https://www.citadel5.com

GS-Base 22.5

    3. Creating pivot table reports

10. Finding file duplicates, photo/MP3/MP4 duplicates, listing files and their history of changes
    1. Verifying file (path) lists in tables
    2. Mass-renaming, copying and deleting files
    3. Searching for photo/image/media file duplicates in folders and on disks
11. Displaying two tables from the same database and setting up a relation between them
    1. Using dual views
    2. Linking tables in dual views by the 1-N relation
12. Joining, splitting and merging tables and records
    1. Joining and splitting (normalizing) tables
    2. Merging and unmerging records
13. Spell-checking
    1. Spell-checking and adding Hunspell dictionaries
14. Formatting: styles, hyperlinks, background images and in-cell charts
    1. Formatting - general styles
    2. Formatting - hyperlinks
    3. Formatting - images
    4. Formatting - charts
15. Searching, filtering, sorting
    1. Searching / filtering records
    2. Searching / saving current filters as named search keys
    3. Searching - duplicates
    4. Searching - unique field values
    5. Searching - quartiles, mean values
    6. Searching - random records
    7. Searching - selected records

8. Full text searches and replacing
9. Find-as-you-type and find-similar searching options
10. Sorting

16. Sending emails and verifying URLs
    1. Sending email messages
    2. Verifying URLs entered in database fields
17. Printing tables, forms, letters and labels, mail-merge
    1. Printing tables, forms, letters and labels
18. File-level password protection and encryption
    1. Password protection and encryption
19. Loading, editing and saving files in various file formats
    1. Opening and saving text files
    2. Saving to PDF
    3. Opening and saving MySQL *.sql files
    4. Opening and saving SQLite *.db files
    5. Opening and saving HTML files
    6. Opening and saving dBase/FoxPro/Clipper files
    7. Opening and saving Excel workbooks
20. Settings and recently used database lists
    1. Modifying program settings
    2. Managing the list of recently-used files
21. Using scripts within GS-Base and COM programming with external tools
    1. JScript / VBScript scripting: samples
    2. C++ COM programming: samples
    3. COM programming: methods and properties
22. Contacting technical support

GS-Base is a high-performance desktop database adaptable to a wide range of data-management tasks — from photo and multimedia albums, disk archives, and inventory lists to large, complex databases linking multiple tables. You can use simple filtering as well as advanced methods such as regex, formula-based, and predefined filters to load and search tables containing up to 256 million records.
GS-Base provides super-fast native editing of CSV, text, Excel, xBase, SQLite, MySQL files, 4GB and above, opening and saving them directly without format conversion, while keeping all data in RAM.

- Up to 256 million records in one table; 16,384 fields in one record.
- Fast filtering of CSV and text files of virtually any size, independent of available RAM.
- Any number of tables in one file; tables in nested subfolders.
- Text, Numeric, Long Text/Memo fields, Image/File fields; Code fields for code snippets with syntax highlighting for 16 programming languages.
- Pivot table data functions in GS-Base: up to 256 million rows and 16,384 columns; 25 data field functions (vs. 11 in Excel), full regex filtering.
- Several record filtering methods, searching for duplicates, for unique values and their frequencies, search-as-you-type, random and quartile searches, full-text searches, fuzzy searches and one-click statistical breakdown analysis. RegEx filtering of millions of records in seconds, regardless of the number of records returned.
- Searching for file duplicates and finding similar photos, images, music, and video files using EXIF and multimedia tags, with the ability to play filtered lists of MP3 files.
- Around 300 built-in calculation functions used for calculated fields, data validation and automatic conversions of the entered text. Formulas to obtain

aggregated field values and statistics from other tables. Extracting and processing EXIF tags from photos.

- Python integration. Create your own data processing functions in Python to automatically update calculated fields, e.g. to load CSV files into fields, or mass-create any charts/graphics to be displayed in GS-Base fields.

- Any number of fields/records with mini-charts (column, line and area charts).

- Instant calculation of running totals and moving averages with weights in columns.

- Definable number (1 to 100) of processor cores used for searching, field and pivot table calculations.

- Drop-down/input lists dynamically selected based on the data from other fields.

- Efficient memory usage to quickly load, edit and save large (greater than 4GB) GS-Base databases and files in other formats. → Examples.

- Loading listings of all files from folders and disks along with their metadata. Adding any type of your own metadata to any file (which can be text of any size, images, the older versions of the file itself, other files etc.).

- Monitoring file changes with a searchable history of modifications.

- Mass-renaming, -copying and -deleting files based on filtered, auto-generated listings.

- Performing all types of table joins and the opposite functions: easy automatic normalization (splitting) of tables with a single click.

- Very fast data consolidation - you don't have to bother with permanent indices; internal indices are created automatically whenever aggregation and binary lookup functions need them.

- Linking individual records by any number of hyperlinks.
- Easy database synchronization between database copies: (un-)merging records from other users' databases, batch database/record updates based on other databases/tables.
- Convenient zip/zip64 native self-describing file format enabling users to view and modify records, tables and embedded files/images even without using GS-Base.
- Using multi-GB CSV/text, dBase/Clipper/FoxPro, MySQL (*.sql), SQLite *.db, HTML, PDF files.
- Native file editing in many file formats, for example, processing CSV files without any undesirable conversions or hidden formatting.
- Loading/editing/saving Excel *.xlsx and *.xls workbooks; databases containing tables with records exceeding the *.xlsx and *.xls row limits are automatically split and saved (or loaded and merged) either as multiple *.xlsx/*.xls workbooks or as multiple worksheets in one workbook.
- A password generator.
- Strong (Twofish CFB) standard password protection and encryption. AutoSave and AutoClose.
- Encrypting either entire files or just individual fields in your tables, and enabling other users to browse/edit the rest.
- Using SHA-256 checksums to verify whether the data remains unmodified, for example when switching between file formats.
- Easy-to-use JScript and VBScript scripting extending GS-Base functionality: automatic merging of records or tables from folders with Excel, text and other files.

- Mass adding, updating, merging and joining tables from folders with several million records can take, for example, just a couple of tens of seconds. Direct COM programming.
- Printing serial forms, letters/reports and any type of mailing labels.
- Sending serial personalized email messages with customized attachments.
- Verifying web links/URLs.
- Multiple panes displaying synchronized tables, forms, memo fields and pivot tables.
- System or user-defined spell checkers for selected fields.
- GS-Base can be installed on any portable storage device and used without performing any registry modifications.
- Fully offline - doesn't need an internet connection.
- To move it to another computer, you can simply copy the installation folder containing a few files.

Back

All examples of numbers, formulas, dates and lists in this Help file use the International/English language settings:

> **Decimal point:** .
> **Thousand separator:** ,

> **List (formula parameter) separator:** ,

Thus, you must enter formulas and numbers as, for example:

```
=SUM(5589.7, 56.55, field1)
```

If your Windows language settings change, e.g. the decimal point or the above separators, you'll need to adapt those examples accordingly. If your language settings look like:

> **Decimal point:** ,
> **Thousand separator:** [space]
> **List separator:** ;

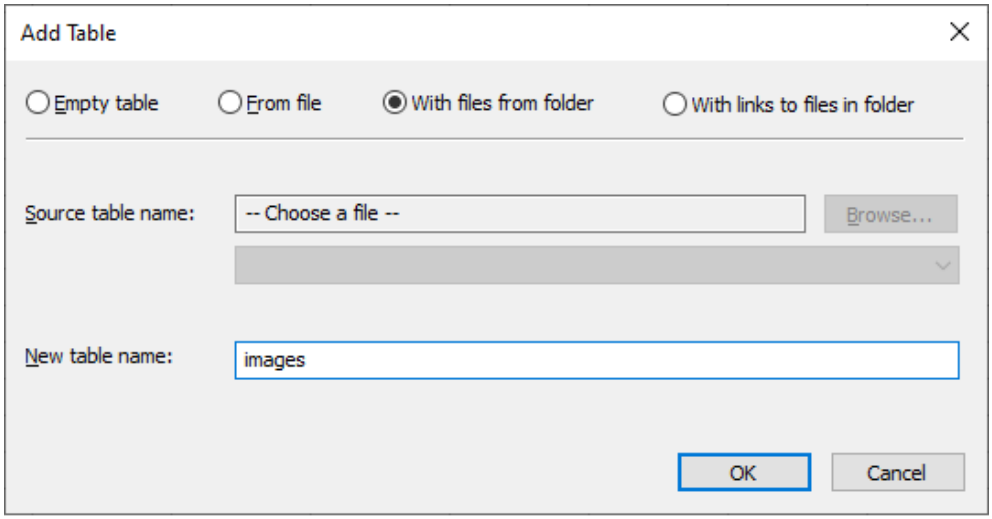
then you need to change the notation to:

```
=SUM(5589,7; 56,55; field1)
```

By default, GS-Base uses the current user's Windows system Language/International Settings. You can also override this by choosing the option **Settings > Locales > Generic (Fixed)**, which will cause the International/English settings to be used regardless of the Windows language version.

To Create a New Database

1. On the File menu, click **New Database**.
2. In the displayed Add Table dialog box, enter the name of the first table. Optionally, you can choose to import that new table from another existing file/database in any of the supported file formats.
   To add other tables or folders later, use the **File > New** commands.



3. If this is a new empty table, you'll be prompted to add at least one field.
   To add other fields later, use the **Database > Insert Field** and **Database > Append Field** commands.

4. **Using date/time fields:**
   Any **Text** field containing (as shown below) generic date/time strings can be used as a date/time field.
   This standard/generic format is required by all built-in date/time functions (except for **datevalue()** and **timevalue()**) and ensures proper sorting.
   To display and enter dates/times in the local system format and to display the calendar control (see **Settings > Options > Editing > Use date picker when editing dates**), set the desired "date" or "time" style for such a **Text** field.
   Text fields containing non-generic dates/times should be converted with the **Edit > Convert Field Data > Text To Standard Date String** command prior to setting the "date" style.

Field Types Available in GS-Base

Type	*Examples*	*Comments*
Numeric	123 -123 123.09 1.23e+02	The actual decimal and thousand separators depend on your system regional settings and the current **Settings > Locales** menu selection.

Type	*Examples*	*Comments*
		Max. positive value: 1.8+308 Min. positive value: 2.2-308 Precision: up to 15 digits
Text	Dog short text 1 2 3 11:34:30 PM 7/20/2007 P1Y2M3DT10H30M50S.000 -P120D	Any text containing up to 8169 bytes (*). When entering date/time and time period values, you must use one of the two string forms: **(1) the current date/time style** that you specified for a given field in the "Format / Style" dialog box, for example: 7/20/2007 7/20/2007 23:34:30 or **(2) the generic date/time and period** string form as specified by www.w3.org for the (XSD schema) date/time/period data types, for example: 2006-01-31 2006-01-31T13:10:55 2006-01-31T13:10:55.123 2006-01-31T23:30:00+02:00 13:10:00 13:10:55.123 13:10:55-00:30

Type	*Examples*	*Comments*
		P1Y2M3DT10H30M50S.000 (which means a period of: 1 year, 2 months, 3 days, 10 hours, 30 minutes, 50 seconds, 0 milliseconds) -P120D (which means a period of minus 120 days) PT63H (which means 63 hours)
Long Text		Any text of any size. By default the text is saved in the Rich Text (*.rtf) format and can contain any supported formatting and inserted graphics or objects. Right-click the field to display the context menu and available editing commands and options. Clicking the **Edit in External Application** command, you can edit the text in the application associated with the *.rtf or *.txt format. Changes are saved automatically after choosing the **Save**

Type	*Examples*	*Comments*
		command in that application. To save only plain text (which may result in smaller files and/or faster loading/saving if there are a lot of entries of that type), change the corresponding option in the **Settings > Options** dialog.
Images/Files		Any number of images or any other files. Files other than *.jpeg, *.png, and *.bmp are represented as icons associated with their file extensions. Use the context menu or the **Settings > Options** dialog box to turn on/off displaying thumbnails and/or file information (file name, size, modification date). If you insert some link *.lnk files linking to some disk files and if you later change the location of those target files, you can specify the new shortcut path globally in the **Settings > Options** dialog.

Type	*Examples*	*Comments*
		Right-click or double-click a given object to display the context menu and available editing commands and options. Clicking the **Edit in External Application** command, you can edit the object in the application associated with the file type of that object. Changes are saved automatically after choosing the **Save** command in that application.
Code		Any text of any size. The type/language of code is determined by the **Subfield** selection in the **Field Setup** dialog box. The code editor uses line numbering and syntax coloring based on the Scintilla (c) project.

(*) - The maximum number of bytes refers to the UTF-8 string representation. In practice, for users using Latin characters this value equals the number of characters;

however, for example, characters in far-eastern languages require a few-byte UTF-8 sequence, hence the character limitation will be smaller.

GS-Base databases are zip archives saved either with the *.zip or *.gsb extensions. For more information about which extension you should choose, please see the Default file extension option in the Settings > Options dialog box.

There are two zip formats that can be used in GS-Base: zip64 and the standard zip (zip32). Zip32 files have a limit of 4GB for one database file, and the number of all memo fields and inserted images (in other words: zip streams/entries) in one file is approx. 64K. Zip64 doesn't have the above limitations, but some 3rd party zip archivers may use their own non-standard zip64 format variants, so if for any reason you need to edit such files without GS-Base, you should make sure the data exchange is possible.

The ZIP64/ZIP32 button on the main window status bar indicates which zip format is currently in use. You can click it and switch between them freely. If you try to save a zip32 file with the amount of data that might exceed the zip32 limits, GS-Base will display a warning message.

Simplified GS-Base zip files contain only plain text files representing tables and can be created manually by users or by other applications. Typically they can be used if you want to transfer a bunch of text files to GS-Base at once, without using COM programming. For details, please see: Simplified GS-Base files.

A database zip file contains files of the following categories:

1. Database records are stored as `[table name].txt` text files. These are plain comma-separated UTF-8 text files containing field names in the first row and records in the subsequent rows.
   You can edit these files manually without GS-Base using any text editor capable of saving UTF-8 text files.
   Deleting such a file from the database zip archive means deleting all records.
   If you insert any text (*.txt, *.csv, *.tsv, *.tab) file into the zip, GS-Base will detect and report new table(s) and will enable you either to set up and add it to the database structure or to retain it in the zip as an unused file.

2. **Long Text**, **Images/Files** and **Code** field subfolders. Each such subfolder has the following form:

   ```
   [table name].r[9-digit record number]f[5-digit field number]
   ```

   For **Long Text** fields, each such subfolder contains one *.rtf or *.txt file.
   For **Images/Files** fields, each such subfolder contains any number of any files.
   Code field subfolders contain one *.txt file.
   You can edit/add/remove these files manually without GS-Base and add/remove/edit subfolders or their names.

3. One configuration **config.dat** file containing various information about the database: the order of tables, page settings and formatting.
   If you delete this file manually, GS-Base can still use all the record tables and binary fields and re-import them automatically as if they were new tables.

4. One **META-INF/manifest.txt** file containing the list of all files used by the database. If the file is encrypted, that file also contains various encryption parameters.
   If the database is not encrypted, deleting this file doesn't affect the database.
   If the database is encrypted, you must not modify that file in any way or you'll risk losing the data.

5. Any number of any files not used by GS-Base that you can drag-and-drop into the database zip file manually.
   If the **Settings > Options > Show the unused files list** option is on, GS-Base will report detecting such files. When the database zip file is saved, all such unused files are stored in the separate **unused/** zip subfolder.

Note: If your database contains a very large number of memo/image/files/code objects (e.g. thousands), it's recommended to use the *.gsb extension, as processing such zips by Windows (e.g. when Windows performs file searching/indexing) might be very slow.

Related Topics

Password protection and encryption
Saving PDF files
Opening and saving text files

Opening and saving HTML files
Opening and saving Excel workbooks

A "simplified" GS-Base file can be created manually by users or by other applications and contains only plain *.txt text files representing lists of records of subsequent database tables. Each such text table should be accompanied by a small *.xml file specifying the text file options and field types. The *.xml files should match the names of the included text *.txt tables. If the *.xml files are not present, opening the database file will require more resources and GS-Base will display the "Open Text File" dialog box for each table.

If text options are omitted, the default text import settings will be used. If the "field name", "field type" or both are omitted, GS-Base will determine these values by reading the text tables.

Sample *.xml files:

Sample 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<gs-base-table>
<text-file-options
        use-separator="true"
        separator='tab'
        use-quoting="true"
        quoting-symbol='"'
        use-fixed-widths="false"
        fixed-widths=""
        field-names-in-first-row="true"
        text-to-numbers="true"
        text-to-dates="true"
        encoding="utf-8"/>
<fields>
        <field>
                <field-name>field a</field-name>
                <field-type>T</field-type>
        </field>
        <field>
                <field-name>field b</field-name>
                <field-type>T</field-type>
        </field>
```

```
        <field>
                <field-name>field c</field-name>
                <field-type>N</field-type>
        </field>
</fields>
</gs-base-table>
```

Sample 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<gs-base-table>
<text-file-options
        use-separator="true"
        separator=','
        use-quoting="true"
        quoting-symbol='"'
        use-fixed-widths="false"
        fixed-widths=""
        field-names-in-first-row="true"
        text-to-numbers="true"
        text-to-dates="true"
        encoding="ansi"
        iso-code-page="Default Ansi code page"/>
<fields>
        <field>
                <field-name>field a</field-name>
                <field-type>T</field-type>
        </field>
        <field>
                <field-name>field b</field-name>
                <field-type>T</field-type>
        </field>
        <field>
                <field-name>field c</field-name>
                <field-type>N</field-type>
        </field>
</fields>
</gs-base-table>
```

Related Topics

GS-Base database file format

The Database Explorer pane enables you to add, delete, re-arrange and rename fields, tables and folders. You can use the **Copy/Paste** commands, the drag-and-drop functions and the commands available via the context menu (displayed after right-clicking the explorer pane).

Dropping the dragged table over a folder places the table at the end of the table list in that folder.
Dropping a field over the table places that field at the end of the field list.

To rename any tree item, select it, then click it or use the **Rename** command from the context menu.

The **Display Field Name With** command toggles on/off displaying optional additional field information: the field type, its status ("calculated", "validation", "conversion", "default value") and whether the field is currently filtered.

To display the Field Setup dialog box, double click a given field in the Database Explorer pane or click that command on the main menu or the context menu. The dialog box offers the following options:

Field Name

Any string starting with a letter and containing up to 63 characters.
Field names should be unique within a given table, as otherwise formulas referring to record fields may produce incorrect results.

Field Type

One of the following five field types: "Text", "Numeric", "Long Text", "Files/Images" and "Code".
For details, please see: Creating new databases
Changing an existing field type may result in conversion and/or modifying the existing field contents.

Special

This value specifies whether the field features some additional functionality.

- **Calculation formula** - the specified formula will be used to calculate the current field value automatically.
  Calculated fields display calculation results only and cannot be edited. They are updated in the following situations:

  - After a given record is modified, all calculated fields within that record are updated.
  - After choosing the **Tools > Update All Calculated Fields (F9)** command, calculated fields of all records in the current table are updated.

  Creating links between two tables is a special case of using calculated fields and the **vLookUp_ex** function from the **Roll-up** formula category. It behaves similarly to the standard spreadsheet **vLookUp**.
  For example, see the included "sample.zip" database and the calculated fields used to perform calculation on record fields and to link the "products" and "orders" tables.
  For more information about the formula syntax, see: Entering formulas.

- **Validation formula or regex** - the specified formula or a regular expression will be used to validate data entered in this field. The data is accepted if the validation formula returns a numeric value other than 0/false, or if the regex matches the entire string.
  For example, if some field ("field_1") should contain only strings at least 4 characters long, you can specify the following formula:
  =len(field_1) >= 4
  For more information about the formula syntax, see: Entering formulas.

- **Conversion formula** - the specified formula or a comma-separated pair of find-and-replace regular expressions will be used to convert data entered in this field.
  For example, to ensure that the entered data doesn't contain leading spaces and first letters in words are uppercase letters, the following formula can be used:
  =proper(trim(field_1))
  For more information about the formula syntax, see: Entering formulas.

- **Default value** - a fixed value that is inserted into selected empty fields after choosing the **Insert > Default / Incremented Max (Ctrl+T)** command.
  Also see: Default field values.

- **Incremented Maximum** - the current (or initial) field maximum value is incremented by 1 and inserted into selected empty fields after choosing the **Insert > Default / Incremented Max (Ctrl+T)** command. The field must be numeric or must contain date/time values.

Statistics

Calculates the breakdown (field value/number of occurrences) for a given field. The generated sorted list has a form of plain text, so it can be selected and copied onto the Clipboard. This and much more advanced functionality is provided by Pivot Tables.

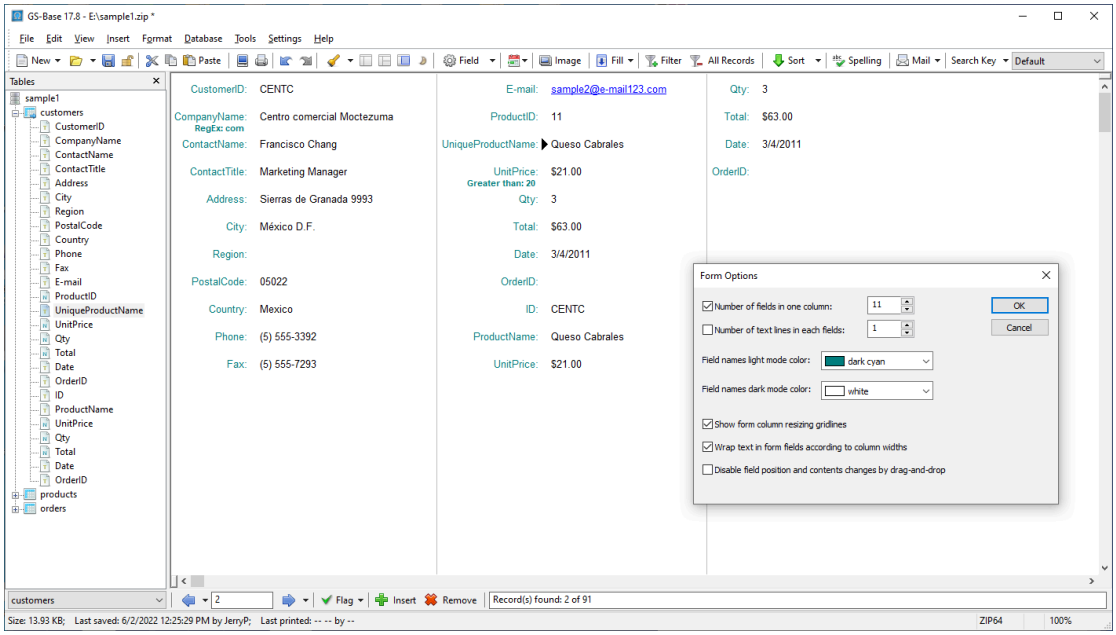
Formulas

Lists all predefined GS-Base functions that can be used in formula expressions in database fields. Database fields are referenced in these expressions by name.

GS-Base Help > *Using table, form and binary views*

By default, records in a newly created database are displayed in tables. Tables can be split into separate views to enable synchronized scrolling of different regions of a given table.

Form views can be activated using the **View > Form** command. They can be displayed side by side with the corresponding table view, or the table view can be hidden.
The form layout is mostly automatic. You can specify various other options using the **View > Form Options** command.
Multi-column forms enable you, e.g., to view/inspect tens of fields at once.



Binary views are used to display Long Text, Images/Files and Code fields. They are opened automatically if you add such contents. You can still explicitly show/hide them using the **View** menu commands, or simply by pressing Enter or double-clicking the corresponding table/form field.

GS-Base Help > Adding Text Notes, Images/Files and Code Snippets Outside of Records

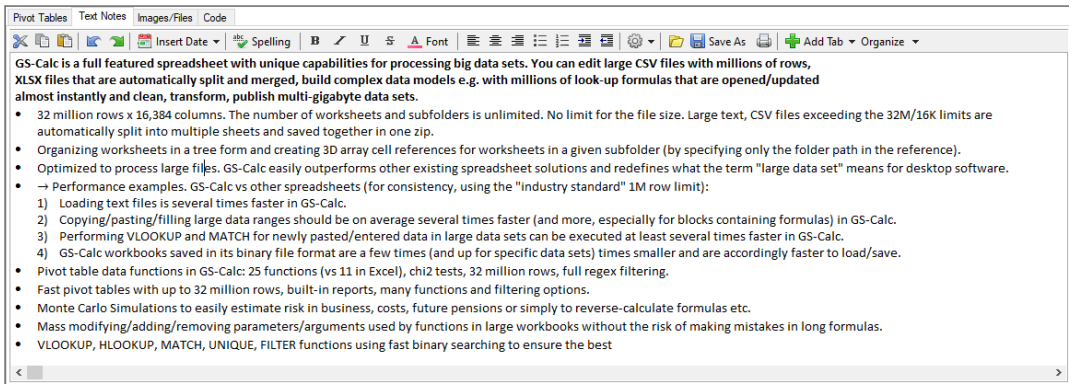
To Add Various Types of Notes to the Database Files

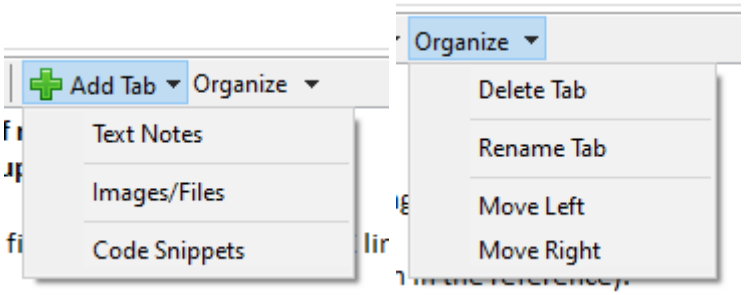
Use the **View > Notes** command to open the bottom window pane with notes (and pivot tables).

