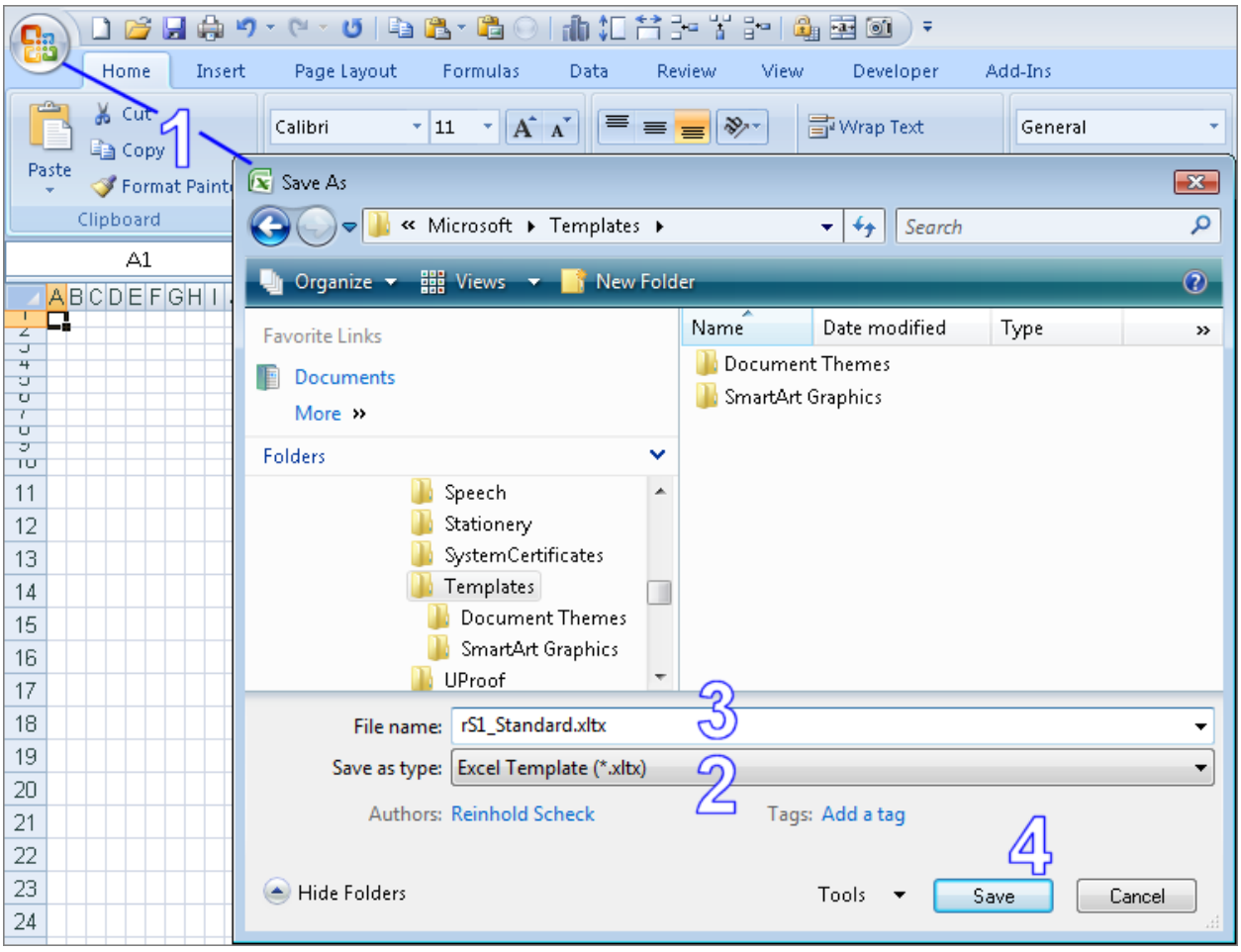


Fig. 25: Saving a file as template

This is how to create an *Excel template*, as illustrated in *Figure 25*:

1. The standard Excel file, already prepared, (*xlsx* file type) is open. Select the *Save As* command after clicking on the Office button.
2. Set the *Excel template* as a file type in the dialog.
3. Enter the desired name as *file name*.
4. The file is stored as a template (*xltx* file type), which can be reproduced after clicking on *Save*.

If you now wish to begin a project for which you require this *Excel template*, follow these steps:

1. Select the *New* command after clicking on the Office button.
2. Select the *My templates* command in the *New Workbook* dialog.
3. The *New* dialog then appears with the *My templates* tab containing a list of available user-defined *Excel templates*. Select the template and then click on *OK*. A copy of the template—not the template itself—is opened.
4. If you now wish to save this copy of the template, the default values are automatically reset by Excel as a storage variant for the type. As a matter of routine, save the copy of the *xltx* template as an *xlsx* file in a "normal" workbook. This procedure protects the saved template from accidental changes and damage.