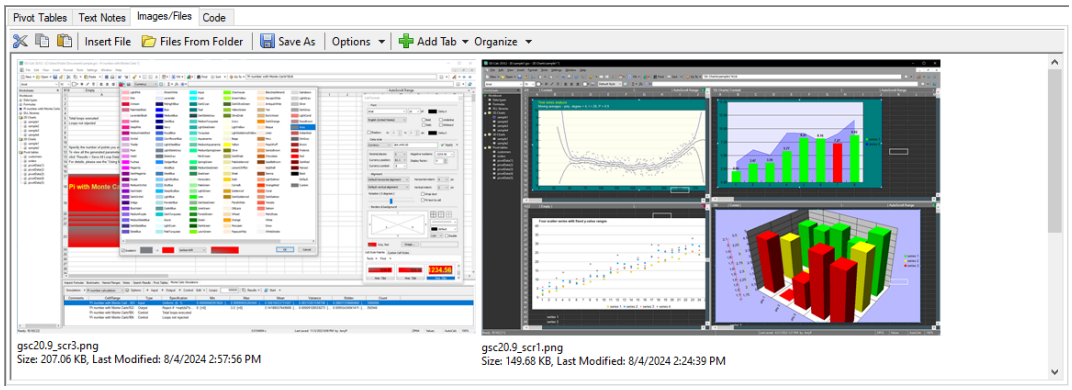
Aside from the always-present "Pivot Tables" tab, initially there are three tabs: "Text Notes", "Images/Files" and "Code". They are the same as the corresponding regular long binary fields in records, with some added functionality. They just allow you to store any type of data without creating a table with records. You can add any number of such tabs of each type, in any order.
Each "Text Notes" and "Code" field can store up to 4GB of data. In each "Images/Files" field you can store any number of objects, each one up to 4GB.

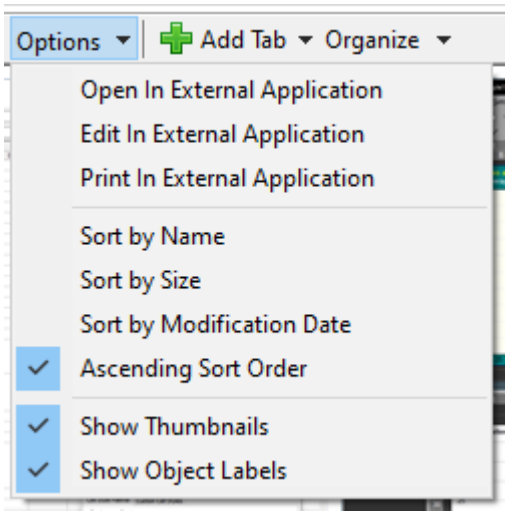
If you use the **File > Protect** command to encrypt your database file, all the data stored here will be encrypted as well, using the industry-standard, strong and safe encryption method, so you can use GS-Base files also as encrypted containers for any type of data in its original format.

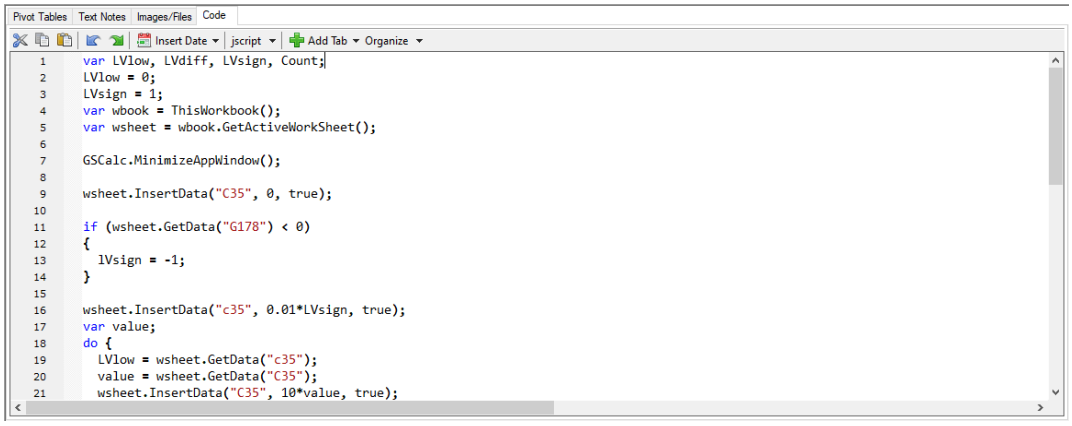
Note: If you want to keep a large amount of text data and font styles are not required, you can consider using the "Code snippets" tabs with the syntax coloring set to "none". For very large documents, they'll be faster than the "Text Notes" tabs, which rely on the slower Windows built-in editor.

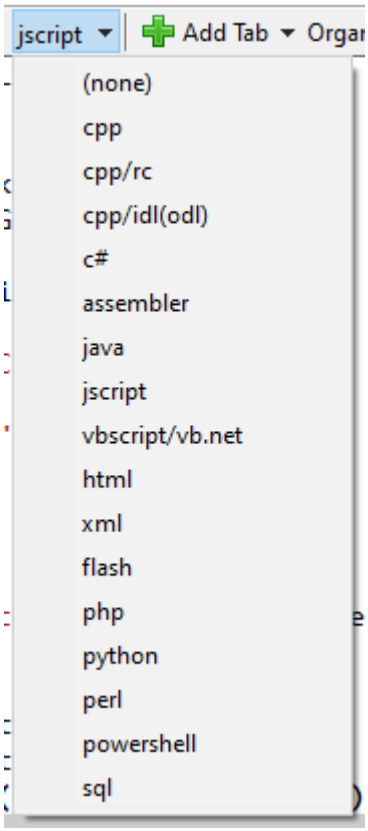







Options
Add Tab
Organize
Open In External Application
Edit In External Application
Print In External Application
Sort by Name
Sort by Size
Sort by Modification Date
Ascending Sort Order
Show Thumbnails
Show Object Labels


Pivot Tables
Text Notes
Images/Files
Code
Insert Date
jscript
Add Tab
Organize
var LVlow, LVdiff, LVsign, Count;
LVlow = 0;
LVsign = 1;
var wbook = ThisWorkbook();
var wsheet = wbook.GetActiveWorkSheet();
GSCalc.MinimizeAppWindow();
wsheet.InsertData("C35", 0, true);
if (wsheet.GetData("G178") < 0)
{
lVsign = -1;
}
wsheet.InsertData("c35", 0.01*LVsign, true);
var value;
do {
LVlow = wsheet.GetData("c35");
value = wsheet.GetData("C35");
wsheet.InsertData("C35", 10*value, true);



Note: The first "Pivot Tables" tab contains pivot tables created independently for each database table. This tab can't be moved or deleted.
All other tabs in this pane/view contain data shared by all tables in a given database file, and can be added/deleted/re-organized.

Use the available local toolbar commands to store and format any text (up to 4GB per tab), images or other files, or code snippets with syntax coloring for a number of programming languages.

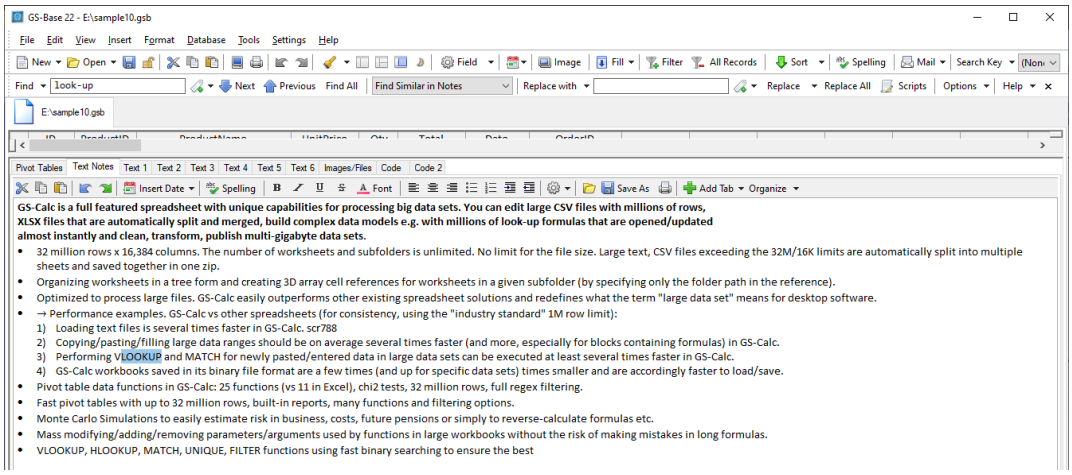
You can also use the drag-and-drop functions to exchange the data with database fields or other applications.

Click the "Add Tab" button to add any number of notes of any type. A new tab is inserted after the current one, so you can easily group tabs of the same type.
The "Organize" menu commands enable you to delete, rename and move the tabs.

All tabs are searchable in one pass - clicking "Find Next/Previous" causes searching all textual data in all tabs and selects the found string.
For the "Images/Files" field, the file names are searched.

You can use two search modes:

- **Find In Notes** - searching using regular expressions
- **Find Similar In Notes** - fuzzy text searching without regex



To Edit Numeric and Text Fields in Tables/Forms

Do one of the following:

- Press **Enter**, **F2** or any letter/digit key.
- Double-click a given field.

You can enter up to 8169 characters in the standard **Text** fields. The **Long Text** and **Code** fields can contain up to 4GB of text. The **Images/Files** fields can contain any number of objects. In GS-Base 16.4 and older versions, the total size of inserted objects and the size of the database file itself can't exceed 4GB, and the total number of binary field entries can't exceed 65K. In newer GS-Base versions the zip64 file format replaced the standard zip format and all the above limitations were lifted (they now equal thousands of TB).

- To complete editing, press **Enter** or - if editing was started by pressing any character key - any of the cursor keys, or simply click the table/form.
- To enter a new line in the edited cell, press **Ctrl+Enter** or **Alt+Enter**.

- To cancel changes, press **Esc**.

For information about entering/editing Unicode characters, please see: Entering Unicode characters.

If you set some standard (not using custom patterns) numeric or date/time format for a given field, GS-Base will try to interpret the data according to that format. Fields formatted using custom patterns require you to enter the data in the default format or in one of the standard formats. The date/time style can be applied to **Text** fields only.

Note: If you want to apply the date format to a text field that already contains some non-standard (that is, other than YYYY-MM-DD) text strings, you should first use the **Edit > Convert > Text Strings To Date Strings** command (or re-enter each date individually). This step is required to ensure correct sorting of such a field.

Trying to edit a **Long Text**, **Images/Files** or **Code** field within a table/form results in the following actions:

1. If the **Options > Settings > Table/Form Editing Options > Edit LongText/Files fields in new windows** option is **unchecked**, GS-Base will open/hide a new pane displaying that data for editing/viewing.
2. If the **Options > Settings > Table/Form Editing Options > Edit LongText/Files fields in new windows** option is **checked**, GS-Base will launch an application associated with the specified data type (e.g. typically Wordpad for text and Paint for graphics). Choosing the "Save" command in such an external application will automatically update the data in GS-Base.

For action (2), the following additional functionality is supported:

- Pressing a digit key opens either the text or the n-th (1...9) object for editing in an external application.
- Pressing a letter key prints either the text or the n-th (a...z) object in an external application.

To Edit Long Text, Images/Files and Code Fields

These fields are saved automatically. They feature all the standard functionality such as drag-and-drop (which includes dropping files/images from other

applications) and additional commands (e.g. exporting and importing text, formatting and display options) accessible through the context menus.

To edit a given memo field or a given image/object in an external application, right-click that content to display the context menu and choose the **Edit In External Application** command. The data will be saved in GS-Base automatically if you use the "Save" command in that external application.

If a given binary field is not empty, the respective table or form field contains some textual information describing it. If you want to delete the entire **Long Text**, **Images/Files** or **Code** field contents at once and remove the respective entry from the database zip file, simply delete the corresponding table/form cell.

To Move/Copy/Paste Field Contents

Tables: Place the mouse cursor over the top edge of the selection and drag it. You can move the data within the same window, different windows or applications - by default GS-Base copies (and provides to other applications) the table data as text and bitmaps.
If you drop file(s) within the table view, GS-Base either pastes the file path if the cursor was over a hyperlink field, or displays a dialog box enabling you to choose into which **Images/Files** field that file(s) should be inserted.

Binary Fields: The standard Windows drag-and-drop procedure applies. The dropped objects are always copied, not moved. If you want to insert a link into the **Images/Files** field, press and hold down the **Ctrl+Shift** keys at the same time.

To Fill a Given Range With Some Values

1. Copy the value of the selected field or fields.
2. Select the destination range. The destination range cannot contain more columns than the source range. If it does, the additional columns are ignored and won't be filled.
3. Paste the copied data.

To Use a Drop-Down List to Enter Data

Use the **Format > List** command and specify which list should be used for a given field(s). The same list can be shared by any number of fields in any number of tables in one database. A database file can contain up to 255 lists. The number of list items is unlimited.
Drop-down lists can be static (with values pre-defined by the user), and they can automatically add each new unique value entered by the user in the associated field(s).

To Use a "Date and Time Picker" Control to Enter Data

The control can be displayed for each text field that (1) is formatted using one of the standard "Date" styles (not using custom patterns), and (2) doesn't have any drop-down list attached to it at the same time.
Displaying that control can be enabled/disabled in the **Settings > Options** dialog box. The following shortcut keys are used by the date/time picker control:

Keys	***Action***
Arrow keys	Changes the active control field and/or changes control field values
End and Home	Sets the maximum and minimum values for the current part of the date (a year, month or day)
Plus and Minus	Increments/decrements the current part of the date (a year, month or day)
Alt+Down	Displays the "Month Calendar" control. Same as clicking the "drop-down" arrow.
PageUp, PageDown	If the "Month Calendar" control is active, moves to the previous/next month

Unicode characters can be edited/entered using the following methods:

1. Enter a sequence of 1-6 hexadecimal digits and immediately press **Alt+X**. Note that any preceding and not separated valid hexadecimal digits will be converted as well. This method can be used only when editing fields; it's not available in dialog boxes.
2. Press **Ctrl+Shift+U**, then enter a sequence of 1-6 hexadecimal digits and immediately press **Space**.
3. Press and hold down **Alt**, then on the numeric pad press **+** and a desired hexadecimal code, then release **Alt**. This method may require adding the following key in the Windows registry:

   ```
   HKEY_CURRENT_USER\Control Panel\Input Method\EnableHexNumpad
   ```

   The value of the above key should be set to "1" (entered as REG_SZ).

To Select a Range of Fields and/or Records in the Table View

Do one of the following:

- Use the mouse cursor.
- Press **Shift** and use any of the scrolling keys.
- Press **Ctrl+Space** to select the current field in all records.
- Press **Shift+Space** to select the entire current record.

To Jump to the Desired Record or to Select a Record Range

Click the **Record** field on the bottom toolbar or press **Ctrl+G**, then enter the record number or a record range (as two numbers separated by ":") and press **Enter**.
You can also use menus attached to the "Previous"/"Next" buttons to scroll up or down by one record or to some specific record:

- Top / bottom
- Previous / next non-empty field
- Previous / next empty field (Ctrl+Up / Ctrl+Down)
- Previous / next identical field value (Ctrl+Alt+Up / Ctrl+Alt+Down)
- Previous / next different field value

Navigation / Shortcut Keys

Up, Down, Left, Right	scroll by one row/column
Ctrl+Up, Ctrl+Down	go up or down to the next empty field
Alt+Ctrl+Up, Alt+Ctrl+Down	go up or down to the next field with the same content
PgUp, PgDn, Alt+PgUp, Alt+PgDn	scroll by one page vertically or horizontally
End	last non-empty column
Home	first column
Ctrl+End	last non-empty row
Ctrl+Home	first row
Ctrl+PgUp	previous table (in the print-preview mode: previous page)
Ctrl+PgDn	next table (in the print-preview mode: next page)

Other Shortcuts

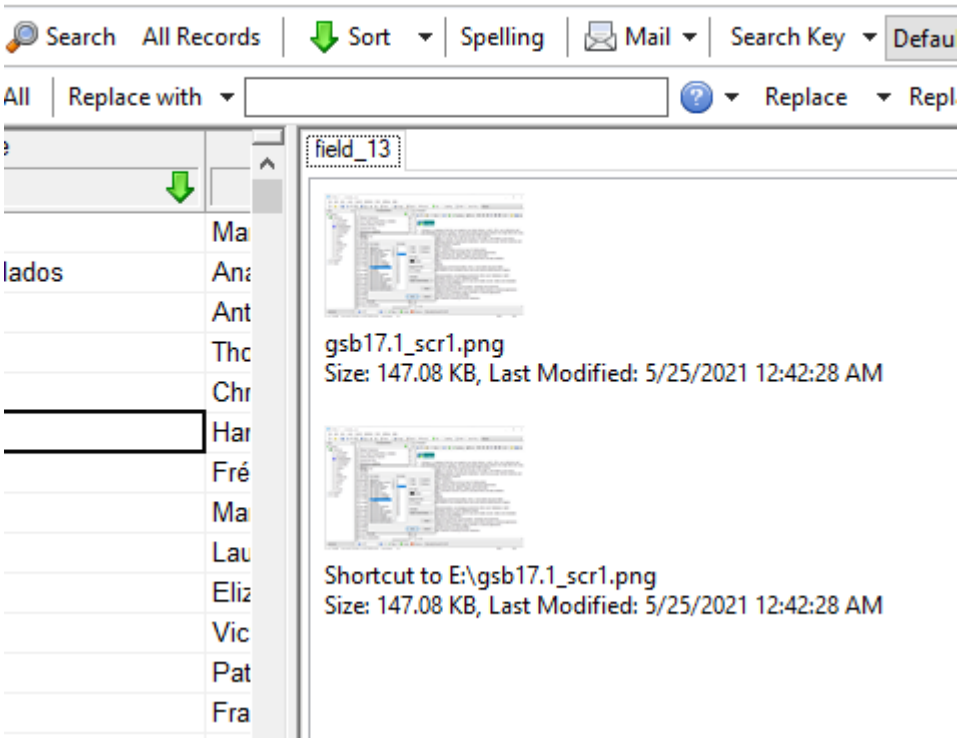
Ctrl+B	bold font
Ctrl+I	italic font
Ctrl+U	underline font
Ctrl+K	strikeout font
F6	next pane
Shift+F6	previous pane

To Use the "Auto-Scroll" Function

Click the worksheet window using the mouse wheel button and choose the desired scrolling speed and direction by moving the mouse cursor around the clicked point.

GS-Base Help > *Inserting images/files vs inserting links to images/files*

Images and files can be inserted in the Images/Files fields as objects that are stored in the database files along with tables, or as links to files that are stored on the user's disk and have to be copied separately whenever the database is moved to another computer. On the other hand, if the total size of all objects is very big, a database with embedded objects will have to load them into computer memory all at once when that database is opened, while linked objects will be loaded gradually, only when they are displayed. For images, both methods result in displaying them in the fields in the same way, for example:



For linked objects, so that you don't need to keep their exact paths consistent across different computers, GS-Base enables you to define the **Settings > Relative Shortcut Path** folder where you intend to copy the linked files. If a given original link points to a location which doesn't exist, GS-Base will look for the specified file name in that alternative folder.

To Insert an Image or File in the "Memo/Files" Pane

Use one of the following methods:

- Use the **Insert > Image/File (Ctrl+M)** command. If no "Images/Files" fields exist, GS-Base will offer to create one.
- Click the "Insert Image/File" toolbar button.
- Drag-and-drop the desired file.

To Insert a Link in the "Memo/Files" Pane

Use one of the following methods:

- Use the **Insert > Shortcut to Image/File (Ctrl+Shift+M)** command. If no "Images/Files" fields exist, GS-Base will offer to create one.

- Press Shift and click the "Insert Image/File" toolbar button.
- Drag-and-drop the desired file while pressing the Ctrl+Shift keys.

To Insert a Link Outside the "Memo/Files" Pane

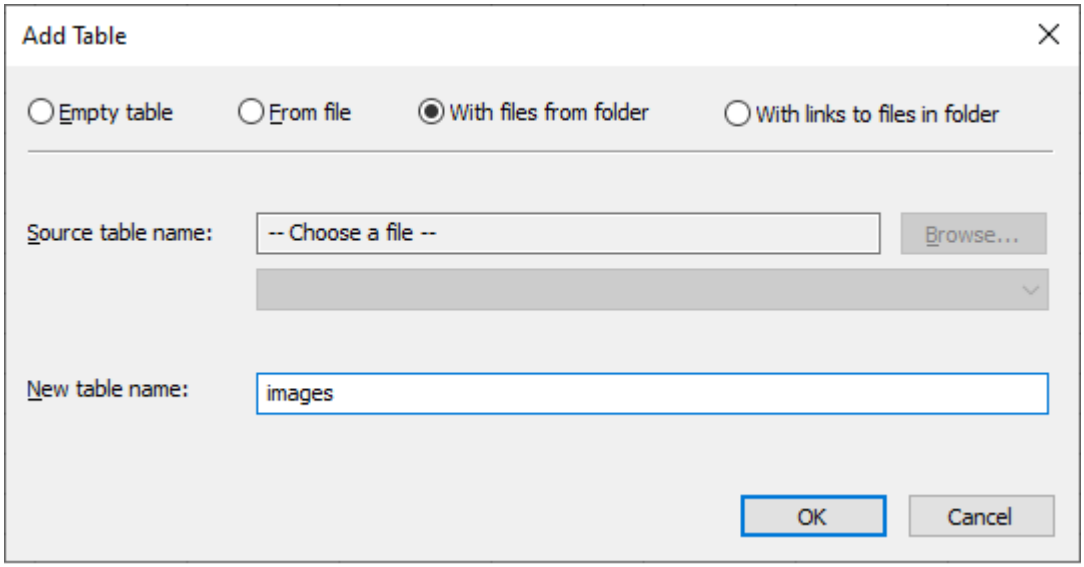
That is, in a table field or a plain text field:

Unlike the "Memo/Files" pane, which displays the linked object itself, this method only shows the clickable file path as plain text — faster to browse when a visual preview isn't necessary.

1. Format a given field using the **Format > Hyperlink** command.
2. Drag-and-drop the desired file to that text field location. This will cause the link/path to be inserted in that text field. (Another option is to simply enter the path manually.)
   Clicking such a link corresponds to the "Edit" Windows shell command for the linked file.

GS-Base Help > *Loading images/files from entire folders*

To load all images/files from a given folder, use the **File > New Database** or **File > New Table** command and, in the displayed **Add Table** dialog box, choose the option "With files from folder" or "With links to files in folder".
The first option inserts files directly in the database file, the second one only links to files in that folder.
This will create a table with records with one loaded image/file or link per field.



Note 1: By default the loaded files are placed in the newly created "Images/Files" field. If these are images, they will be displayed in the binary field panes as usual. All other files will be displayed as their associated icons. If these are, e.g., text, HTML, code snippets, etc., or any files containing textual contents, you can later use the **Database > Field Setup** command and dialog box to change the field type to "Long Text" or "Code" and display the actual text in the binary field pane.

Note 2: You can also load all images/files from a given folder into a single "Images/Files" field, or save all images/files from a given "Images/Files" field to a folder. To do this, right-click that "Images/Files" field to display the context menu.

GS-Base Help > Using the Increment, Decrement and Operations commands

To Add/Multiply/Divide/Subtract Values in Selected Fields Automatically

Select a numeric or date/time (a text field with the date/time format) field value or any group of such values within a table and use the **Edit > Increment (Ctrl+Alt+'+')** or **Edit > Decrement (Ctrl+Alt+'-')** command. By default, for numeric fields these commands respectively add 1 and subtract 1. For textual date values they add and subtract one day, and for textual time they add and subtract one hour.

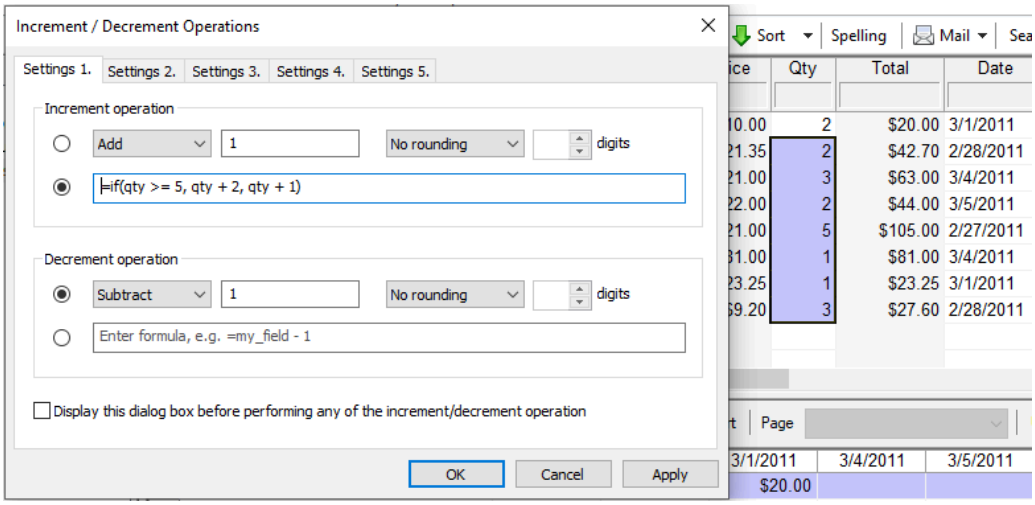
The above plain adding and subtracting can be overwritten with the **Edit > Operations** command to perform more complex field value changes.

To Define Operations Executed When You Use the "Edit > Increment" and "Edit > Decrement" Commands

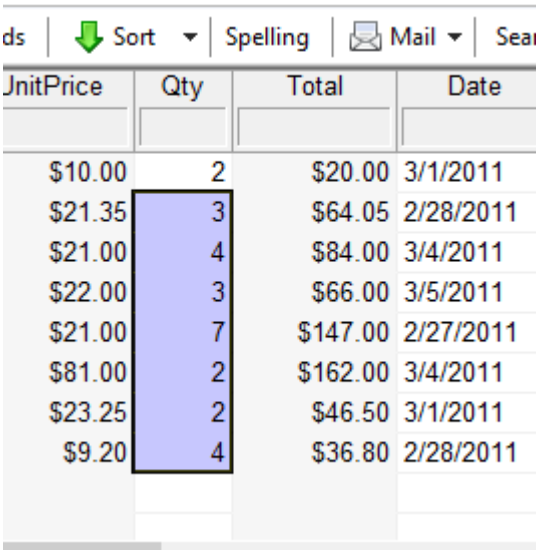
Use the **Edit > Operations** command and specify the incrementing and decrementing operations, which can be adding/subtracting/dividing/multiplying with a fixed value and with a given precision. The operation can also be defined as a formula referring to a given field (or fields).
There are five different increment/decrement pair settings, and the active one is determined by the active tab in the **Operations** dialog box.

For example:

1. Define some "increment" formula-based action on the "Settings1" tab and make it active.



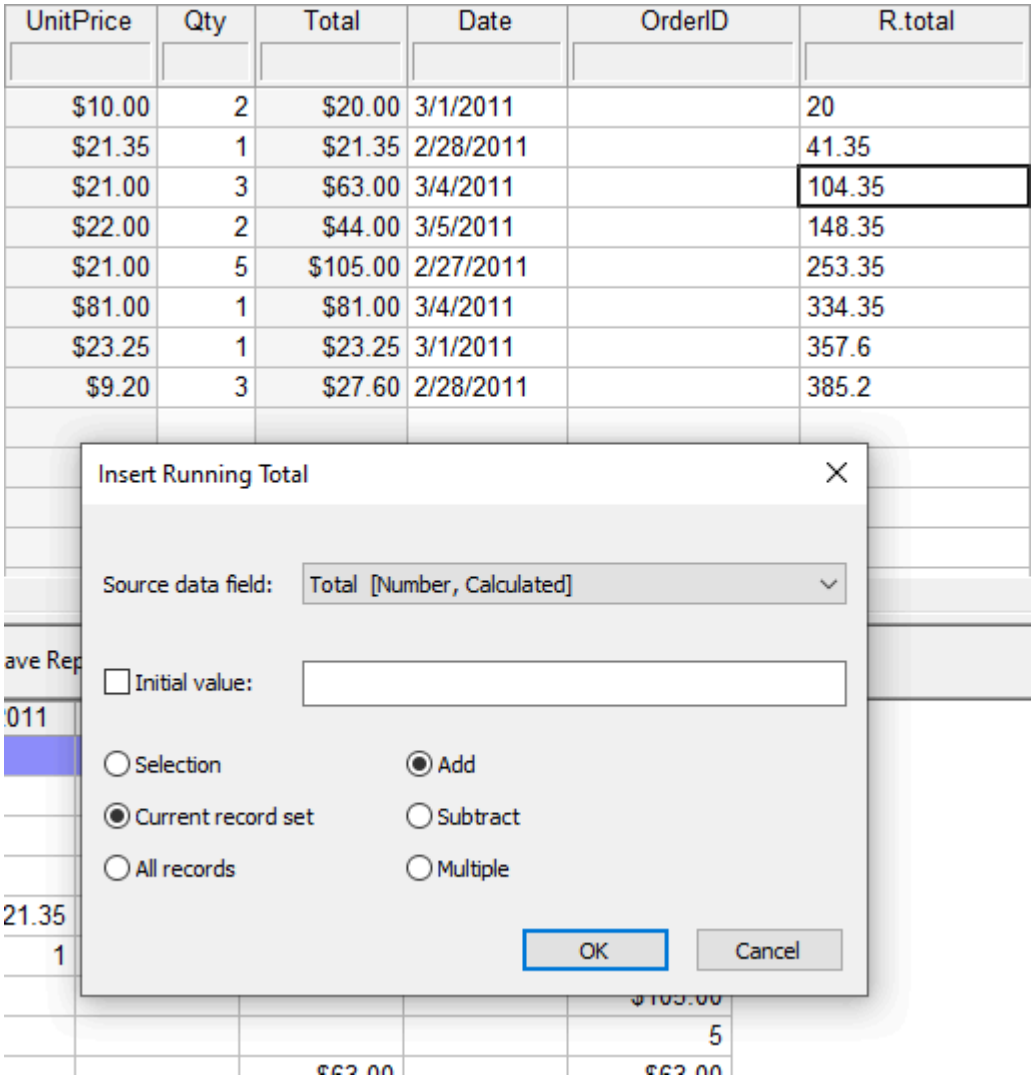
2. Use the "increment" command (Ctrl+Alt+'+'). Values in the selected fields will be modified according to the defined formula:



To insert running totals and moving averages in a given field, use the **Insert > Running Total** and **Insert > Moving Average** commands for that field.
The displayed dialog box enables you to specify the source data field these values are calculated for and the following options:

Running Totals

- The initial value the total is set to
- The total type: sum, difference or product
- The range of the fields to be processed: current selection, all currently filtered records, all records in the table

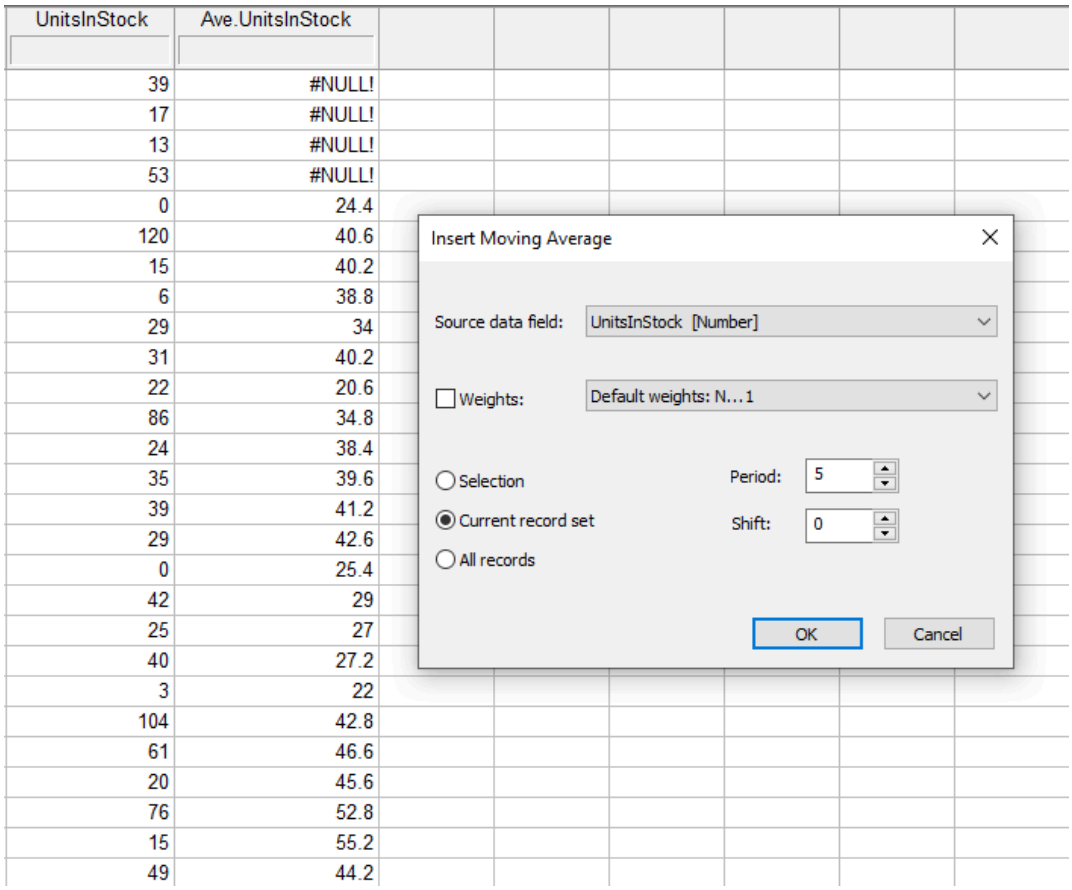


Moving Averages

- Weights - check this option to apply weights to data series values. The default "N...1" selection means that the latest value in the N-element period will have the weight equal to N and the oldest one: 1.
  You can specify your own weights as a column in some other table. The number of weights should be equal to the size of the period, and the first row corresponds to the latest period value. If the number of weights is smaller, the remaining (missing) weights are assumed to be 1. If it's greater, the last weights are ignored.
- Period - the number of values each average is calculated for.
- Shift - For the default 0 value, each period for each average ends in the row/record the average is inserted in, and starts N rows/records above it. As there isn't enough data for the initial N-1 averages, the corresponding

fields will be filled with the #NULL! error values.
Positive 'shift' values move this range down/forward, and negative shift values move it further up.

- The range of the fields to be processed: current selection, all currently filtered records, all records in the table



GS-Base Help > *Inserting series*

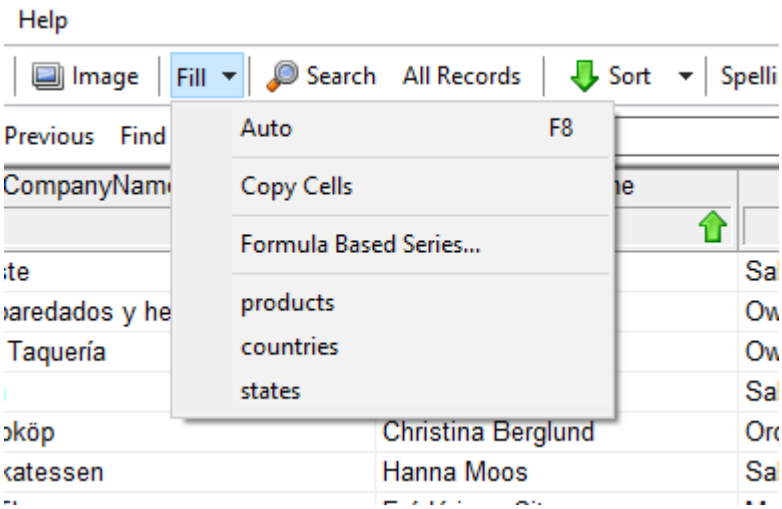
Inserting Automatic Series (F8)

This will fill a given range of fields with values that can be numbers, dates/times, built-in lists (day/month names) or custom lists.
The type of the series is determined automatically by the contents of the first field in that range. The "step" (for numbers, dates/times) is determined automatically

by the value of the next field in that range, and if the next field is empty or contains incorrect data types, a unit step is used (1, one day, one hour).

If the contents of the first field don't match any of the above series types, a simple "copy" operation is performed.

- Built-in list (day/month names) items can be substrings (words) within text strings in fields. For example, if the first field contains "departure on Monday", the rest of the inserted series will be in the same form: "departure on Tuesday", etc.
- **Custom list items can also be substrings (words) within a field, but to be recognized, instead of the F8 command such series must be used explicitly by its name, by clicking the Fill button.**

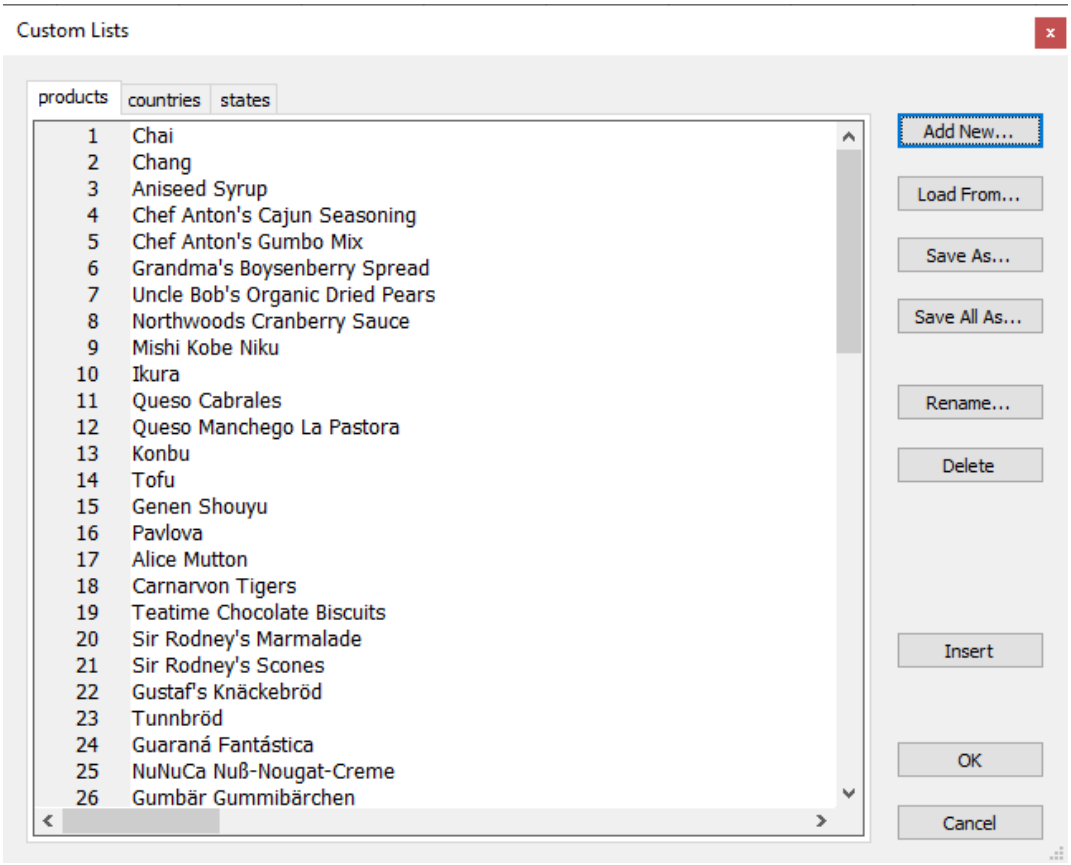


Choosing a Specific Custom List to Insert Series

Click the **Fill** toolbar button and choose the desired list, or use the **Manage** dialog box and use the **Insert** button.

If you choose a specific custom list by name:

- Custom list items can also be substrings (words) within a field, but to be recognized, instead of the **F8** command such series must be used explicitly by its name, by clicking the **Fill** button.
- The selected range of fields doesn't have to contain the initial series item. If it's empty, the series will simply start from the first list item.

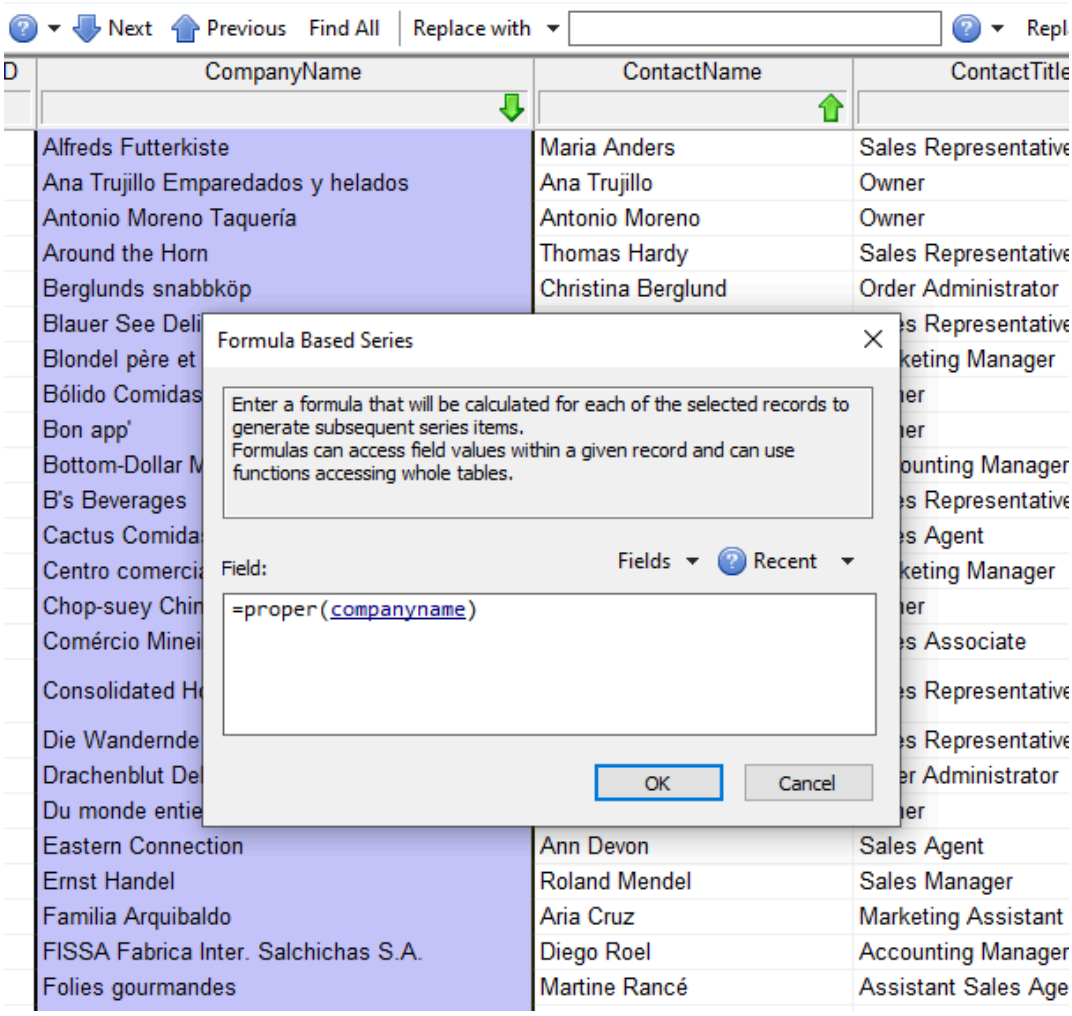


Inserting Formula-Based Series

This enables you to define any type of data series, including those referring to resources in other tables.

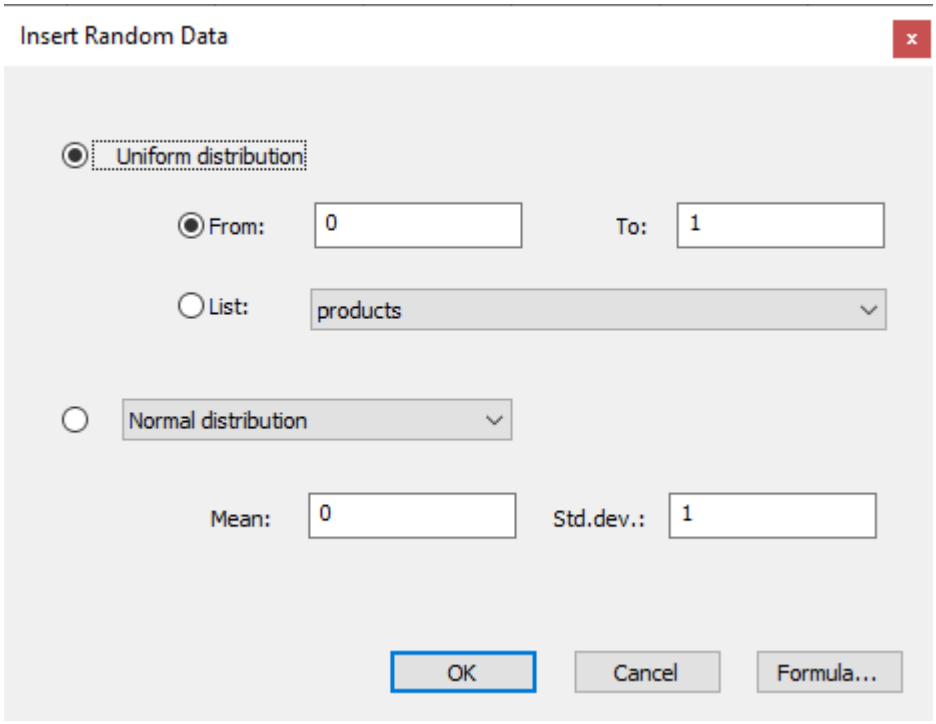
For this series type you need to define a formula that will be calculated for each record within the selected range.

Used formulas are added to the "recently used" list, which is stored in the settings file.



Inserting Random Series

You can use several popular distribution types. Additionally, for the uniform distribution you can choose to create a series of items randomly selected from a given custom list.



GS-Base Help > *Entering data: drop-down lists*

You can use the **Drop-Down Lists** dialog box to enable displaying drop-down lists for the current field(s) and/or to manage (create, edit, delete) lists in the current database.

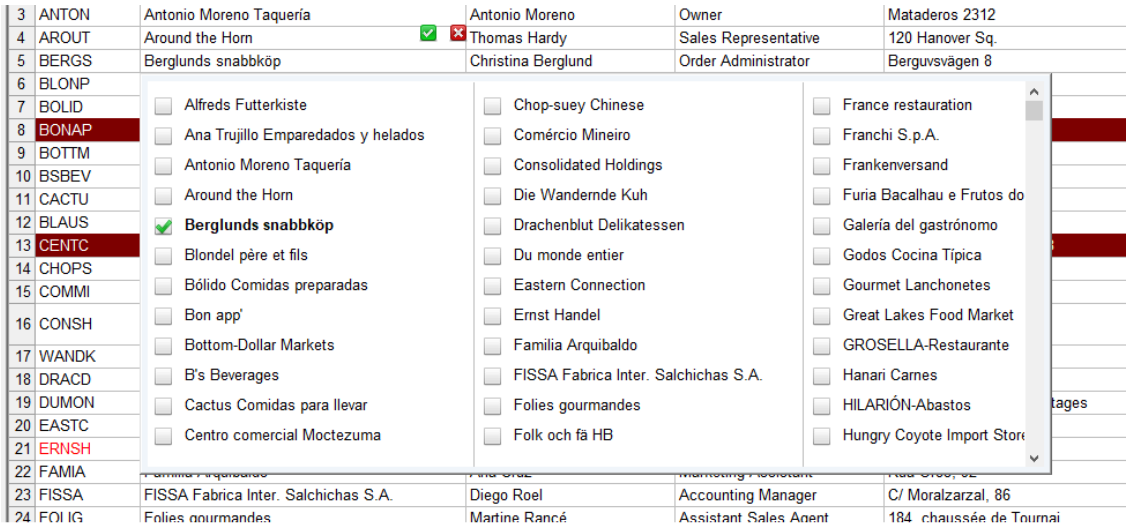
When editing a field with a drop-down list attached to it, GS-Base opens a window with a list of values with checkboxes, and you can quickly choose predefined field values instead of re-typing them.
Pressing the cursor keys **Up** (in the first line of that edited field) or **Down** (in the last line of that edited field) switches the "focus" to the displayed list window. To switch back to the edited field, scroll the list to its first item and press **Up** or simply click the edited field.

To scroll the list, use the standard cursor keys: **Up**, **Down**, **PgUp**, **PgDn**, **Home**, **End**. If a list has multiple columns you can also use **Left**, **Right** to jump between columns.

To (un)check a given list item, press **Space** or click the check-box. If the "multi-selection" option is disabled for a list, the check-box button of the current list item is always "on" and checking another one always automatically unchecks the previous one.

To accept selected list item(s), press Enter, double click (one of) the selected list item(s) or click the "OK" button displayed above the edited field.

You can display list items in **1 to 99 columns**. Dragging the vertical grid-lines in the list window resizes all these list columns. To resize the list window, drag its bottom-right corner or its edges.



To create a new list, click the **New List** button, and to add new list items, enter them in the **List Items** edit field: one item per line.

You can also choose to load list items automatically from a record field from any table in the same database file. GS-Base will load all current unique values from that field. The maximum number of list items is the same as the maximum number of records: 256 million.

If you select the **AutoAppend** option, each new unique value entered in the corresponding field(s) will be added to the list automatically. This option is disabled if the list is loaded from a database field.

If you choose the **AutoComplete** option, the edited cell contents will be automatically completed/replaced with the full found matching text from the list after you type the initial characters.

Choosing the **Validate entered text** option ensures that a given text will be accepted in the edited field only if it's already on the list.

If the **Sort items** option is not selected, the list will display all its elements in the same order as they were added/entered. Otherwise the list items will be sorted before displaying (thus for static extremely large lists that need to be sorted, it pays off to keep them sorted permanently without this option).

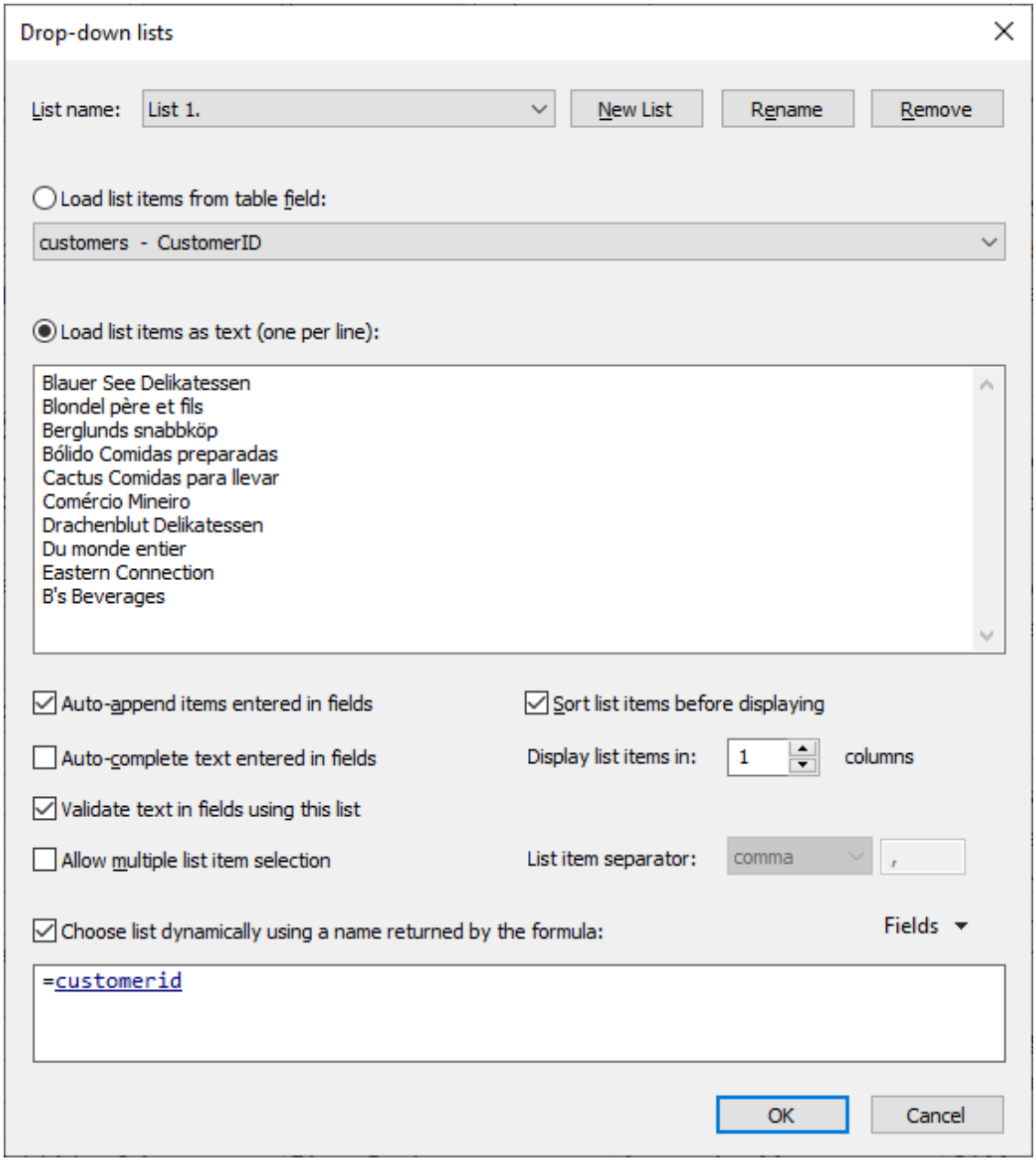
In the **Choose list dynamically using a name returned by the formula** edit field you can specify a formula returning the table name that will overwrite the default one selected at the top.

For example, if you have a table with the "MusicGenre" and "Artist" fields, you can add a drop-down list to the latter, check this option and enter the following formula for that drop-down list:

=MusicGenre

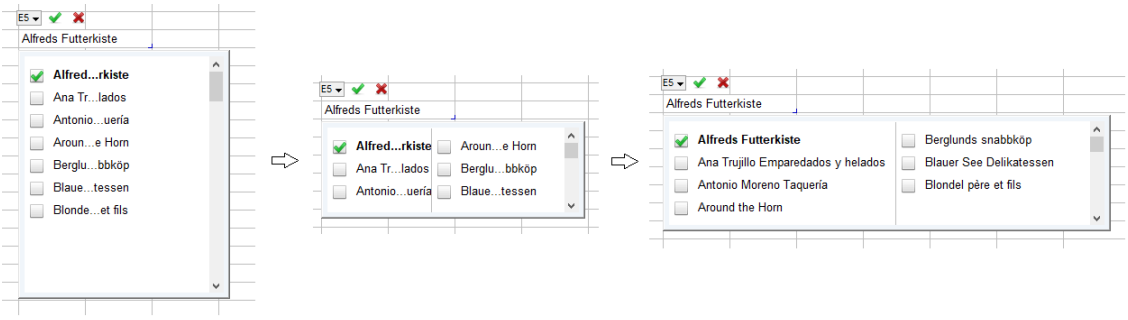
Then, if you define a separate list for each music genre and name these lists accordingly, what you enter in the "MusicGenre" field will determine which drop-down list will be displayed when you start editing the "Artist" field in the same record.

If there is no strict relation between the list names and the field values, you can use a formula, e.g. with nested IF() for a few cases, or you can use a working table with the two fields and the vLookUp_ex() function to look up which field value is linked to which list name.



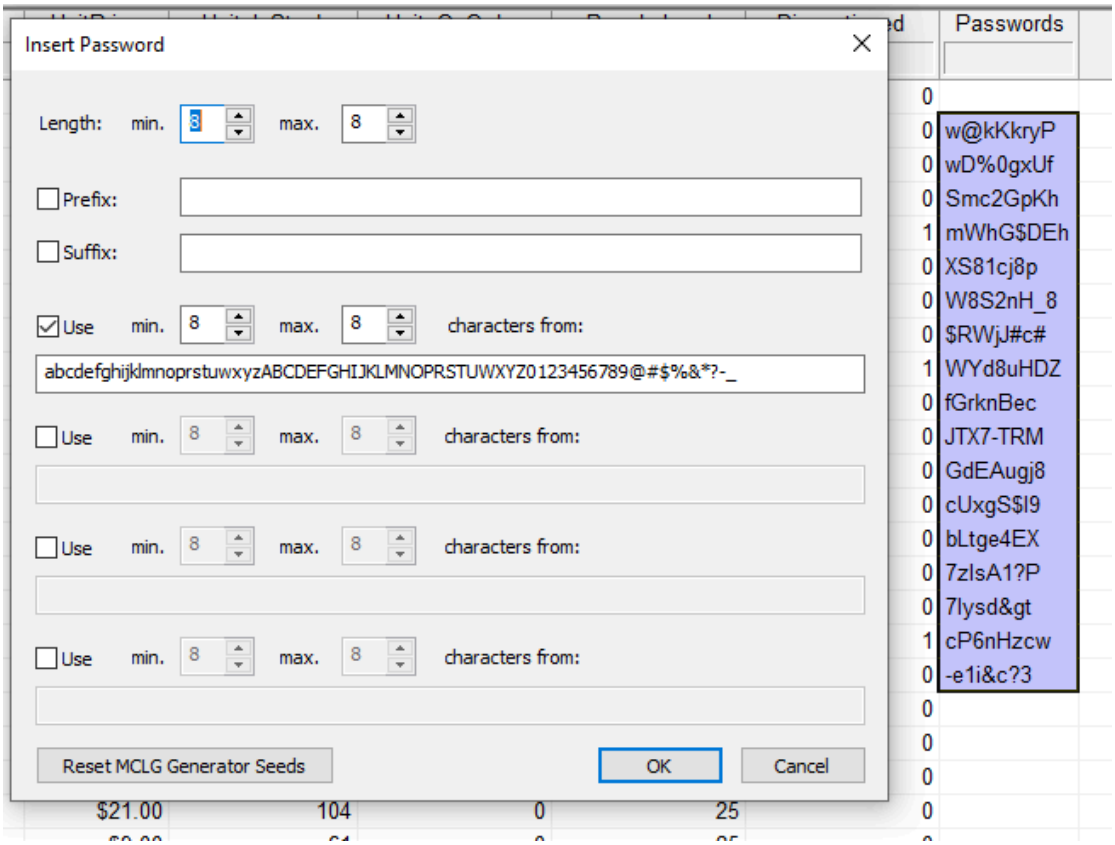
One list can be used by any number of fields in any table in a given database file. You can create up to 100 lists in one database file. To move lists between workbooks, you have to copy/paste the data from the **List Items** edit field.

Note:
If you specify the "columns" to be > 1, the splitter lines are displayed only if the list items occupy more than one column. Thus, to resize columns you might need to temporarily shrink the list window, for example:



GS-Base Help > *Generating passwords*

The **Insert > Password** command enables you to insert unique passwords (either as a series of any number of unique passwords, filling in the currently selected range of table fields, or as single unique strings subsequently used in edited text/code/memo fields). There are several options that help you increase the probability that the generated passwords are unique. For typical settings, 8-character passwords should be unique within one to several million generated case-sensitive passwords of one series, which is continuously generated with each use of this command until you use the **Reset MCLG Generator Seeds** button. You can use the **Tools > Find Duplicates** command to verify the uniqueness of these passwords. Generated passwords can contain Unicode characters, but non-ASCII characters should be used with care, as some systems (especially various recovery systems) might not support them.



GS-Base Help > *Encrypting field contents*

To password-protect and encrypt only a part of your database, selected record fields, use the **Edit > Encrypt/Decrypt Field Contents** commands.
(If there are no fields to decrypt, the **Decrypt** command is inactive. If there are already encrypted fields in a given table, the **Encrypt** command is inactive.)

In comparison with file encryption, there are several benefits of using such partial encryption, for example:

- you can publish your database for viewing and updating of the unencrypted (e.g. non-sensitive) parts of the database tables by others, and you can be sure that the encrypted table fields can't be changed, misplaced or deleted within the table in any way;
- for any file with partially encrypted data you can still open it without a password to verify the file version, and to see what the table and field names are, etc.

To encrypt the data, GS-Base uses the standard Twofish block cipher algorithm, which guarantees that it can't be broken and your data can't be accessed/read without knowing your password.

GS-Base displays in encrypted fields the actual encrypted data in the MIME64 encoding form.

For encrypted text fields, the displayed encrypted contents length is proportional to the length of the source text strings.
If you would like to protect this information as well, you can add trailing spaces to these source strings before encrypting.
For encrypted numeric fields, the displayed encrypted values have the same length.
For encrypted LongText/Code/Images/Files fields, the entire contents are encrypted and the "encrypted" label is displayed in table/form cells.

The encrypted part of a given database table can't be modified - GS-Base disables any editing actions for the encrypted fields. If you delete records with encrypted data or insert new records before/between records with encrypted data, GS-Base won't let you save such a modified file. Of course, you're free to edit/delete and save any unencrypted fields and append new records (leaving the encrypted fields empty).

If a given table contains calculated fields, after encrypting one or more of its fields the last calculation results become "frozen" until you decrypt these fields.

Field encryption can be applied independently to each table, so you can use a different password for each table.

Having some encrypted fields, you can still use file encryption.

To Specify a Default Value for a Given Text/Numeric Field

Open the Field Setup dialog box (double-clicking that field in the Database Explorer window or using the **Field Setup** command), select the **Special > Default value** option and enter the desired value, which can be:

- For numeric fields: any unformatted number.
- For text fields: any text string.
  If that text field is formatted as "Date", the default value should represent the generic date/time string: YYYY-MM-DD.

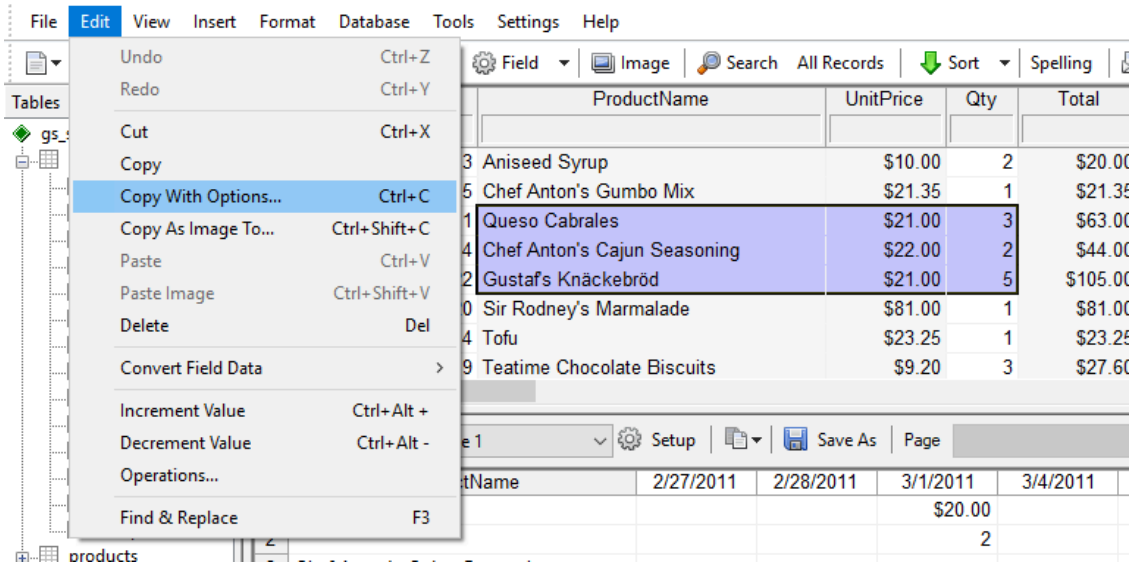
To Insert the Default Field Value

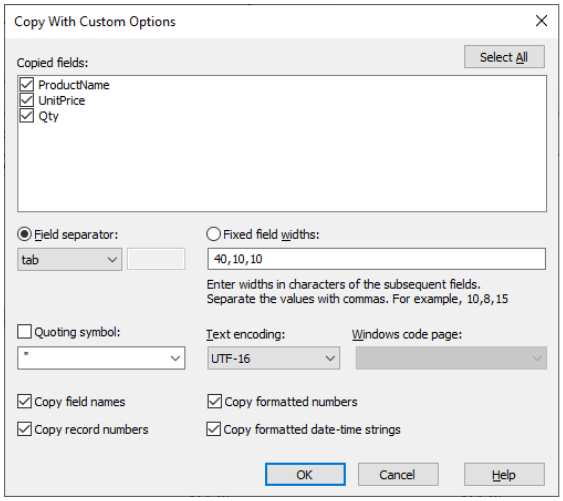
Select a range of fields and/or records and use the **Insert > Default Value (Ctrl+T)** command. Default values are inserted into all empty fields within that selection.

GS-Base Help > Using the Copy With Options command

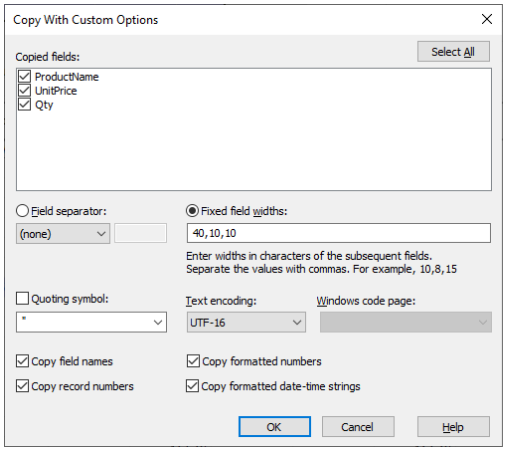
The **Copy With Options** command has been introduced as an enhanced alternative to the standard **Copy** command. It provides flexible control over how data in both tables and forms is copied from GS-Base to external applications. The **Ctrl+C** shortcut can be assigned to either command, allowing users to choose the behavior that best fits their workflow.

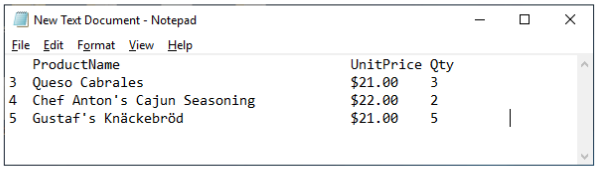
Below are sample screens illustrating different copy configurations, along with examples of how the copied content appears in a spreadsheet and a text editor.

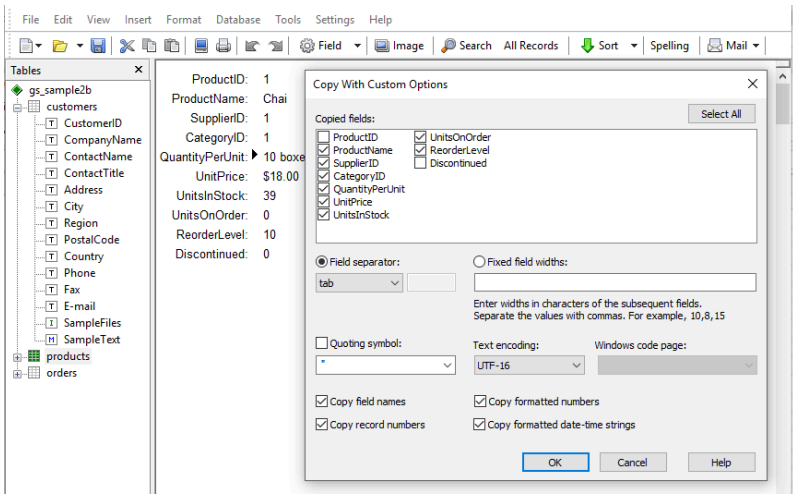




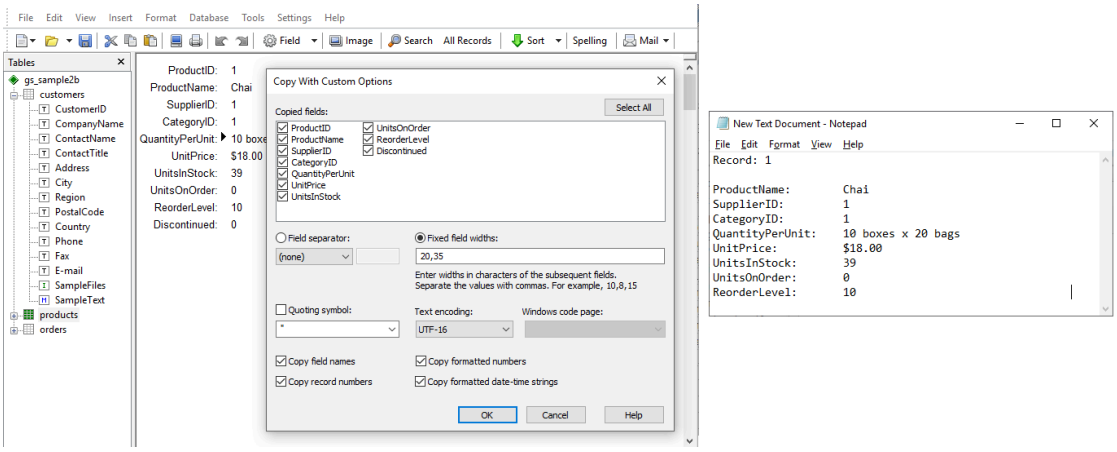
	ProductName	UnitPrice	Qty
3	Queso Cabrales	$21.00	3
4	Chef Anton's Cajun Seasoning	$22.00	2
5	Gustaf's Knäckebröd	$21.00	5







Record:	1
ProductName:	Chai
SupplierID:	1
CategoryID:	1
QuantityPerUnit:	10 boxes x 20 bags
UnitPrice:	$18.00
UnitsInStock:	39
UnitsOnOrder:	0
ReorderLevel:	10



GS-Base Help > *Improving performance when using large files*

- Increase the number of processor cores that GS-Base can use. They are used efficiently when mass-updating calculation fields, updating pivot tables, and calculating automatic table row heights and column widths.

- GS-Base enables you to save all the automatic search and sort indexes created when filtering tables and when recalculating calculated formulas, especially with various inter-table aggregation formulas that use binary searches and require such sort indexes.
  This vastly improves how you use large databases, because if the indexes are not saved:

  (1) opening a database with active filters required waiting until the filtering is performed for the last used filter set,

  (2) after opening a database, performing the first inter-table calculation with various forms of lookup formulas required re-creating the internal sort indexes to enable binary searches. (The displayed "progress" dialog box in the status field first shows "sorting" and much later the "calculating" status.)
  Whenever you are opening a database, both (1) and (2) could increase the actual total opening time by tens of seconds for large (e.g. a few tens of millions of rows or more, and multiple regex filters) tables.

  If you check the two **Save indexes(...)** options in the **Options > Settings** dialog box, both the (1) and (2) steps will be skipped, and the databases will be displayed instantly after the file is read.

Technically, for each table you can save up to 16,384 indexes in three versions each (depending on the sorting/language parameters, like both the case-sensitive and case-insensitive versions), and each one for up to 256 million records. However, the practical limit will be lower due to the RAM/paging limitations, as one index entry (one record/row) uses 4 bytes.

- When entering multiple (several, more than ten, etc.) field filters for very large tables, turn off automatic filtering until you enter all filtering expressions. You can do this by unchecking the **Search Now** check box in the **Field Search Filter** dialog box. In this case, to finally apply all entered filters, use the **Tools > Search All Now** menu command.
  Otherwise, accepting each subsequent filter will trigger repetitive filtering with the expressions you have entered so far.

- If you're performing large data block operations within tables (e.g. transforming/converting entire table columns, pasting entire columns), you may want to limit the **Undo** level in the **Options > Settings** dialog box. Otherwise the **Undo** buffer will hold large amounts of **Undo** data. (Of course, if your amount of RAM is much larger than this, it won't matter.)

- If your database contains a very large number of memo/image/files/code objects (e.g. tens of thousands), you should always use the *.gsb file extension for saved GS-Base native databases, not the *.zip extension, because otherwise Windows will try to index any found zip file, and with thousands of zip substreams saved by GS-Base, the Windows system won't be able to process such structures and might become unresponsive for minutes to hours or more.

GS-Base uses formulas in the following procedures/functions:

- You can enter them directly in fields. They will be evaluated and the results will be saved. This applies to the standard Text and Numeric fields.

- You can use them with the **Insert > Generating formula** command. This command enables you to mass fill a given field in all records based on values of other fields in related records. It can be used for all field types.
  It can be especially useful when combined with the **UDF() Python** functions that

can return to GS-Base any data (graphics or text) of any size to be inserted in the GS-Base binary fields (LongText, Code, Images/Files). You can, for example:

  - use the FIELDS(start, end) and FIELDNAMES(start, end) functions to pass data series (up to the 16K field counter limit) to such Python functions to "mass"-generate charts in Image/Files field(s) in all records;
  - use the FIELDS(start, end) function to archive all current Text/Number field values in the LongText fields;
  - populate any binary fields in all records with files, documents, graphics, reports based on other fields, etc. See: Using Python functions as UDF()

- You can use them to create calculated fields:
  in the **Field Setup** dialog box, from the **Special** (functionality) list choose **Calculation formula** and enter the desired formula in the field below.
  If the menu option **Tools > Automatic Updating** is checked, each editing action within the table will cause automatic recalculation of the formulas for fields from records affected by that editing action.
  For example, if you paste some values to a field for records 10 to 20, all fields in these records with such calculation formulas defined will be updated and the values will be inserted in those fields.
  If the menu option **Tools > Automatic Updating** is unchecked, to update all calculated fields you must use the **Tools > Update All Calculated Fields** command to perform the calculations for the newly entered data. For details, please see: Managing database fields and tables.

- You can use formulas to construct search expressions for fields. For details, please see: Searching / Filtering Records.

- Formulas can be used to define the "Increment/Decrement Operations". For details, please see: Using the Increment, Decrement and Operations commands.

- Formulas can be used when designing printed labels to specify what should be placed on a given label.

- You can use formulas when inserting formula-based series in records. For details, please see: Inserting series.

To Enter/Edit a Formula

Formulas available in GS-Base are similar to those used in spreadsheets, with a few exceptions:

- Cell references are replaced by field names. All field names occurring in a given formula refer to one and the same record.
- The range ":" operator is not available. Formulas requiring array/range parameters can use the "array()" function that creates an array out of database fields, strings, numbers, etc.
- References to other tables via the "!" and "_" operators are not available. Instead, there is a separate category of functions: **Roll-up** functions that group functions accessing data from other database tables in the same database. Some of those functions can be used to create links/relations between tables. (See the included "sample.zip" database and its calculated fields used to link the "products" and "orders" tables.)

For example:

- to sum up two fields 'subtotal' and 'tax', use the following formula:

  =subtotal + tax

- to merge two fields 'name' and 'country', use the following formula:

  =name & ", " & country

- and to obtain a period string representing the difference between two given dates:

  =dateDiff("2011-01-01", "2011-12-31") (which returns "P364D")

- to find the current age for birth dates stored in the "date" field:

  =year(today()) - year(date) - if(month(date) < month(today()), 0, if(month(date) > month(today()), 1, day(date) >= day(today())))

  Note: If the "date" field is not a valid GS-Base date field (that is, a text field containing generic date/time YYYY-MM-DD strings that can be formatted to be displayed in the desired local format), the above "date" argument should be replaced by dateValue(date).

A few other examples:

=Unit_Price*Qty
="abc"
=1 + 2
="a" & 'b' & 3
=fv(0.75%, 36, -500, -5500, 0)
=LProg({2,2;1,2;4,0}, {1;1;1}, {14;8;16}, {2; 4},0,,)
=sum(field1, field2, field3, sum(field4, field5, field6))

Numbers and dates used as arguments cannot be formatted. For example, the following expressions are incorrect:

=$1,000.00 + 1
=dateDiff("6/3/11", "8/9/2011")

unless the description specifically states that parameters are arbitrary text strings to be converted to a specific type, like:

=value("$1,000.00") + 1
=dateDiff(dateValue("6/3/11"), dateValue("8/9/2011"))

Text strings should be delimited by either single or double quotation marks.

Formulas should not contain circular field references.

To browse all available formula categories and examples, please see the Field Setup dialog box or the Search dialog box.

Note: Square brackets [,] in parameter lists denote parameters that are optional and can be omitted. For example:

LProg(A, s, b, c, vector, **[epsilon]**, **[m]**)

However, you still have to use the corresponding parameter list separators, that is:

LProg(A, s, b, c, vector,,)

If your current Windows regional settings define commas as decimal separators, use semicolons (**;**) to separate function arguments, or, if you prefer locale-independent (US) settings, select the generic locale via the **Settings > Locales > Generic** command.

Available operators:

Operator	*Operation*	*Comments*	*Precedence*
=	Equal	Compares numbers or text strings (the comparison is not case-sensitive). Example: A1=4, B2="abc"	6
<	Less than	Compares numbers or text strings (the comparison is not case-sensitive). Example: A1<4, B2<"abc"	6
>	Greater than	Compares numbers or text strings (the comparison is not case-sensitive). Example: A1>4, B2>"abc"	6
<=	Less than or equal	Compares numbers or text strings (the comparison is not case-sensitive). Example: A1<=4, B2<="abc"	6
>=	Greater than or equal	Compares numbers or text strings (the comparison is not case-sensitive). Example: A1>=4, B2>="abc"	6

Operator	*Operation*	*Comments*	*Precedence*
<>	Not equal	Compares numbers or text strings (the comparison is not case-sensitive). Example: A1<>4, B2<>"abc"	6
+	Addition	Adds numbers.	5
-	Subtraction	Subtracts numbers.	5
&	String concatenation	Merges text strings. Example: A1 & "abc", "a" & "b"	5
*****	Multiplication	Multiplies numbers.	4
/	Division	Divides numbers.	4
^	To the power of	Calculates the power of.	3
-	Negative	Changes the sign of a number. Example: -A1	2
%	Percent	Specifies a number entered as a percentage. Example: 12%	1

makeUnique(table, v, field)

Checks whether the value 'v' occurs in the 'field' in the specified 'table' and, if so, creates a copy of it by adding the (n) suffix for text strings, or adding 1 for numbers, until the unique value is found.

'v' can be any text, number or a text field name. The type of 'v' and the type of 'field' must be the same.

```
=makeUnique("products", "tofu", "productnames")    returns
tofu(1)
=makeUnique("products", "tofu(1)", "productnames")    returns
tofu(2)
=makeUnique("products", 10, "productID")    returns 78
```

isUnique(v)

Checks whether the value 'v' (a text string or a number) already occurs in the current field and, if so, returns 1. Otherwise it returns 0.

NOTE: The makeUnique() and isUnique() functions are two special functions that have the following restrictions:
If the 'table' is the same table where the function is used, makeUnique() can be executed only as a single-action field conversion or validation formula, and is inactive for any block operations or recalculation operations.
The isUnique() function is always used in-place, in the same field it refers to, and it can be executed only as a single-action field conversion or validation formula.

GS-Base Help > *Using objectStats() functions to obtain file/image data*

objectStats(field, type)

objectStats(field, imageName, propertyType)

Returns Images/Files field statistics, or the metadata encoded in JPG/TIFF/PNG images (e.g. by digital cameras).

objectStats(field, type)

type = 0 - returns the number of objects
type = 1 - returns the size in bytes of the biggest object in a field
type = 2 - returns the size in bytes of the smallest object in a field
type = 3 - returns 1 if a field contains images (in a format supported by GS-Base), or 0 otherwise.
To obtain the total physical size of the Images/Files field, use the len() function.

objectStats(field, imageName, propertyType)

NOTE: this function should be used in calculated **Text** fields.
imageName - if there are multiple images in a field, you can specify the name of a given image; if you specify an empty string, the first image from the field will be used.
propertyType - a numeric property code; to generate a list of all possible codes for a given image, specify -1.

```
=objectStats(files, 0)
=objectStats(images, 1)
```

```
=objectStats(images, "", hex2dec("132"))
```

Returns the 0x132 property value (a date).

```
=objectStats(images, "some_image.jpg", hex2dec("132"))
```

Returns the 0x132 property value for a given image.

```
=objectStats(images, "", -1)
```

Returns the list of all properties encoded in a given image.

Property values can be read for JPG/TIFF/EXIF/PNG images. A picture taken by a digital camera can contain, for example, the following tags:

Tag	*Code*	*Tag*	*Code*
ImageWidth	`0x100`	ImageHeight	`0x101`
EquipMake	`0x10f`	EquipModel	`0x110`

Tag	*Code*	*Tag*	*Code*
Orientation	`0x112`	XResolution	`0x11a`
YResolution	`0x11b`	ResolutionUnit	`0x128`
SoftwareUsed	`0x131`	DateTime	`0x132`
YCbCrPositioning	`0x213`	ExifExposureTime	`0x829a`
ExifFNumber	`0x829d`	ExifExposureProg	`0x8822`
ExifISOSpeed	`0x8827`	ExifVer	`0x9000`
ExifDTOrig	`0x9003`	ExifDTDigitized	`0x9004`
ExifCompConfig	`0x9101`	ExifShutterSpeed	`0x9201`
ExifAperture	`0x9202`	ExifBrightness	`0x9203`
ExifExposureBias	`0x9204`	ExifMaxAperture	`0x9205`
ExifSubjectDist	`0x9206`	ExifMeteringMode	`0x9207`
ExifFlash	`0x9209`	ExifFocalLength	`0x920a`
ExifMakerNote	`0x927c`	ExifDTSubsec	`0x9290`
ExifDTOrigSS	`0x9291`	ExifDTDigSS	`0x9292`
ExifFPXVer	`0xa000`	ExifColorSpace	`0xa001`
ExifPixXDim	`0xa002`	ExifPixYDim	`0xa003`
ExifSensingMethod	`0xa217`	ExifSceneType	`0xa301`
GpsVer	`0x0`	GpsLatitudeRef	`0x1`
GpsLatitude	`0x2`	GpsLongitudeRef	`0x3`
GpsLongitude	`0x4`	GpsAltitudeRef	`0x5`
GpsAltitude	`0x6`	GpsGpsTime	`0x7`
GpsGpsDop	`0xb`	ThumbnailData	`0x501b`
ThumbnailImageWidth	`0x5020`	ThumbnailImageHeight	`0x5021`

Tag	*Code*	*Tag*	*Code*
ThumbnailCompression	`0x5023`	ThumbnailOrientation	`0x5029`
ThumbnailResolutionX	`0x502d`	ThumbnailResolutionY	`0x502e`
ThumbnailResolutionUnit	`0x5030`	JPEGInterFormat	`0x201`
JPEGInterLength	`0x202`	ChrominanceTable	`0x5091`
LuminanceTable	`0x5090`	ICCProfile	`0x8773`

For the full list of possible tags, please use any internet search engine and search for the official specification of image/picture metadata tags.

GS-Base Help > Using textStats() functions to obtain LongText and Code field statistics

textStats(field, type)

Returns LongText or Code field statistics:

1. type = 0 - returns the number of words.
2. type = 1 - returns the number of letters and numbers.
3. type = 2 - returns the number of letters, numbers and spaces.
4. type = 3 - returns the number of characters as returned by type 2, plus punctuation characters and separators.
5. type = 4 - returns the number of newline characters.

NOTE: this function should be used in calculated **Text** fields.
To obtain the total physical size of the LongText or Code field, use the len() function.

```
=textStats(memo, 0)
=textStats(description, 3)
```

GS-Base Help > Using fileStats() functions to obtain EXIF tags and metadata from multimedia files

fileStats(filePath, type)

This function returns:

1. EXIF tags for files representing photos (saved by digital cameras) and any other JPG/TIFF/PNG/BMP/ICO images.
2. Various multimedia-related metadata for multimedia (video/sound) files.

filePath

A textual field containing full file paths (or a directly entered fixed string). File paths from a given folder or entire disk partitions can be loaded with the "Tools > Verify Files" functions.

type

A numeric code representing one of two types of extracted tags, depending on the file type:
if the path points to a photo or other JPG/TIFF/PNG/BMP/ICO image, this code represents an extracted EXIF tag;
if the path points to a multimedia video or sound file, this code represents multimedia file metadata.

NOTE: this function should be used in calculated **Text** fields.
For best performance for very large tables that you update at once, increase the number of processor cores used for calculations in the **Settings > Options > General** window.

```
=fileStats(images, 1)    returns the GPS latitude for the location
where the image was taken
=fileStats("e:\photo3.jpg", 6)    returns the GPS altitude for the
location where the image was taken
=fileStats(images, hex2dec("132"))    returns the 0x132 property
value (a date) for a given image
```

```
=fileStats(images, -1)    returns the list of all EXIF tags
encoded in a given image
```

```
=fileStats(videos, 3)
=fileStats("e:\some_file.avi", 3)
=fileStats(videos, 20)
=fileStats(videos, 11)
```

EXIF Property Codes

EXIF property value (hexadecimal) codes for photos and other JPG/TIFF/EXIF/PNG/BMP/ICO images:

Tag	*Code*	*Tag*	*Code*
ImageWidth	`0x100`	ImageHeight	`0x101`
EquipMake	`0x10f`	EquipModel	`0x110`
Orientation	`0x112`	XResolution	`0x11a`
YResolution	`0x11b`	ResolutionUnit	`0x128`
SoftwareUsed	`0x131`	DateTime	`0x132`
YCbCrPositioning	`0x213`	ExifExposureTime	`0x829a`
ExifFNumber	`0x829d`	ExifExposureProg	`0x8822`
ExifISOSpeed	`0x8827`	ExifVer	`0x9000`
ExifDTOrig	`0x9003`	ExifDTDigitized	`0x9004`
ExifCompConfig	`0x9101`	ExifShutterSpeed	`0x9201`
ExifAperture	`0x9202`	ExifBrightness	`0x9203`
ExifExposureBias	`0x9204`	ExifMaxAperture	`0x9205`
ExifSubjectDist	`0x9206`	ExifMeteringMode	`0x9207`
ExifFlash	`0x9209`	ExifFocalLength	`0x920a`
ExifMakerNote	`0x927c`	ExifDTSubsec	`0x9290`

Tag	*Code*	*Tag*	*Code*
ExifDTOrigSS	`0x9291`	ExifDTDigSS	`0x9292`
ExifFPXVer	`0xa000`	ExifColorSpace	`0xa001`
ExifPixXDim	`0xa002`	ExifPixYDim	`0xa003`
ExifSensingMethod	`0xa217`	ExifSceneType	`0xa301`
GpsVer	`0x0`	GpsLatitudeRef	`0x1`
GpsLatitude	`0x2`	GpsLongitudeRef	`0x3`
GpsLongitude	`0x4`	GpsAltitudeRef	`0x5`
GpsAltitude	`0x6`	GpsGpsTime	`0x7`
GpsGpsDop	`0xb`	ThumbnailData	`0x501b`
ThumbnailImageWidth	`0x5020`	ThumbnailImageHeight	`0x5021`
ThumbnailCompression	`0x5023`	ThumbnailOrientation	`0x5029`
ThumbnailResolutionX	`0x502d`	ThumbnailResolutionY	`0x502e`
ThumbnailResolutionUnit	`0x5030`	JPEGInterFormat	`0x201`
JPEGInterLength	`0x202`	ChrominanceTable	`0x5091`
LuminanceTable	`0x5090`	ICCProfile	`0x8773`

Multimedia Metadata Codes

Multimedia metadata tag codes for video/sound files:

Code	*Description*
1	Digital Living Network Alliance(DLNA) profile identifier.

Code	*Description*
2	Number of audio channels.
3	Average audio bit rate, in bits per second.
4	Audio subtype.
5	Indicates whether the audio stream uses variable bit-rate encoding.
6	Peak volume level of audio content.
7	Audio sample rate in samples per second.
8	Number of bits per audio sample.
9	Identifier of the audio stream.
10	Author.
11	A comment attached to a file.
12	Copyright information.
13	Indicates whether the content is protected using digital rights management.
14	Keywords.
15	Language.
16	URL of the author's website.
17	Average volume level of audio content.
18	The string representation of a GUID that identifies the primary class of media.
19	The string representation of a GUID that identifies the secondary class of media.
20	The string representation of a GUID that identifies the collection group.

Code	*Description*
21	The string representation of a GUID that identifies the collection.
22	Distributor of the content.
23	The string representation of a GUID that identifies the collection.
24	Time when the content was encoded.
25	Original release date.
26	Duration, in 100-nanosecond units.
27	Digital video disc identifier.
28	Name of the person or group that encoded the content.
29	Description of the settings used to encode the content.
30	Music CD identifier. This value is used to identify a CD.
31	Name of the metadata content provider.
32	Name of the producer of the content.
33	URL of a website offering a promotion related to the content.
34	Rating of the content as assigned by the metadata content provider.
35	Style or genre of the content as assigned by the metadata content provider.
36	Publisher.
37	Subtitle.
38	A generic string that can be used to identify the file.
39	Writer.

Code	*Description*
40	Year the content was published.
41	Primary artist for the album.
42	Album title.
43	Artist.
44	Beats per minute.
45	Composer.
46	Conductor.
47	Description of the content (boxed set or series).
48	Genre.
49	The initial key of the music.
50	Indicates whether the music file is part of a compilation.
51	Lyrics.
52	Mood.
53	The part number and the total number of parts.
54	Period.
55	Track number.
56	Parental rating.
57	Reasons for the assigned parental rating.
58	User rating.
59	Thumbnail image.
60	Title.
61	Video subtype.
62	Director.

Code	*Description*
63	Average video bit rate, in bits per second.
64	The FOURCC of the video encoding format.
65	Video frame height.
66	Video frame rate, expressed as frames per second × 1000.
67	Video frame width.
68	The horizontal component of the pixel aspect ratio.
69	Indicates whether the video stream contains stereo video content.
70	Identifier of the video stream.
71	Total data rate for all video and audio streams, in bits per second.

The functions listed below are a separate category of functions listed in the **Field Setup** dialog box. They are useful for performing mass calculated-field updates in a given table, based on its relation to other tables and to filtered records from other tables. For best performance when updating calculated fields in tables that both contain a large number of records at the same time, use the functions below that use binary searching (which performs automatic background sorting and indexing). You can also increase the number of used processor cores in the **Settings > Options** dialog box.

Function Index

sum_ex min_ex max_ex quartile1_ex median_ex quartile3_ex mode_ex
count_ex sumIfs_ex minIfs_ex maxIfs_ex quartile1Ifs_ex medianIfs_ex
quartile3Ifs_ex modeIfs_ex countIfs_ex vLookUp_ex

Roll-up Calculation Functions

sum_ex(table, v, searchField, calcField, options, [emptyValue])

Searches the 'searchField' field of the specified 'table' for the 'v' value and returns the sum of the 'calcField' field values for all found records. If 'v' is a name (text string) specified without quotation marks, it's assumed to be a name of the field from the table where the formula is used. The 'table', 'searchField' and 'calcField' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

NOTE: if the fast binary searching is used, the type of the 'v' parameter must match the type of the searched field. They must be both either textual or numeric. Otherwise an error is returned.

The 'options' argument is a sum of the following values:

- 0 - the 'table' will be searched for 'v' using the fastest binary searches; case-insensitive. This is the recommended option if the source record set contains a large number of records with the 'v' values and the external 'table' has many records to search as well.
- 1 - performs case-sensitive searching.
- 2 - performs string sorting/comparison (hyphen and apostrophe are sorted, as opposed to the default word-sorting).
- 4 - the 'v' value is a regular expression; instead of the fast binary exact searches, slower sequential searching will be used.
- 8 - allows empty matches when using regular expressions.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

```
=sum_ex("products", ProductID, "ProductID", "ProductName", 0,
0)
=sum_ex("folder1/products", ProductID, "ProductID",
"UnitPrice", 4, -1)
```

---

min_ex(table, v, searchField, calcField, options, [emptyValue])

Searches the 'searchField' field of the specified 'table' for the 'v' value and returns the minimum of the 'calcField' field values for all found records. If 'v' is a name (text string) specified without quotation marks, it's assumed to be a name of the field from the table where the formula is used. The 'table', 'searchField' and 'calcField' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

NOTE: if the fast binary searching is used, the type of the 'v' parameter must match the type of the searched field. They must be both either textual or numeric. Otherwise an error is returned.

The 'options' argument is a sum of the following values:

- 0 - the 'table' will be searched for 'v' using the fastest binary searches; case-insensitive. This is the recommended option if the source record set contains a large number of records with the 'v' values and the external 'table' has many records to search as well.
- 1 - performs case-sensitive searching.
- 2 - performs string sorting/comparison (hyphen and apostrophe are sorted, as opposed to the default word-sorting).
- 4 - the 'v' value is a regular expression; instead of the fast binary exact searches, slower sequential searching will be used.
- 8 - allows empty matches when using regular expressions.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

```
=min_ex("products", ProductID, "ProductID", "ProductName", 0,
0)
```

```
=min_ex("folder1/products", ProductID, "ProductID",
"UnitPrice", 4, -1)
```

---

max_ex(table, v, searchField, calcField, options, [emptyValue])

Searches the 'searchField' field of the specified 'table' for the 'v' value and returns the maximum of the 'calcField' field values for all found records. If 'v' is a name (text string) specified without quotation marks, it's assumed to be a name of the field from the table where the formula is used. The 'table', 'searchField' and 'calcField' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

NOTE: if the fast binary searching is used, the type of the 'v' parameter must match the type of the searched field. They must be both either textual or numeric. Otherwise an error is returned.

The 'options' argument is a sum of the following values:

- 0 - the 'table' will be searched for 'v' using the fastest binary searches; case-insensitive. This is the recommended option if the source record set contains a large number of records with the 'v' values and the external 'table' has many records to search as well.
- 1 - performs case-sensitive searching.
- 2 - performs string sorting/comparison (hyphen and apostrophe are sorted, as opposed to the default word-sorting).
- 4 - the 'v' value is a regular expression; instead of the fast binary exact searches, slower sequential searching will be used.
- 8 - allows empty matches when using regular expressions.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

```
=max_ex("products", ProductID, "ProductID", "ProductName", 0,
0)
=max_ex("folder1/products", ProductID, "ProductID",
"UnitPrice", 4, -1)
```

---

quartile1_ex(table, v, searchField, calcField, options, [emptyValue])

Searches the 'searchField' field of the specified 'table' for the 'v' value and returns the 1st quartile of the 'calcField' field values for all found records. If 'v' is a name (text string) specified without quotation marks, it's assumed to be a name of the field from the table where the formula is used. The 'table', 'searchField' and 'calcField' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

NOTE: if the fast binary searching is used, the type of the 'v' parameter must match the type of the searched field. They must be both either textual or numeric. Otherwise an error is returned.

The 'options' argument is a sum of the following values:

- 0 - the 'table' will be searched for 'v' using the fastest binary searches; case-insensitive. This is the recommended option if the source record set contains a large number of records with the 'v' values and the external 'table' has many records to search as well.
- 1 - performs case-sensitive searching.
- 2 - performs string sorting/comparison (hyphen and apostrophe are sorted, as opposed to the default word-sorting).
- 4 - the 'v' value is a regular expression; instead of the fast binary exact searches, slower sequential searching will be used.
- 8 - allows empty matches when using regular expressions.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

```
=quartile1_ex("products", ProductID, "ProductID",
"ProductName", 0, 0)
=quartile1_ex("folder1/products", ProductID, "ProductID",
"UnitPrice", 4, -1)
```

---

median_ex(table, v, searchField, calcField, options, [emptyValue])

Searches the 'searchField' field of the specified 'table' for the 'v' value and returns the median of the 'calcField' field values for all found records. If 'v' is a name (text string) specified without quotation marks, it's assumed to be a name of the field from the table where the formula is used. The 'table', 'searchField' and 'calcField' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

NOTE: if the fast binary searching is used, the type of the 'v' parameter must match the type of the searched field. They must be both either textual or numeric. Otherwise an error is returned.

The 'options' argument is a sum of the following values:

- 0 - the 'table' will be searched for 'v' using the fastest binary searches; case-insensitive. This is the recommended option if the source record set contains a large number of records with the 'v' values and the external 'table' has many records to search as well.
- 1 - performs case-sensitive searching.
- 2 - performs string sorting/comparison (hyphen and apostrophe are sorted, as opposed to the default word-sorting).
- 4 - the 'v' value is a regular expression; instead of the fast binary exact searches, slower sequential searching will be used.
- 8 - allows empty matches when using regular expressions.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

```
=median_ex("products", ProductID, "ProductID", "ProductName",
0, 0)
=median_ex("folder1/products", ProductID, "ProductID",
"UnitPrice", 4, -1)
```

---

quartile3_ex(table, v, searchField, calcField, options, [emptyValue])

Searches the 'searchField' field of the specified 'table' for the 'v' value and returns the 3rd quartile of the 'calcField' field values for all found records. If 'v' is a name (text string) specified without quotation marks, it's assumed to be a name of the

field from the table where the formula is used. The 'table', 'searchField' and 'calcField' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

NOTE: if the fast binary searching is used, the type of the 'v' parameter must match the type of the searched field. They must be both either textual or numeric. Otherwise an error is returned.

The 'options' argument is a sum of the following values:

- 0 - the 'table' will be searched for 'v' using the fastest binary searches; case-insensitive. This is the recommended option if the source record set contains a large number of records with the 'v' values and the external 'table' has many records to search as well.
- 1 - performs case-sensitive searching.
- 2 - performs string sorting/comparison (hyphen and apostrophe are sorted, as opposed to the default word-sorting).
- 4 - the 'v' value is a regular expression; instead of the fast binary exact searches, slower sequential searching will be used.
- 8 - allows empty matches when using regular expressions.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

```
=quartile3_ex("products", ProductID, "ProductID",
"ProductName", 0, 0)
=quartile3_ex("folder1/products", ProductID, "ProductID",
"UnitPrice", 4, -1)
```

---

mode_ex(table, v, searchField, calcField, options, [emptyValue])

Searches the 'searchField' field of the specified 'table' for the 'v' value and returns the most frequently occurring value of the 'calcField' field values for all found records. If 'v' is a name (text string) specified without quotation marks, it's assumed to be a name of the field from the table where the formula is used. The 'table', 'searchField' and 'calcField' parameters must be placed in quotation marks. If

'table' is located in a nested folder, its full path must be specified with the inner slashes.

NOTE: if the fast binary searching is used, the type of the 'v' parameter must match the type of the searched field. They must be both either textual or numeric. Otherwise an error is returned.

The 'options' argument is a sum of the following values:

- 0 - the 'table' will be searched for 'v' using the fastest binary searches; case-insensitive. This is the recommended option if the source record set contains a large number of records with the 'v' values and the external 'table' has many records to search as well.
- 1 - performs case-sensitive searching.
- 2 - performs string sorting/comparison (hyphen and apostrophe are sorted, as opposed to the default word-sorting).
- 4 - the 'v' value is a regular expression; instead of the fast binary exact searches, slower sequential searching will be used.
- 8 - allows empty matches when using regular expressions.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

```
=mode_ex("products", ProductID, "ProductID", "ProductName", 0,
0)
=mode_ex("folder1/products", ProductID, "ProductID",
"UnitPrice", 4, -1)
```

---

count_ex(table, v, searchField, options, [emptyValue])

Searches the 'searchField' field of the specified 'table' for the 'v' value and returns the number of found records. If 'v' is a name (text string) specified without quotation marks, it's assumed to be a name of the field from the table where the formula is used. The 'table' and 'searchField' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

NOTE: if the fast binary searching is used, the type of the 'v' parameter must match the type of the searched field. They must be both either textual or numeric. Otherwise an error is returned.

The 'options' argument is a sum of the following values:

- 0 - the 'table' will be searched for 'v' using the fastest binary searches; case-insensitive. This is the recommended option if the source record set contains a large number of records with the 'v' values and the external 'table' has many records to search as well.
- 1 - performs case-sensitive searching.
- 2 - performs string sorting/comparison (hyphen and apostrophe are sorted, as opposed to the default word-sorting).
- 4 - the 'v' value is a regular expression; instead of the fast binary exact searches, slower sequential searching will be used.
- 8 - allows empty matches when using regular expressions.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

```
=count_ex("products", ProductID, "ProductID", 0, 0)
=count_ex("folder1/products", ProductID, "ProductID", 4, -1)
```

---

sumIfs_ex(table, calcField, field1, criteria1 [, field2, criteria2, ...], [emptyValue])

Returns the sum of the 'calcField' field values from the specified 'table' for records that meet the specified list of filters/criteria.

The 'table', 'calcField' and 'field(n)' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

References to field values/names in the same table where the formula resides must be used without any quotation marks.

The criteria can be one of the following:

1. a name of the field in the current table, optionally combined with the =, >, >=, <, <= operators.
2. a text string beginning with the =, >, >=, <, <= operators. Numeric and date/time searches require unformatted numbers and generic date/time strings (YYYY-MM-DD).
3. a number or a search pattern: a text string optionally containing the special characters '?' (any character) or '*' (any string). To search for a literal ? or * character, place a tilde (~) before it. Numbers and dates must be used in the unformatted form.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

NOTE: If some 'criteria' can contain pattern characters (*?~) but you don't want to perform pattern searches, use the '&' operator to prefix that criteria with '='. For example: `"=" & CustomerID`

NOTE: Unlike the sum_ex() function, the sumIfs_ex() function always performs slower sequential searching. If both the source and target tables contain hundreds of thousands or millions of records at the same time, it might be too slow. You can improve its performance by including the leading "=" before each criteria (to avoid pattern matching) and by increasing the number of used processor cores in the Settings > Options dialog box.

```
=sumIfs_ex("orders", "Total", "id", CustomerID, 0)
```

Returns the total sales amount for a given customer. 'orders' is a table with all orders, 'id' and 'Total' are its fields containing customers' IDs and total values for each order, respectively, and 'CustomerID' is a field of the table where the result is placed.

```
=sumIfs_ex("folder1/orders", "Total", "id", "=" & CustomerID,)
```

Returns the total sales amount for a given customer; the CustomerID may contain pattern characters, but no pattern search will be performed.

```
=sumIfs_ex("orders", "Qty", "ProductName", "Tofu",
"ProductName", "Chocolade", 0)
```

```
=sumIfs_ex("folder1/folder2/orders", "Total", "Date", ">2011-
01-01", "Date", "<2011-07-01", 0)
```

---

minIfs_ex(table, calcField, field1, criteria1 [, field2, criteria2, ...], [emptyValue])

Returns the minimum of the 'calcField' field values from the specified 'table' for records that meet the specified list of filters/criteria.

The 'table', 'calcField' and 'field(n)' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

References to field values/names in the same table where the formula resides must be used without any quotation marks.

The criteria can be one of the following:

1. a name of the field in the current table, optionally combined with the =, >, >=, <, <= operators.
2. a text string beginning with the =, >, >=, <, <= operators. Numeric and date/time searches require unformatted numbers and generic date/time strings (YYYY-MM-DD).
3. a number or a search pattern: a text string optionally containing the special characters '?' (any character) or '*' (any string). To search for a literal ? or * character, place a tilde (~) before it. Numbers and dates must be used in the unformatted form.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

NOTE: If some 'criteria' can contain pattern characters (*?~) but you don't want to perform pattern searches, use the '&' operator to prefix that criteria with '='. For example: `"=" & CustomerID`

NOTE: Unlike the min_ex() function, the minIfs_ex() function always performs slower sequential searching. If both the source and target tables contain hundreds of thousands or millions of records at the same time, it might be

too slow. You can improve its performance by including the leading "=" before each criteria (to avoid pattern matching) and by increasing the number of used processor cores in the Settings > Options dialog box.

```
=minIfs_ex("orders", "Total", "id", CustomerID, 0)
```

Returns the minimum sale amount for a given customer. 'orders' is a table with all orders, 'id' and 'Total' are its fields containing customers' IDs and total values for each order, respectively, and 'CustomerID' is a field of the table where the result is placed.

```
=minIfs_ex("folder1/orders", "Total", "id", "=" & CustomerID,
-1)
```

Returns the minimum sale amount for a given customer; the CustomerID may contain pattern characters, but no pattern search will be performed.

```
=minIfs_ex("orders", "Qty", "ProductName", "Tofu",
"ProductName", "Chocolade", -1)
=minIfs_ex("folder1/folder2/orders", "Total", "Date", ">2011-
01-01", "Date", "<2011-07-01", 0)
```

---

maxIfs_ex(table, calcField, field1, criteria1 [, field2, criteria2, ...], [emptyValue])

Returns the maximum of the 'calcField' field values from the specified 'table' for records that meet the specified list of filters/criteria.

The 'table', 'calcField' and 'field(n)' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

References to field values/names in the same table where the formula resides must be used without any quotation marks.

The criteria can be one of the following:

1. a name of the field in the current table, optionally combined with the =, >, >=, <, <= operators.

2. a text string beginning with the =, >, >=, <, <= operators. Numeric and date/time searches require unformatted numbers and generic date/time strings (YYYY-MM-DD).
3. a number or a search pattern: a text string optionally containing the special characters '?' (any character) or '*' (any string). To search for a literal ? or * character, place a tilde (~) before it. Numbers and dates must be used in the unformatted form.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

NOTE: If some 'criteria' can contain pattern characters (*?~) but you don't want to perform pattern searches, use the '&' operator to prefix that criteria with '='. For example: `"=" & CustomerID`

NOTE: Unlike the max_ex() function, the maxIfs_ex() function always performs slower sequential searching. If both the source and target tables contain hundreds of thousands or millions of records at the same time, it might be too slow. You can improve its performance by including the leading "=" before each criteria (to avoid pattern matching) and by increasing the number of used processor cores in the Settings > Options dialog box.

```
=maxIfs_ex("orders", "Total", "id", CustomerID, -1)
```

Returns the maximum sale amount for a given customer. 'orders' is a table with all orders, 'id' and 'Total' are its fields containing customers' IDs and total values for each order, respectively, and 'CustomerID' is a field of the table where the result is placed.

```
=maxIfs_ex("folder1/orders", "Total", "id", "=" & CustomerID,
-1)
```

Returns the maximum sale amount for a given customer; the CustomerID may contain pattern characters, but no pattern search will be performed.

```
=maxIfs_ex("orders", "Qty", "ProductName", "Tofu",
"ProductName", "Chocolade", 0)
=maxIfs_ex("folder1/folder2/orders", "Total", "Date", ">2011-
01-01", "Date", "<2011-07-01", 0)
```

quartile1lfs_ex(table, calcField, field1, criteria1 [, field2, criteria2, ...], [emptyValue])

Returns the first quartile of the 'calcField' field values from the specified 'table' for records that meet the specified list of filters/criteria.

The 'table', 'calcField' and 'field(n)' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

References to field values/names in the same table where the formula resides must be used without any quotation marks.

The criteria can be one of the following:

1. a name of the field in the current table, optionally combined with the =, >, >=, <, <= operators.
2. a text string beginning with the =, >, >=, <, <= operators. Numeric and date/time searches require unformatted numbers and generic date/time strings (YYYY-MM-DD).
3. a number or a search pattern: a text string optionally containing the special characters '?' (any character) or '*' (any string). To search for a literal ? or * character, place a tilde (~) before it. Numbers and dates must be used in the unformatted form.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

NOTE: If some 'criteria' can contain pattern characters (*?~) but you don't want to perform pattern searches, use the '&' operator to prefix that criteria with '='. For example: `"=" & CustomerID`

NOTE: Unlike the quartile1_ex() function, the quartile1lfs_ex() function always performs slower sequential searching. If both the source and target tables contain hundreds of thousands or millions of records at the same time, it might be too slow. You can improve its performance by including the leading "=" before each criteria (to avoid pattern matching) and by increasing the number of used processor cores in the Settings > Options dialog box.

```
=quartile1Ifs_ex("orders", "Total", "id", CustomerID, -1)
```

Returns the 1st quartile of the sales amounts for a given customer. 'orders' is a table with all orders, 'id' and 'Total' are its fields containing customers' IDs and total values for each order, respectively, and 'CustomerID' is a field of the table where the result is placed.

```
=quartile1Ifs_ex("folder1/orders", "Total", "id", "=" &
CustomerID,)
```

Returns the 1st quartile of the sales amounts for a given customer; the CustomerID may contain pattern characters, but no pattern search will be performed.

```
=quartile1Ifs_ex("orders", "Qty", "ProductName", "Tofu",
"ProductName", "Chocolade",)
=quartile1Ifs_ex("folder1/folder2/orders", "Total", "Date",
">2011-01-01", "Date", "<2011-07-01", 0)
```

---

medianIfs_ex(table, calcField, field1, criteria1 [, field2, criteria2, ...], [emptyValue])

Returns the median of the 'calcField' field values from the specified 'table' for records that meet the specified list of filters/criteria.

The 'table', 'calcField' and 'field(n)' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

References to field values/names in the same table where the formula resides must be used without any quotation marks.

The criteria can be one of the following:

1. a name of the field in the current table, optionally combined with the =, >, >=, <, <= operators.
2. a text string beginning with the =, >, >=, <, <= operators. Numeric and date/time searches require unformatted numbers and generic date/time strings (YYYY-MM-DD).

3. a number or a search pattern: a text string optionally containing the special characters '?' (any character) or '*' (any string). To search for a literal ? or * character, place a tilde (~) before it. Numbers and dates must be used in the unformatted form.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

NOTE: If some 'criteria' can contain pattern characters (*?~) but you don't want to perform pattern searches, use the '&' operator to prefix that criteria with '='. For example: `"=" & CustomerID`

NOTE: Unlike the median_ex() function, the medianIfs_ex() function always performs slower sequential searching. If both the source and target tables contain hundreds of thousands or millions of records at the same time, it might be too slow. You can improve its performance by including the leading "=" before each criteria (to avoid pattern matching) and by increasing the number of used processor cores in the Settings > Options dialog box.

```
=medianIfs_ex("orders", "Total", "id", CustomerID, 0)
```

Returns the median of the sales amounts for a given customer. 'orders' is a table with all orders, 'id' and 'Total' are its fields containing customers' IDs and total values for each order, respectively, and 'CustomerID' is a field of the table where the result is placed.

```
=medianIfs_ex("folder1/orders", "Total", "id", "=" &
CustomerID,)
```

Returns the median of the sales amounts for a given customer; the CustomerID may contain pattern characters, but no pattern search will be performed.

```
=medianIfs_ex("orders", "Qty", "ProductName", "Tofu",
"ProductName", "Chocolade",)
=medianIfs_ex("folder1/folder2/orders", "Total", "Date",
">2011-01-01", "Date", "<2011-07-01",)
```

quartile3Ifs_ex(table, calcField, field1, criteria1 [, field2, criteria2, ...], [emptyValue])

Returns the third quartile of the 'calcField' field values from the specified 'table' for records that meet the specified list of filters/criteria.

The 'table', 'calcField' and 'field(n)' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

References to field values/names in the same table where the formula resides must be used without any quotation marks.

The criteria can be one of the following:

1. a name of the field in the current table, optionally combined with the =, >, >=, <, <= operators.
2. a text string beginning with the =, >, >=, <, <= operators. Numeric and date/time searches require unformatted numbers and generic date/time strings (YYYY-MM-DD).
3. a number or a search pattern: a text string optionally containing the special characters '?' (any character) or '*' (any string). To search for a literal ? or * character, place a tilde (~) before it. Numbers and dates must be used in the unformatted form.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

NOTE: If some 'criteria' can contain pattern characters (*?~) but you don't want to perform pattern searches, use the '&' operator to prefix that criteria with '='. For example: `"=" & CustomerID`

NOTE: Unlike the quartile3_ex() function, the quartile3Ifs_ex() function always performs slower sequential searching. If both the source and target tables contain hundreds of thousands or millions of records at the same time, it might be too slow. You can improve its performance by including the leading "=" before each criteria (to avoid pattern matching) and by increasing the number of used processor cores in the Settings > Options dialog box.

```
=quartile3Ifs_ex("orders", "Total", "id", CustomerID, 0)
```

Returns the 3rd quartile of the sales amounts for a given customer. 'orders' is a table with all orders, 'id' and 'Total' are its fields containing customers' IDs and total values for each order, respectively, and 'CustomerID' is a field of the table where the result is placed.

```
=quartile3Ifs_ex("folder1/orders", "Total", "id", "=" &
CustomerID,)
```

Returns the 3rd quartile of the sales amounts for a given customer; the CustomerID may contain pattern characters, but no pattern search will be performed.

```
=quartile3Ifs_ex("orders", "Qty", "ProductName", "Tofu",
"ProductName", "Chocolade",)
=quartile3Ifs_ex("folder1/folder2/orders", "Total", "Date",
">2011-01-01", "Date", "<2011-07-01",)
```

---

modeIfs_ex(table, calcField, field1, criteria1 [, field2, criteria2, ...], [emptyValue])

Returns the most frequently occurring value of the 'calcField' field values from the specified 'table' for records that meet the specified list of filters/criteria.

The 'table', 'calcField' and 'field(n)' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

References to field values/names in the same table where the formula resides must be used without any quotation marks.

The criteria can be one of the following:

1. a name of the field in the current table, optionally combined with the =, >, >=, <, <= operators.
2. a text string beginning with the =, >, >=, <, <= operators. Numeric and date/time searches require unformatted numbers and generic date/time strings (YYYY-MM-DD).
3. a number or a search pattern: a text string optionally containing the special characters '?' (any character) or '*' (any string). To search for a literal ? or *

character, place a tilde (~) before it. Numbers and dates must be used in the unformatted form.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

NOTE: If some 'criteria' can contain pattern characters (*?~) but you don't want to perform pattern searches, use the '&' operator to prefix that criteria with '='. For example: `"=" & CustomerID`

NOTE: Unlike the mode_ex() function, the modeIfs_ex() function always performs slower sequential searching. If both the source and target tables contain hundreds of thousands or millions of records at the same time, it might be too slow. You can improve its performance by including the leading "=" before each criteria (to avoid pattern matching) and by increasing the number of used processor cores in the Settings > Options dialog box.

```
=modeIfs_ex("orders", "Total", "id", CustomerID, 1)
```

Returns the most frequently occurring sale amount for a given customer. 'orders' is a table with all orders, 'id' and 'Total' are its fields containing customers' IDs and total values for each order, respectively, and 'CustomerID' is a field of the table where the result is placed.

```
=modeIfs_ex("folder1/orders", "Total", "id", "=" & CustomerID,
-1)
```

Returns the most frequently occurring sale amount for a given customer; the CustomerID may contain pattern characters, but no pattern search will be performed.

```
=modeIfs_ex("orders", "Qty", "ProductName", "Tofu",
"ProductName", "Chocolade", 0)
=modeIfs_ex("folder1/folder2/orders", "Total", "Date", ">2011-
01-01", "Date", "<2011-07-01", 0)
```

---

countIfs_ex(table, field1, criteria1 [, field2, criteria2, ...], [emptyValue])

Counts records in the specified 'table' that meet the specified list of filters/criteria.

The 'table' and 'field(n)' parameters must be placed in quotation marks. If 'table' is located in a nested folder, its full path must be specified with the inner slashes.

References to field values/names in the same table where the formula resides must be used without any quotation marks.

The criteria can be one of the following:

1. a name of the field in the current table, optionally combined with the =, >, >=, <, <= operators.
2. a text string beginning with the =, >, >=, <, <= operators. Numeric and date/time searches require unformatted numbers and generic date/time strings (YYYY-MM-DD).
3. a number or a search pattern: a text string optionally containing the special characters '?' (any character) or '*' (any string). To search for a literal ? or * character, place a tilde (~) before it. Numbers and dates must be used in the unformatted form.

The 'emptyValue' argument specifies what should be returned (a number or a text string) if no 'v' values are found. If it's left empty, the function returns the #NULL! error code.

NOTE: If some 'criteria' can contain pattern characters (*?~) but you don't want to perform pattern searches, use the '&' operator to prefix that criteria with '='. For example: `"=" & CustomerID`

NOTE: Unlike the count_ex() function, the countIfs_ex() function always performs slower sequential searching. If both the source and target tables contain hundreds of thousands or millions of records at the same time, it might be too slow. You can improve its performance by including the leading "=" before each criteria (to avoid pattern matching) and by increasing the number of used processor cores in the Settings > Options dialog box.

```
=countIfs_ex("orders", "id", CustomerID, 0)
```

Returns the number of orders for a given customer. 'orders' is a table with all orders, 'id' and 'Total' are its fields containing customers' IDs and total values for each order, respectively, and 'CustomerID' is a field of the table where the result is placed.

```
=countIfs_ex("folder1/orders", "id", "=" & CustomerID, 0)
```

Returns the number of orders for a given customer; the CustomerID may contain pattern characters, but no pattern search will be performed.

```
=countIfs_ex("orders", "ProductName", "Tofu", "ProductName",
"Chocolade", 0)
=countIfs_ex("folder1/folder2/orders", "Total", "Date", ">2011-
01-01", "Date", "<2011-07-01", 0)
```

---

vLookUp_ex(table, v, searchField, returnField, [type])

Searches the 'searchField' field of the specified table for the 'v' value and, if found, returns the 'returnField' value from the found record. Except for the 'v' parameter, all other parameters must be placed in quotation marks. If 'table' is located in nested folders, its full path must be specified with slashes.

The 'type' argument specifies how the searching procedure should be performed:

- 0 - vLookUp_ex will search for an exact match; case-insensitive.
- 1 - if an exact match is not found, vLookUp_ex will search for the largest value that is not greater than 'v'.
- -1 - if an exact match is not found, vLookUp_ex will search for the smallest value that is not smaller than 'v'.
- 2 - vLookUp_ex will search for an exact match; 'v' can be a search pattern containing '?' (any single character) and '*' (any string). To search for a literal '?' or '*' character, place a tilde (~) before it.
- 4 - vLookUp_ex will search for an exact match; 'v' can be a regular expression. Add 8 to perform case-sensitive searching; add 16 to allow empty matches.

NOTE: compared fields must be of the same type (either both text or numeric).

If 'type' is omitted, it's assumed to be 0.

If the match is not found, the function returns the #N/A! error value.

Fastest binary searches are performed for type=0, type=1 and type=-1.

```
=vLookUp_ex("products", ProductID, "ProductID", "ProductName",)
=vLookUp_ex("folder1/products", ProductID, "ProductID",
"UnitPrice",)
```


Function Index

Mathematical Functions

abs(x)

Returns the absolute value of **x**.

=abs(-1) → 1

---

acos(x)

Returns the arccosine of **x**.

=acos(-1) → 3.14159265358979

---

acosh(x)

Returns the inverse hyperbolic cosine of **x**.

=acosh(10) → 2.99322284612638

---

asin(x)

Returns the arcsine of **x**.

=asin(1) → 1.5707963267949 (=pi()/2)

---

asinh(x)

Returns the inverse hyperbolic sine of **x**.

=asinh(10) → 2.99822295029797

---

atan(x)

Returns the arctangent of **x** (-PI/2, PI/2).

=atan(1) → 0.78539816339745 (=pi()/4)

---

atan2(x, y)

Returns the arctangent of y/x (-PI, PI). If **x** is 0, or if both **x** and **y** are zero, atan2 returns 0.

=atan2(1, 1) → 0.78539816339745 (=pi()/4)

---

atanh(x)

Returns the inverse hyperbolic tangent of **x**.

=atanh(0.76159415595576) → 0.9(9)

---

ceiling(x, m)

Rounds **x** up to the nearest integer, or the nearest multiple of **m**.

=ceiling(5.1, 1) → 6
=ceiling(-5.1, 1) → -6

---

combin(n, k)

Returns the number of k-element combinations for an n-element set.

=combin(8, 2) → 28

---

combin2(n, k)

Returns the number of k-element combinations with repeating elements for an n-element set.

=combin2(8, 2) → 36

---

cos(x)

Returns the cosine of **x**. The argument is in radians.

=cos(pi()) → -1

---

cosh(x)

Returns the hyperbolic cosine of **x**.

=cosh(1) → 1.54308063481524

---

countIf(array, criteria)

Counts cells specified by a given criteria. The criteria can be one of the following:

- a string beginning with the =, >, >=, <, <= operators,
- a search pattern: a number or a string containing the special characters "?" (any character) or "*" (any string). To search for a literal "?" or "*", place a tilde (~) before it,
- an error code.

=countIf({1,2;3,4}, 2) → 1
=countIf({1,2;3,4}, ">2") → 2
=countIf({"abcde","def";"abc","a"}, "?bc*") → 2
=countIf(R1C1:R5C5, "*") → the number of cells containing any labels
=countIf({"abcde","def";"bc","a"}, "bc") → 1
=countIf({"ABcde","def";"Bc","a"}, ">=bc") → 2
=countIf({1,2;"a",#N/A!;3,#N/A!}, #N/A!) → 2

---

degrees(x)

Converts radians to degrees.

=degrees(pi()/2) → 90

---

even(x)

Rounds **x** up (away from zero) to the nearest even integer.

=even(1.5) → 2
=even(-1) → -2

---

exp(x)

Raises e to the power of **x**.

=exp(1) → 2.71828182845905

---

fact(n)

Returns the factorial of **n**. The argument must be in the range <1, 170>.

=fact(6) → 720

---

factDouble(n)

If **n** is an even number, factDouble returns the product n(n-2)(n-4)...(4)(2). If **n** is an odd number, factDouble returns the product n(n-2)(n-4)...(3)(1). The argument must be in the range <1, 170>.

=factDouble(6) → 48

---

floor(x, m)

Rounds **x** down to the nearest integer, or the nearest multiple of **m**.

=floor(5.1, 1) → 5
=floor(-5.1, 1) → -5

---

gcd(n1, n2, ...)

Returns the greatest common divisor for a given list of arguments (numbers/arrays of numbers).

=gcd(32, 24) → 8
=gcd({24,64,32}, 128) → 8

---

int(x)

Rounds **x** down to the nearest integer.

=int(5.1) → 5
=int(-5.1) → -6

---

lcm(n1, n2, ...)

Returns the least common multiple for a given list of arguments (numbers/arrays of numbers).

=lcm(8, 7) → 56
=lcm({16, 44, 32}, 11) → 352

---

ln(x)

Returns the natural logarithm of **x**.

=ln(2.71828182845905) → 1

---

log(x, [y])

Returns the logarithm of **x** to the base **y**. If **y** is omitted, it is assumed to be 10.

=log(10) → 1

---

log10(x)

Returns the base-10 logarithm of **x**.

=log10(100) → 2

---

mDeterm(m)

Returns the determinant of the square matrix **m**.

=mDeterm({8,5,-2; 2,3,1; 3,-1,-3}) → 3

---

mInverse(m)

Returns the inverse of the square matrix **m**.

=mInverse({8,5,-2; 2,3,1; 3,-1,-3}) → {-2.66666666666667, 5.66666666666667, 3.66666666666667; 3, -6, -4; -3.66666666666667, 7.66666666666667, 4.66666666666667}

---

mMult(m1, m2, ...)

Multiplies the specified matrices.

=mMult({1,2,1;0,2,1}, {1;1;2}, {2,1,1}*2) → {20, 10, 10; 16, 8, 8}

---

mod(x, y)

Returns the floating-point remainder of x/y.

=mod(10, 3) → 1
=mod(-3, 2) → -1
=mod(3, -2) → 1
=mod(-3, -2) → -1

---

mRound(x, m)

Rounds **x** to the nearest multiple of **m**, where **x** and **m** have the same sign.

=mRound(10, 3) → 9
=mRound(1.3, 0.2) → 1.4

---

multinomial(number1, number2, ...)

Returns the ratio (number1+number2+...)!/(number1!·number2!·...).

=multinomial(2, 3, 4) → 1260

---

odd(x)

Rounds **x** up (away from zero) to the nearest odd integer.

=odd(6) → 7
=odd(-4) → -5

---

pi()

Returns the value of PI (3.14159265358979).

---

power(x, y)

Returns **x** raised to the power of **y**.

---

product(x1, x2, ...)

Multiplies all the specified arguments, which can be numbers or arrays of numbers. Empty cells in arrays are skipped.

=product(3, 4, {1, 5}) → 60

---

quotient(x, y)

Returns the integer part of x/y.

=quotient(5, 2) → 2
=quotient(-10, 3) → -3

---

radians(x)

Converts degrees to radians.

=radians(180) → 3.14159265358979 (=pi())

---

rand()

Returns an evenly distributed number in the range <0, 1). A new number is returned each time the workbook is updated.

See the **mtxRand** function.

---

randBetween(x1, x2)

Returns an evenly distributed number in the range <**x1**, **x2**). A new number is returned each time the workbook is updated.

---

round(x, [digits])

Rounds **x** to the specified number of digits. If the **digits** argument is greater than 0, **x** is rounded to the specified number of decimal places. If it is 0, **x** is rounded to the nearest integer. If it is less than 0, **x** is rounded to the left of the decimal point. The default value of **digits** is 0.

=round(2.6) → 3
=round(2.15, 1) → 2.2
=round(2.149, 1) → 2.1
=round(-1.475, 2) → -1.48
=round(123.5, -2) → 100

---

roundDown(x, [digits])

Rounds **x** down (towards 0) to the specified number of digits. If the **digits** argument is greater than 0, **x** is rounded to the specified number of decimal places. If it is 0, **x** is rounded to the nearest integer. If it is less than 0, **x** is rounded to the left of the decimal point. The default value of **digits** is 0.

=roundDown(2.6) → 2
=roundDown(2.15, 1) → 2.1
=roundDown(2.7, 0) → 2
=roundDown(-2.7, 0) → -2

roundUp(x, [digits])

Rounds **x** up (away from zero) to the specified number of digits. If the **digits** argument is greater than 0, **x** is rounded to the specified number of decimal places. If it is 0, **x** is rounded to the nearest integer. If it is less than 0, **x** is rounded to the left of the decimal point. The default value of **digits** is 0.

=roundUp(2.6) → 3
=roundUp(2.15, 1) → 2.2
=roundUp(2.7, 0) → 3
=roundUp(-2.7, 0) → -3

seriesSum(x, n, m, a)

For the given numbers **x**, **n**, **m**, and array **a**, seriesSum returns the value of the power series:

a(1)x^n + a(2)x^(n+m) + a(3)x^(n+2m) + ... + a(l)x^(n+(l-1)m)

where **l** is the number of cells in the array **a**, and a(i) is the i-th cell (row-wise) in the array **a**.

For example, for the polynomial 5 + 2x + 3x^3:

=seriesSum(2, 0, 1, {5, 2, 0, 3}) → 33

sign(x)

Returns 1 if **x** is positive, 0 if **x** equals 0, and -1 if **x** is negative.

=sign(-5) → -1
=sign(5) → 5

sin(x)

Returns the sine of **x**. The argument is in radians.

=sin(pi()/2) → 1

---

sinh(x)

Returns the hyperbolic sine of **x**.

=sinh(1) → 1.1752011936438

---

sqrt(x)

Returns the square root of **x**.

=sqrt(81) → 9

---

sqrtPi(x)

Returns the square root of x*PI.

=sqrtPi(1) → 1.77245385090552

---

sum(x1, x2, ...)

Adds all the specified arguments, which can be numbers or arrays of numbers.

=sum(3, 4, {1, 5}) → 13
=sum(true, 1) → 2

---

sumIf(searchArray, criteria, [sumArray])

Adds cells specified by a given criteria. The criteria can be one of the following:

- a text string beginning with the =, >, >=, <, <= operators,
- a number or a search pattern: a text string containing the special characters "?" (any character) or "*" (any string). To search for a literal "?" or "*", place a tilde (~) before it.

The cells in **sumArray** are summed only if the corresponding cells in **searchArray** match the specified criteria. If **sumArray** is omitted, the cells from **searchArray** are summed.

=sumIf({1,2;3,2}, 2) → 4
=sumIf({1,2;3,4}, ">2") → 7
=sumIf({"abcde","def";"abc","a"}, "?bc*", {2, 4; 3, 1}) → 5
=sumIf({"abcde","def";"bc","a"}, "bc", {2, 4; 3, 1}) → 3
=sumIf({"ABcde","def";"Bc","a"}, ">=bc", {2, 4; 3, 1}) → 7

---

sumProduct(array1, array2, ...)

Multiplies numbers from the specified arrays and adds the obtained products. Empty cells are ignored.

=sumProduct({1,3}, {2,4}) → 11

---

sumSq(x1, x2, ...)

Adds the squares of all the specified arguments, which can be numbers or arrays of numbers.

=sumSq(3, 4, {1, 5}) → 51
=sumSq(true, 2) → 5

---

sumX2mY2(x_array, y_array)

Returns the sum of the differences of the squares of numbers from two arrays. Empty cells are treated as 0.

=sumX2mY2({2,5}, {6,1}) → -8

---

sumX2pY2(x_array, y_array)

Returns the sum of the sums of the squares of numbers from two arrays. Empty cells are treated as 0.

=sumX2pY2({2,5}, {6,1}) → 66

---

sumXmY2(x_array, y_array)

Returns the sum of the squares of the differences of numbers from two arrays. Empty cells are treated as 0.

=sumXmY2({2,5}, {6,1}) → 32

---

tan(x)

Returns the tangent of **x**. The argument is in radians.

=tan(pi()/4) → 1

---

tanh(x)

Returns the hyperbolic tangent of **x**.

=tanh(1) → 0.76159415595576

---

trunc(x, [digits])

Removes the fractional part of **x** to the precision specified by the number of digits to the left of the decimal point. The default value of the **digits** argument is 0.

=trunc(8.9) → 8
=trunc(-8.9) → -8
=trunc(1.256, 2) → 1.25

Function Index

Textual Functions

char(n)

Returns the character corresponding to the **n** code.

=char(65) → "A"

---

clean(text)

Removes non-printable characters from **text**.

=clean(char(7) & "abc") → "abc"

---

code(text)

Returns the code of the first character in **text**.

=code("ABC") → 65

---

concatenate(t1, t2, ...)

Merges all strings specified by the arguments, which can be text strings, numbers, or arrays. Empty cells in arrays are ignored.

=concatenate("abc", 1, "abc", 2) → "abc1abc2"

---

dollar(x, [decimals])

Converts **x** to a text string using the default currency format and the specified precision. If the **decimals** argument is greater than 0, **x** is rounded to the specified number of decimal places. If it is 0, **x** is rounded to the nearest integer. If it is less than 0, **x** is rounded to the left of the decimal point. The default value of **decimals** is 2.

=dollar(10.95) → "$10.95"
=dollar(10.9487, 3) → "$10.949"

---

exact(text1, text2)

Returns 1 if **text1** and **text2** are the same, 0 otherwise. The comparison is case-sensitive.

=exact("Abc", "abc") → 0
=exact("Abc", "Abc") → 1

---

find(searchText, withinText, [n])
find(pattern, subject, [start], [flags])

Searches **withinText** for **searchText**, starting from the position **n**, and returns the position of the first occurrence found (except when the "RegExStr" flag is used). If the string is not found, or if **n** is invalid (less than 1, or greater than the length of **withinText**), the function returns the #VALUE! error value.

The first version accepts a plain text string as **searchText**; the comparison is case-sensitive.

The second version also accepts regular expressions as **pattern** and can perform both case-sensitive and case-insensitive comparisons. The **flags** parameter can be any combination of the following options:

- 1 – **pattern** is a regular expression,

- 2 – use case-sensitive string comparison (regular expressions can override this),
- 4 – **pattern** is a regular expression, but the matching substring (or an empty string) is returned instead of its position,
- 32, 64, 128 – find the 2nd, 3rd, or 4th capturing group (the 1st is the default).

=find("text", "sample text", 1) → 8
=find("e*\d", "abcdef abcee abcde5", 1, 1) → 18

---

fixed(x, [digits], [no_separators])

Rounds **x** to the specified number of digits and formats it using the general number format. If the **digits** argument is greater than 0, **x** is rounded to the specified number of decimal places. If it is 0, **x** is rounded to the nearest integer. If it is less than 0, **x** is rounded to the left of the decimal point. If **no_separators** is 0, the number is formatted using thousands separators. If **digits** is omitted, it is assumed to be 2. The default value of **no_separators** is 0.

=fixed(10.956) → "10.96"
=fixed(1234.89, -2, 0) → "1,200"

---

left(text, [n])

Returns the first **n** characters from **text**. If **n** is omitted, it is assumed to be 1. If **n** is greater than the number of characters in **text**, the entire text is returned.

=left("abc") → "a"

---

len(text)

Returns the number of characters in **text**.

=len("text") → 4

---

lenb(text)

Returns the number of bytes in **text**. Since text is stored internally as a UTF-8 string, the number of bytes will be larger than the number of characters for strings containing non-ASCII characters (characters with codes above 127).

=lenb("örtlich") → 8

---

lower(text)

Converts all letters in **text** to lowercase.

=lower("TEXT") → "text"

---

mid(text, n1, n2)

Returns up to **n2** characters from **text**, starting from the position **n1**. If **n1** is greater than the number of characters in **text**, an empty string is returned. If **n1** is less than 1, or if **n2** is negative, the function returns the #VALUE! error value.

=mid("some text", 3, 2) → "me"

---

proper(text)

Converts the first character of each word in **text** to uppercase and all other characters to lowercase.

=proper("some teXT") → "Some Text"

---

replace(originalText, n1, n2, replaceText)
replace(pattern, subject, start, replace, [flags])

The first version removes **n2** characters from **originalText** at the position **n1** and replaces them with **replaceText**.

The second version searches **subject** for **pattern**, starting at the position **start**, and replaces all found occurrences with **replace**. The **pattern** parameter can be either a plain text string or a regular expression. The **flags** parameter can be any combination of the following options:

- 1 – **pattern** is a regular expression,
- 2 – use case-sensitive string comparison (regular expressions can override this).

The **replace** argument can contain:

- absolute references (by number) to capturing subpatterns, e.g. \1, \2 ... \999,
- \l, \L, \u, \U literals to toggle upper- and lower-case conversion on or off,
- \r, \n – "carriage return" and "line feed" literals, respectively (by default, pressing Ctrl+Enter while editing a cell inserts the \r\n sequence).

=replace("abcd", 3, 2, "*") → "ab*"
=replace("(.)a+\d{1,3}", "abc aa0102", 1, "\1", 1) → "abc 2"
=replace("(ab)", "abcdef ghijk abb123", 1, "\u\1", 1) → "ABcdef ghijk ABb123"

---

rept(text, n)

Duplicates **text** **n** times.

=rept("abc", 3) → "abcabcabc"

---

right(text, [n])

Returns the last **n** characters from **text**. If **n** is omitted, it is assumed to be 1. If **n** is greater than the number of characters in **text**, the entire **text** string is returned.

=right("abc") → "c"

---

search(searchText, withinText, [n])

Finds the first occurrence of **searchText** in **withinText** and returns its position. The search starts at the position **n**. If **n** is omitted, it is assumed to be 1 (the first

character). The **searchText** argument can represent a search pattern containing the special characters "?" (any character) or "*" (any string). To search for a literal "?" or "*", place a tilde (~) before it.

=search("text", "some text", 1) → 6
=search("?bc?e*", "abc abcd abcde ab") → 10

---

substitute(searchText, withinText, replaceText, [n])

Finds the **n**-th occurrence of **searchText** in **withinText** and replaces it with **replaceText**. To replace all occurrences of **searchText**, use 0 as the **n** argument. The default value of **n** is 0.

=substitute("a1b1c1d", "1", "-", 0) → "a-b-c-d"
=substitute("abc abcd abcde ab", "?bc?e*a", "x", 1) → "abc abcd xb"

---

trim(text)

Removes all whitespace from **text** except for single spaces between words.

=trim(" abc def ghi") → "abc def ghi"

---

upper(text)

Converts all letters in **text** to uppercase.

=upper("text") → "TEXT"

---

value(text)

Converts **text** to a number.

=value("123.45") → 123.45
=value("10 4/5") → 10.8
=value("$1,000") → 1000

Function Index

Date/Time Functions

date(year, month, day)

Returns a generic date string representing a given date.

Note: all date/time functions except **dateValue** require arguments entered as a generic date/time string, as defined by www.w3.org for date/time/period types. For example:
2006-01-31
2006-01-31T13:10:55
2006-01-31T13:10:55.123
2006-01-31T23:30:00+02:00
13:10:00
13:10:55.123
13:10:55-00:30
P1Y2M3DT10H30M50S.000 (which means a period of 1 year, 2 months, 3 days, 10 hours, 30 minutes, 50 seconds, 0 milliseconds)
-P120D (which means a period of minus 120 days)
PT63H (which means 63 hours)

=date(2005, 7, 15) → "2005-07-15"

---

dateDiff(date1, date2)

Returns a period string representing the difference between the two given dates. Both arguments must be generic date/time strings.

Note: all date/time functions except **dateValue** require arguments entered as a generic date/time string, as defined by www.w3.org for date/time/period types. For example:
2006-01-31
2006-01-31T13:10:55
2006-01-31T13:10:55.123
2006-01-31T23:30:00+02:00
13:10:00
13:10:55.123
13:10:55-00:30
P1Y2M3DT10H30M50S.000 (which means a period of 1 year, 2 months, 3 days, 10 hours, 30 minutes, 50 seconds, 0 milliseconds)
-P120D (which means a period of minus 120 days)
PT63H (which means 63 hours)

Use the **day()**, **hour()**, **minute()**, and **second()** functions to extract the respective values from the period string.

=dateDiff("2005-01-01", "2005-12-31") → "P364D"
=dateDiff("2005-12-31", "2005-01-01") → "-P364D"
=dateDiff("2005-01-01T18:15:10", "2005-01-04T12:10:00") → "P2DT17H50M50S"
=dateDiff(date(2005, 1, 1), date(2005, 12, 31)) → "P364D"
=dateDiff("07:15:00", "12:20:00") → "PT5H5M"

---

dateValue(text)

Converts text representing a date in one of the standard date/time formats to a generic date string.

Note: all date/time functions except **dateValue** require arguments entered as a generic date/time string, as defined by www.w3.org for date/time/period types. For

example:
2006-01-31
2006-01-31T13:10:55
2006-01-31T13:10:55.123
2006-01-31T23:30:00+02:00
13:10:00
13:10:55.123
13:10:55-00:30
P1Y2M3DT10H30M50S.000 (which means a period of 1 year, 2 months, 3 days, 10 hours, 30 minutes, 50 seconds, 0 milliseconds)
-P120D (which means a period of minus 120 days)
PT63H (which means 63 hours)

=dateValue("8/11/2005") → "2005-08-11"

---

day(date)

If the argument is a generic date string, **day()** returns an integer <1, 31> representing the day of the month of a given date. If the argument is a generic period string, **day()** returns the number of days of the period.

=day("2005-02-09") → 9
=day(dateValue("Apr-15")) → 15
=day("PD350T20M") → 350

---

eDate(date, n)

Returns a generic date string representing a date **n** months after or before – depending on the sign of **n** – a given date.

=eDate("2005-02-15", 4) → "2005-06-15"
=eDate("2005-01-30", 1) → "2005-02-28"

---

eDate2(date, period)

Adds **period** to **date** and returns a generic date string representing a date after or before – depending on the period – the given date. The **date** argument must be a

generic date/time string and the **period** argument must be a generic period string.

Note: all date/time functions except **dateValue** require arguments entered as a generic date/time string, as defined by www.w3.org for date/time/period types. For example:
2006-01-31
2006-01-31T13:10:55
2006-01-31T13:10:55.123
2006-01-31T23:30:00+02:00
13:10:00
13:10:55.123
13:10:55-00:30
P1Y2M3DT10H30M50S.000 (which means a period of 1 year, 2 months, 3 days, 10 hours, 30 minutes, 50 seconds, 0 milliseconds)
-P120D (which means a period of minus 120 days)
PT63H (which means 63 hours)

=eDate2("2005-01-01", "P2D") → "2005-01-03"
=eDate2("2005-02-01", "P30D") → "2005-03-03"
=eDate2("2005-03-03", "-P30D") → "2005-02-01"
=eDate2("13:34:22.33", "PT1H7M") → "14:41:22.330"
=eDate2("2005-08-10T15:27:36.10", "P10DT3H7M12.95S") → "2005-08-20T18:34:49.050"

---

eoMonth(date, n)

Returns a generic date string representing the last day of the month that is **n** months after or before the month of a given date. The **date** argument must be a generic date/time string.

=eoMonth("2005-01-01", 1) → "2005-02-28"
=eoMonth("2005-01-01", -1) → "2004-12-31"

---

hour(time)

If the argument is a generic date/time string, **hour()** returns an integer <0, 23> representing the hour. If the argument is a generic period string, **hour()** returns the number of hours of the period.

=hour("12:59:11") → 12
=hour("2005-08-08T20:00:01.01") → 20
=hour("PT21H13M") → 21

---

minute(time)

If the argument is a generic time string, **minute()** returns an integer <0, 59> representing the minute.

If the argument is a generic period string, **minute()** returns the number of minutes of the period.

=minute("12:59:11") → 59
=minute("2005-08-08T20:00:01.01") → 0
=minute("PT21H13M") → 13

---

month(date)

Returns an integer <1, 12> representing the month of a given date. The **date** argument must be a generic date/time string.

=month("2005-01-01") → 1

---

networkDays(date1, date2, [offDateArray])

Returns the number of working days between **date1** and **date2** (including both dates), excluding weekends and the dates specified in the **offDateArray** array. All arguments must be generic date/time strings.

Note: all date/time functions except **dateValue** require arguments entered as a generic date/time string, as defined by www.w3.org for date/time/period types. For example:
2006-01-31
2006-01-31T13:10:55
2006-01-31T13:10:55.123
2006-01-31T23:30:00+02:00
13:10:00

13:10:55.123
13:10:55-00:30
P1Y2M3DT10H30M50S.000 (which means a period of 1 year, 2 months, 3 days, 10 hours, 30 minutes, 50 seconds, 0 milliseconds)
-P120D (which means a period of minus 120 days)
PT63H (which means 63 hours)

=networkDays("2005-01-01", "2005-01-31") → 21
=networkDays("2005-01-01", "2005-01-31", {"2005-01-12", "2005-01-19"}) → 19

---

now()

Returns a time string representing the current time.

=now() → "14:52:16"

---

second(time)

If the argument is a generic time string, **second()** returns an integer <0, 59> representing the second.

If the argument is a generic period string, **second()** returns the number of seconds of the period.

=second("12:59:11") → 11
=second("2005-08-08T20:00:01.01") → 1
=second("PT21H13M7S") → 7

---

time(hour, minute, second)

Returns a generic time string representing a given time.

=time(16, 20, 15) → "16:20:15"

---

timeValue(text)

Converts text representing a time in one of the standard date/time formats to a generic time string.

=timeValue("6:24 PM") → "18:24:00"

---

today()

Returns a date string representing the current date.

=today() → "2005-08-15"

---

weekDay(date, [numbering])

Returns the day of the week for a given date. The **date** argument must be a generic date string. The **numbering** argument specifies which day-numbering variant should be used:

- 1 – Sunday = 1, ..., Saturday = 7
- 2 – Monday = 1, ..., Sunday = 7
- 3 – Monday = 0, ..., Sunday = 6

If **numbering** is omitted, it is assumed to be 1.

=weekDay("2005-01-01") → 7

---

weekNum(date, [numbering])

Returns the number of the week for a given date. The **date** argument must be a generic date string. The **numbering** argument specifies on what day the week should begin:

- 1 – on Sunday
- 2 – on Monday

If **numbering** is omitted, it is assumed to be 1.

=weekNum("2005-01-21") → 4

---

workDay(date, n, [offDateList])

Returns a date string representing a date **n** working days – depending on the sign of **n** – before or after a given day, excluding weekends. The optional **offDateList** array contains additional dates that should be excluded. All arguments must be in the form of generic date/time strings.

=workDay("2005-01-01", 21) → "2005-01-31"
=workDay("2005-01-01", 19, {"2005-01-12", "2005-01-19"}) → "2005-01-31"

---

year(date)

Returns an integer <1, 50000> representing the year of a given date. The **date** argument must be a generic date/time string.

=year("2005-01-01") → 2005
=year("40000-01-01T15:10:00") → 40000

Function Index

Informational Functions

textStats(field, type)

Returns statistics for a LongText or Code field:

- type = 0 – returns the number of words,
- type = 1 – returns the number of letters and numbers,
- type = 2 – returns the number of letters, numbers, and spaces,
- type = 3 – returns the count from type 2, plus punctuation characters and separators,
- type = 4 – returns the number of newline characters.

To obtain the total physical size of a LongText or Code field, use the **len()** function.

=textStats(memo, 0)

---

objectStats(field, type)
objectStats(field, imageName, propertyType)

Returns Images/Files field statistics, or the metadata encoded in JPG/TIFF/PNG/BMP/ICO images (e.g., by digital cameras).

For the **objectStats(field, type)** version:

- type = 0 – returns the number of objects,
- type = 1 – returns the size, in bytes, of the biggest object in the field,
- type = 2 – returns the size, in bytes, of the smallest object in the field,
- type = 3 – returns 1 if the field contains images (in a format supported by GS-Base), or 0 otherwise.

To obtain the total physical size of the Images/Files fields, use the **len()** function.

For the **objectStats(field, imageName, propertyType)** version, note that this version of the function should be used in Text fields.

imageName – if there are multiple images in a field, you can specify the name of a given image; if you specify an empty string, the first image in the field is used.

propertyType – a numeric property code. For the full list, please see the "objectStats" help topic. To generate a list of all possible codes for a given image, specify -1.

=objectStats(files, 0)
=objectStats(images, 1)

=objectStats(images, "", hex2dec("132")) → the 0x132 property value – a date
=objectStats(images, "", -1) → the list of all properties encoded in the given image

---

fileStats(field, propertyType)

Returns the metadata encoded in JPG/TIFF/PNG/BMP/ICO images (e.g., by digital cameras), or various multimedia-related metadata for multimedia (video/sound) files.

field – a textual field containing full file paths (or a directly entered fixed string). File paths from a given folder, or an entire disk partition, can be loaded with the "Tools > Verify Files" functions.

propertyType – a numeric property code. For the full list, please see the "fileStats" help topic. To generate a list of all possible codes for a given file, specify -1.

=fileStats(filePaths, hex2dec("132")) → the 0x132 property value – a date
=fileStats("d:\some_photo.jpg", -1) → the list of all properties encoded in the given image
=fileStats("e:\movie1.avi", 60) → returns the file's title
=fileStats("e:\sound2.mp3", 43) → returns the file's artist

---

countBlank(argument1, argument2, ...)

Counts empty cells and cells containing empty strings for the specified list of arguments.

=countBlank({1, 2, 3,,}, {""},,) → 5

---

errorType(error)

Returns an integer representing a given **error** value.

For the complete list of error codes, please see the "Data types" help topic.

---

isBlank(x)

Returns 1 if **x** refers to an empty cell or an empty string, 0 otherwise.

=isBlank("") → 1

---

isErr(x)

Returns 1 if **x** is an error value other than #N/A!, 0 otherwise.

=isErr(1/0) → 1
=isErr(#SYNTAX!) → 1

---

isError(x)

Returns 1 if **x** is an error value, 0 otherwise.

=isError(#N/A!) → 1

---

isEven(n)

Returns 1 if **n** is even, 0 otherwise. All numbers are rounded to the nearest integers.

=isEven(12) → 1
=isEven(12.6) → 0

---

isLogical(n)

Returns 1 if **n** is 1 or 0, 0 otherwise.

=isLogical(1) → 1

---

isNA(error)

Returns 1 if **error** refers to the #N/A value, 0 otherwise.

=isNA(#N/A!) → 1

---

isNonText(x)

Returns 1 if **x** refers to any value that is not a text string, 0 otherwise.

=isNonText("") → 0
=isNonText(1) → 1

---

isNumber(x)

Returns 1 if **x** represents a number (which also includes a string that can be converted to a number), 0 otherwise.

=isNumber(9) → 1
=isNumber("9") → 1

---

isOdd(n)

Returns 1 if **n** is odd, 0 otherwise. All numbers are rounded to the nearest integers.

=isOdd(12) → 0
=isOdd(12.6) → 1

---

isRef(x)

Returns 1 if **x** is a reference, 0 otherwise.

=isRef(R1C1) → 1
=isRef({1,2,3}) → 0

---

isText(x)

Returns 1 if **x** represents a text string, 0 otherwise.

=isText("a") → 1

---

n(x)

Converts **x** to a number.

=n("2005") → 2005

---

na()

Returns the #N/A error value.

=na() → #N/A!

---

type(x)

Returns the type of **x**:

- 1 – Number
- 2 – Text
- 16 – Error value
- 64 – Array

=type(1) → 1
=type({1, 2, 3}) → 64

GS-Base Help > *Calculation functions* > *Statistical functions*

Function Index

aveDev average averageA binomDist chiDist chiInv chiTest chiTest2 confidence correl count countA covar critBinom devSq exponDist fDist fInv fisher fisherInv forecast frequency fTest geoMean harMean hypGeomDist intercept kurt large linEst max maxa median min minA mode normDist normsDist normInv normsInv pearson percentile percentRank permut poisson polyReg polyPoints prob quartile rank rsq skew slope small standardize stdev stdevA stdevP stdevPA stEyx tDist timeSeries tInv trend trimMean tTest tTest2 var varA varP varPA weibull zTest zTest2

Statistical Functions

aveDev(x1, x2, ...)

Returns the average of the absolute deviations of the specified numbers from their mean.

=aveDev(4, 5, 6, 3, 5, 4) → 0.83333333333333

---

average(x1, x2, ...)

Returns the arithmetic mean of the specified arguments.

=average(4, 5, 6, 3, 5, 4) → 4.5

---

averageA(x1, x2, ...)

Returns the arithmetic mean of the specified arguments.

=averageA(4, 5, 6, 3, 5, 4) → 4.5

---

binomDist(s, t, p, c)

Returns the binomial distribution probability.

The **s** argument represents the number of successes, the **t** argument is the number of trials, and the **p** argument is the probability of success on each trial.

If the **c** argument is 1 (true), the cumulative distribution function is returned. Otherwise, the probability mass function is returned.

=binomDist(6,10,0.5,FALSE) → 0.205078125

---

chiDist(x, f_degrees)

Returns the one-tailed probability of the chi-squared distribution. The returned value is calculated as P(X < x). The **f_degrees** argument specifies the number of degrees of freedom.

=chiDist(18.307, 10) → 0.9499994109077

---

chiInv(probability, f_degrees)

Returns the inverse of the **chiDist** function for a given probability.

=chiInv(0.9499994109077, 10) → 18.307

---

chiTest(observed, expected)

Performs the test for independence of two discrete variables x (x[1], ..., x[k]) and y (y[1], ..., y[l]). The **observed** argument specifies the k × l contingency table with elements representing the observed number of experiments in which each x[i] & y[j] pair was obtained. The returned probability is calculated as P(X < c^2), where c^2 is the obtained chi-squared statistic.

The **expected** argument specifies a corresponding table with the expected number of subsequent experiments.

=chiTest({58, 35; 11, 25; 10, 23}, {45.35, 47.65; 17.56, 18.44; 16.09, 16.91}) → 0.99969180798302

---

chiTest2(observed, expected)

Returns the chi2-statistic for two discrete variables x (x[1], ..., x[k]) and y (y[1], ..., y[l]). The **observed** argument specifies the k × l contingency table with elements representing the observed number of experiments in which each x[i] & y[j] pair was obtained. The returned value can be compared against the critical chi2 values obtained with the **chiInv** function.

The **expected** argument specifies a corresponding table with the expected number of subsequent experiments.

=chiTest2({58, 35; 11, 25; 10, 23}, {45.35, 47.65; 17.56, 18.44; 16.09, 16.91}) → 16.1695750747969

---

confidence(alpha, deviation, size)

Returns the confidence interval for a population mean. The **alpha** parameter specifies the significance level (which equals 1 – confidence level). The **deviation** argument specifies the (known) population standard deviation. The **size** argument specifies the sample size.

=confidence(0.05, 2.5, 50) → 0.6929519802241

---

correl(array1, array2)

Returns the correlation coefficient of **array1** and **array2**. Both arguments can be arrays or cell ranges having the same dimensions.

=correl({3, 2, 4, 5, 6}, {9, 7, 12, 15, 17}) → 0.99705448550158

---

count(x1, x2, ...)

Counts the number of cells containing numbers. The **x** arguments can be any data type, including arrays and references.

=count({1, 3, "abc"}, {"4.5", 5.6}) → 4

---

countA(x1, x2, ...)

Counts the number of cells that are not empty. The **x** arguments can be any data type, including arrays and references.

=countA({1, 3, "abc"}, {"4.5", 5.6,,}) → 5

---

covar(array1, array2)

Returns the covariance of **array1** and **array2**. Both arguments can be arrays or cell ranges having the same dimensions.

=covar({3, 2, 4, 5, 6}, {9, 7, 12, 15, 17}) → 5.2

---

critBinom(trials, probability, alpha)

Returns the smallest value (the number of successes) for which the cumulative binomial distribution function is greater than or equal to **alpha**, for a given number of trials and the probability of success in each trial.

=critBinom(6,0.5,0.75) → 4

---

devSq(x1, x2, ...)

Returns the sum of squares of deviations of the specified numbers from their mean. The **x** arguments can be numbers, arrays of numbers, or references.

=devSq(4, 5, 8, 7, 11, 4, 3) → 48

---

exponDist(x, lambda, cumulative)

Returns the probability of the exponential distribution. The returned value is calculated as P(X < x). The **lambda** argument specifies the distribution parameter. If the **cumulative** argument is 1/TRUE, the cumulative distribution function is returned. Otherwise, the probability density function is returned.

=exponDist(0.2, 10, 1) → 0.86466471676339

---

fDist(x, f_degrees1, f_degrees2)

Returns the F probability distribution function. The **f_degrees1** and **f_degrees2** arguments specify the numbers of degrees of freedom used in the F expression, as the numerator and denominator values respectively. The returned value is calculated as P(F < x).

=fDist(15.20675, 6, 4) → 0.99729988597273

---

fInv(x, f_degrees1, f_degrees2)

Returns the inverse of the **fDist** probability distribution function. The **f_degrees1** and **f_degrees2** arguments specify the numbers of degrees of freedom used in the F expression, as the numerator and denominator values respectively.

=fInv(0.99729988597273, 6, 4) → 15.206750002102

---

fisher(x)

Returns the Fisher transformation for a given **x**. The returned value is calculated as z = 0.5*ln((1 - x)/(1 + x)).

=fisher(0.75) → 0.97295507452766

---

fisherInv(y)

Returns the inverse of the **fisher** function. The returned value is calculated as x = (exp(2*y) - 1)/(exp(2y) + 1).

=fisherInv(0.97295507452766) → 0.75

---

forecast(x, array_y, array_x)

Finds a best-fit regression line for the known x- and y-values and calculates the future value using the existing **x** value.

If the procedure is not convergent, the function returns #NUM!.

=forecast(30, {6, 7, 9, 15, 21}, {20, 28, 31, 38, 40}) → 10.6072530864198

---

frequency(data_array, bins_array)

Counts how often values in **data_array** occur in the ranges specified by **bins_array**. The returned value is a vector (a one-column array) of numbers. If **bins_array** has n elements, the returned array has n + 1 elements. Each **bins_array** element defines a range of values smaller than or equal to that element. The last range includes all values greater than the last **bins_array** element.

=frequency({79, 85, 78, 85, 83, 81, 95, 88, 97}, {70, 79, 89}) → {0; 2; 5; 2}

---

fTest(array1, array2)

Performs the F-test and returns the one-tail probability that variances in the **array1** and **array2** data sets are significantly different.

=fTest({6, 7, 9, 15, 21}, {20, 28, 31, 38, 40}) → 0.67584107664634

---

geoMean(x1, x2, ...)

Returns the geometric mean for the specified values. The **x** arguments can be numbers, arrays of numbers, or references.

=geoMean(4, 5, 8, 7, 11, 4, 3) → 5.47698696965696

---

harMean(x1, x2, ...)

Returns the harmonic mean for the specified values. The **x** arguments can be numbers, arrays of numbers, or references.

=harMean(4, 5, 8, 7, 11, 4, 3) → 5.02837596206173

---

hypGeomDist(sample_s, sample_size, population_s, population_size)

Returns the hypergeometric distribution function: the probability of **sample_s** successes in a **sample_size**-element sample, with a known **population_s** number of successes in the **population_size**-element population.

=hypGeomDist(1, 4, 8, 20) → 0.36326109391125

---

intercept(array_y, array_x)

Finds a best-fit regression line for the known x- and y-values and returns the point where it intersects the y-axis.

The **array_y** and **array_x** arrays must have the same dimensions.

If the procedure is not convergent, the function returns #NUM!.

=intercept({2, 3, 9, 1, 8}, {6, 5, 11, 7, 5}) → 0.04838709677419

---

kurt(x1, x2, ...)

Returns the kurtosis of the specified numbers. The **x** arguments can be numbers, arrays of numbers, or references.

=kurt(3, 4, 5, 2, 3, 4, 5, 6, 4, 7) → -0.1517993720842

---

large(array, k)

Returns the k-th largest value in the specified **array**.

=large({3, 4, 5, 2, 3, 4, 5, 6, 4, 7}, 3) → 5

---

linEst(array_y, [array_x], [const_b])

Finds a best-fit regression line for the known x- and y-values and returns a two-element vector containing the a and b coefficients of the y = a*x + b line.

If the **const_b** argument is 0 (false), the b parameter is assumed to be 0. If **const_b** is omitted, it is assumed to be 1.

If the **array_x** argument is omitted, it is assumed to be an array containing integers from 1 to the number of elements in **array_y**.

The **array_y** and **array_x** arrays must have the same dimensions.

If the procedure is not convergent, the function returns #NUM!.

=linEst({2, 3, 9, 1, 8}, {6, 5, 11, 7, 5},) → {0.04838709677419; 0.66935483870968}

---

max(x1, x2, ...)

Returns the largest value. The **x** arguments can be numbers, arrays of numbers, or references.

=max(4, 5, 8, 7, 11, 4, 3) → 11

---

maxa(x1, x2, ...)

Returns the largest value. The **x** arguments can be numbers, arrays of numbers, or references.

=maxa(4, 5, 8, 7, 11, 4, 3) → 11

---

median(x1, x2, ...)

Returns the median of the specified numbers. The median is a number such that half of the values are greater and half are smaller than it. The **x** arguments can be numbers, arrays of numbers, or references.

=median(1, 2, 3, 4, 5, 6) → 3.5

---

min(x1, x2, ...)

Returns the smallest value. The **x** arguments can be numbers, arrays of numbers, or references.

=min(4, 5, 8, 7, 11, 4, 3) → 3

---

minA(x1, x2, ...)

Returns the smallest value. The **x** arguments can be numbers, arrays of numbers, or references.

=minA(4, 5, 8, 7, 11, 4, 3) → 3

---

mode(x1, x2, ...)

Returns the most frequently occurring number among the specified numbers. The **x** arguments can be numbers, arrays of numbers, or references.

=mode({5.6, 4, 4, 3, 2, 4}) → 4

---

normDist(x, mean, deviation, cumulative)

Returns the normal distribution for a given mean and standard deviation. If the **cumulative** argument is 1 (true), the **normDist** function returns the cumulative distribution function. Otherwise, the probability mass function is returned.

=normDist(42, 40, 1.5, true) → 0.90878877482188

---

normsDist(x)

Returns the standard normal cumulative distribution (where the mean is 0 and the standard deviation is 1).

=normsDist(1.33333333432674) → 0.90878877498481

---

normInv(probability, mean, deviation)

Returns the inverse of the normal distribution for the given mean and standard deviation.

=normInv(0.90878877482188, 40, 1.5) → 42.0000000014901

---

normsInv(probability)

Returns the inverse of the standard normal cumulative distribution (where the mean is 0 and the standard deviation is 1).

=normsInv(0.90878877482188) → 1.33333333432674

---

pearson(array1, array2)

Returns the Pearson correlation coefficient for numbers in **array1** and **array2**. Both arrays must have the same dimensions.

=pearson({9,7,5,3,1},{10,6,1,5,3}) → 0.69937860618024

---

percentile(array, k)

Returns the k-th percentile for the numbers in **array**. The **k** argument must be in the range <0, 1>. If k is not a multiple of 1/(n - 1), where n is the number of values in **array**, the returned value is interpolated.

=percentile({1, 2, 3, 4}, 0.3) → 1.9

---

percentRank(array, x, [significance])

Returns the rank of the value **x** in the specified array of numbers. The returned percentage value represents the relative standing of **x** within **array**.

The **significance** argument specifies the number of decimal places in the result. If it is omitted, it is assumed to be 3. If **array** does not include **x**, the function returns an interpolated value.

=percentRank({1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, 4,) → 0.333

---

permut(n, k)

Returns the number of k-element permutations of an n-element set.

=permut(100, 3) → 970200

---

poisson(x, mean, cumulative)

Returns the Poisson distribution for a given number of events **x**, where **mean** is the distribution parameter.

If the **cumulative** argument is 1 (true), the **poisson** function returns the cumulative distribution function. Otherwise, the probability mass function is returned.

=poisson(2, 5, false) → 0.08422433748857

---

polyReg(t, y, sy, r, type)

Performs polynomial regression using orthogonal polynomials. If w(t) is a polynomial of the controlled variable **t**, the **polyReg** function calculates the coefficients of a family of r orthogonal polynomials (from degree 0 to degree r-1) and suggests the most suitable (minimal) degree of w(t).

The polynomial of the (r - 1)-th degree can be expressed as:

w(t) = x[1]*f_1(t) + x[2]*f_2(t) + ... + x[r]*f_r(t)

where the f_i functions belong to a family of r orthogonal polynomials defined by the lower triangular matrix B[r,r], such that for each f_i, which is a polynomial of degree j - 1:

f_i(t) = Sum<k=1, j>(B[j, k] * t^(k - 1))

and

Sum<i=1, N>(g(i) * f_j(t[i]) * f_k(t[i])) = { 1 if i=j, 0 if i≠j }

for given measurement weights g(i).

The **t** argument is a column vector (a one-column array) containing N values t[i] of the controlled variable **t**.

The **y** argument is a column vector containing N measurements y[i] of the w(t) variable.

The **sy** argument is a column vector containing N values of normally distributed measurement errors.

The **r** argument specifies the number of coefficients to find (r - 1 is the maximum degree of the searched polynomial); r <= N.

- If **type** = 1, the function returns the x column vector of the calculated x[1...r] coefficients. By definition, all x[i] have the same standard deviation of 1; thus, the last x[i] coefficient significantly different from 0 determines the degree of the polynomial that should be used to describe w(t).
- If **type** = 2, the function returns the B[r,r] matrix.
- If **type** = 3, the function returns a column vector of chi2[1...r] values that can be used to verify the goodness-of-fit for each analyzed polynomial of the i-th degree and N-i-1 degrees of freedom.

=polyReg({1; 2; 3; 4; 5}, {1.1; 1.2; 1.3; 1.4; 1.5}, {0.2; 0.2; 0.2; 0.2; 0.23}, 4, 1) → {14.0688; 1.4979; 0; 0} – which means that the data can be described with a 1st-degree polynomial (a line).

---

polyPoints(x, B, chi2, n, r, t, p)

The **polyPoints** function uses results returned by **polyReg** to calculate values lying on the specified polynomial line, along with their confidence limits, for a given probability **p**.

The **x**, **chi2** vectors and the **B** matrix are values returned by the **polyReg** function.

The **n** argument specifies the number of elements t[i] passed to the **polyReg** function (the **t** argument).

The **r** argument specifies the r-1 degree of the polynomial that the calculation is performed for. It must be greater than or equal to 1, and not greater than the **r** value passed to the **polyReg** function.

The **t** argument is a column vector (a one-column array) containing N new values that the polynomial is calculated for.

The **p** argument specifies the probability used to calculate the confidence limits for a given point lying on the polynomial line.

The function returns a two-column matrix; the first column contains N values y'[i] lying on the polynomial line, and the second column contains the corresponding N confidence limits eta[i] for each y'[i], such that the true value is expected to be in the range <y'[i] - eta[i], y'[i] + eta[i]> at the specified probability **p**.

Input arguments used for **polyReg**:
t = {1; 2; 3; 4; 5; 6; 7}
y = {3.1; 1.2; 1.3; 2.4; 3.5; 6.5; 7.5}
sy = {0.2; 1.8; 0.2; 0.2; 1.9; 2.8; 3.8}
r = 7

Arguments returned by **polyReg**:
x[1...7]
B[1...7, 1...7]
chi2[1...7]

Calculating polynomial values for the following new points:
t = {1.5; 2.5; 3.5; 8}

=polyPoints(x, B, chi2, 7, 5, t, 0.85) → {1.901, 1.176; 1.112, 0.681; 1.754, 0.320; 9.930, 22.740}
=index(polyPoints(x, B, chi2, 7, 5, t, 0.85),,1) → {1.901; 1.112; 1.754; 9.930}

---

prob(array_x, array_p, lower_limit, [upper_limit])

Returns the cumulative probability for values in the range <**lower_limit**, **upper_limit**>, based on the intervals specified by **array_x** and the corresponding probabilities in **array_p**.

Each **array_p** element must be in the range <0, 1>, and their sum must equal 1. If the **upper_limit** argument is omitted, the function returns the probability for the **lower_limit** value.

=prob({0, 1, 2, 3}, {0.2, 0.3, 0.1, 0.4}, 1, 3) → 0.8

---

quartile(array, n)

Returns the quartile of the data set. The **n** argument specifies which value should be returned:

- 0 – minimum value
- 1 – 1st quartile
- 2 – 2nd quartile
- 3 – 3rd quartile
- 4 – maximum value

=quartile({1, 2, 4, 7, 8, 9, 10, 12}, 1) → 3.5

---

rank(x, array, [order])

Returns the rank of a given number in the data set specified by the **array** argument. The rank of a number is its position after the data set is sorted. Equal numbers have the same rank.

If the **order** argument is 1/TRUE, the returned rank corresponds to ascending order. Otherwise, descending order is used. If it is omitted, it is assumed to be 0.

=rank(3.5, {7, 3.5, 3.5, 1, 2}, 1) → 3

---

rsq(array1, array2)

Returns the square of the Pearson correlation coefficient for numbers in **array1** and **array2**. Both arrays must have the same dimensions.

=rsq({9, 7, 5, 3, 1}, {10, 6, 1, 5, 3}) → 0.48913043478261

---

skew(x1, x2, ...)

Returns the skewness of (the distribution consisting of) the specified numbers. The **x** arguments can be numbers, arrays of numbers, or references.

=skew(3, 4, 5, 2, 3, 4, 5, 6, 4, 7) → 0.3595430714068

---

slope(array_y, array_x)

Finds a best-fit regression line for the known x- and y-values and returns the slope of that line.

The **array_y** and **array_x** arrays must have the same dimensions.

=slope({2, 3, 9, 1, 8}, {6, 5, 11, 7, 5}) → 0.66935483870968

---

small(array, k)

Returns the k-th smallest value in the specified **array**.

=small({3, 4, 5, 2, 3, 4, 5, 6, 4, 7}, 3) → 3

---

standardize(x, mean, deviation)

Returns a normalized value z for the standard normal distribution, such that normDist(x, mean, deviation) = normsDist(z).

=standardize(42, 40, 1.5) → 1.3(3)

---

stdev(x1, x2, ...)

Returns the standard deviation based on a given sample. The **x** arguments can be numbers, arrays of numbers, or references.

=stdev(1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) → 27.4639157198435

---

stdevA(x1, x2, ...)

Returns the standard deviation based on a given sample. The **x** arguments can be numbers, arrays of numbers, or references.

=stdevA(1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) → 27.4639157198435

---

stdevP(x1, x2, ...)

Returns the standard deviation based on the entire population. The **x** arguments can be numbers, arrays of numbers, or references.

=stdevP(1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) → 26.0545581424825

---

stdevPA(x1, x2, ...)

Returns the standard deviation based on the entire population. The **x** arguments can be numbers, arrays of numbers, or references.

=stdevPA(1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) → 26.0545581424825

---

stEyx(array_y, array_x)

Finds a best-fit regression line for the known x- and y-values and returns the standard error of the predicted y-values lying on that line.

=stEyx({2, 3, 9, 1, 8}, {6, 5, 11, 7, 5}) → 3.74560674557182

---

tDist(x, f_degrees)

Returns the Student's t-distribution function. The **x** argument is the numeric value for which the distribution is calculated. The **f_degrees** argument is the number of degrees of freedom.

The returned value is calculated as P(X < x).

=tDist(1.96, 60) → 0.97267753526449

---

timeSeries(y, l, k, p)

Analyzes a time series: a set of measurements y[i] performed at regular intervals t[i]. The actual relationship between the real y[i] values and the t[i] values is unknown, and the measurement errors are assumed to be normally distributed but of unknown magnitude. To find a trend line, the function uses moving averages and assumes that the relationship is linear (polynomial) in the neighborhood of each y[i].

The **y** argument is a column vector containing N measurements y[i].

The **l** argument specifies the degree of the polynomial that should be used to approximate the trend line.

The **k** argument specifies, for each y[i], the range <i - k, i + k> of points that will be used to calculate the moving average and the polynomial coefficients. For points lying within the first and last compartments, such that i <= k or i > n - k, the function extends the trend line using the coefficients calculated respectively for the points i = k + 1 and i = n - k.

The **p** argument specifies the probability used to calculate the confidence limits for the points lying on the trend line.

The function returns a two-column array; the first column contains N + 2k values y'[i] lying on the trend line, and the second column contains N + 2k values eta[i] defining the confidence limits for each y'[i], such that the true value is expected to be in the range <y'[i] - eta[i], y'[i] + eta[i]> at the specified probability **p**.

=timeSeries({11; 8; 7; 2; 1; 3; 8; 4}, 3, 2, 0.8) →
{7.800, 71.396; 10.800; 10.800; 8.800; 5.800; 2.771; 0.914; 4.029; 7.314; 4.171; 11.4; 45.400}

=timeSeries({11; 8; 7; 2; 1; 3; 8; 4}, 2, 3, 0.8) →
{35.071, 10.531; 25.928, 7.154; 18.285, 4.447; 12.142, 2.490; 7.500, 1.525; 2.714, 1.647; 2.761, 2.363; 2.928, 2.188; 4.017, 2.188; 6.190, 3.573; 9.285, 6.380; 13.357, 10.264; 18.404, 15.109}

=index(timeSeries({11; 8; 7; 2; 1; 3; 8; 4}, 2, 3, 0.8),,1) →
{30.071; 25.928; 18.285; 12.142; 7.500; 2.714; 2.761; 2.928; 4.017; 6.190; 9.285; 13.357; 18.404}

---

tInv(p, f_degrees)

Returns the inverse of the Student's (one-tailed) t-distribution function for a given probability value.

=tInv(0.97267753526449, 60) → 1.95999983312891

---

trend(array_y, [array_x], [array_n], [const_b])

Finds a best-fit regression line for the known x- and y-values and returns a vector (a one-column array) of predicted y-values for the new x-values specified in the **array_n** array.

If the **const_b** argument is 0/FALSE, the b parameter is assumed to be 0. The default value of **const_b** is 1.

If the **array_x** argument is omitted, it is assumed to be an array containing integers from 1 to the number of elements in **array_y**. If the **array_n** argument is omitted, it is assumed to be the same as **array_x**.

The **array_y** and **array_x** arrays must have the same dimensions.

If the procedure is not convergent, the function returns #NUM!.

=trend({2, 3, 9, 1, 8}, {6, 5, 11, 7, 5}, {12, 13, 14}, 1) → {8.08064516129032; 8.75; 9.41935483970968}

---

trimMean(array, percent)

Returns the mean of the specified data set after excluding a given percentage of numbers from the top and bottom of that data set.

The **percent** argument must be in the range <0, 1>. It is rounded to the nearest multiple of 2 so that the data set elements can be excluded symmetrically.

=trimMean({4, 5, 6, 7, 2, 3, 4, 5, 1, 2, 3}, 0.2) → 3.77777777777778

---

tTest(array1, array2, tails)

Performs the T-test and returns the probability that the means in the **array1** and **array2** data sets are significantly different.

The **tails** argument specifies the number of distribution tails: 1 or 2.

=tTest({3, 4, 5, 8, 9, 1, 2, 4, 5}, {6, 19, 3, 2, 14, 4, 5, 17, 1}, 2) → 0.1919958849411

---

tTest2(array1, array2)

Returns the t-statistic for the **array1** and **array2** data sets.

The returned value can be compared against critical t-statistic values obtained with the **tInv** function.

=tTest2({3, 4, 5, 8, 9, 1, 2, 4, 5}, {6, 19, 3, 2, 14, 4, 5, 17, 1}) → -1.3622298275595

---

var(x1, x2, ...)

Returns the variance based on a given sample. The **x** arguments can be numbers, arrays of numbers, or references.

=var(1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) → 754.266666666667

---

varA(x1, x2, ...)

Returns the variance based on a given sample. The **x** arguments can be numbers, arrays of numbers, or references.

=varA(1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) → 754.266666666667

---

varP(x1, x2, ...)

Returns the variance based on the entire population. The **x** arguments can be numbers, arrays of numbers, or references.

=varP(1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) → 678.84

---

varPA(x1, x2, ...)

Returns the variance based on the entire population. The **x** arguments can be numbers, arrays of numbers, or references.

=varPA(1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) → 678.84

---

weibull(x, alpha, beta, cumulative)

Returns the Weibull distribution for a given **x**. The **alpha** and **beta** arguments are the distribution parameters.

If the **cumulative** argument is 1 (true), **weibull** returns the cumulative distribution function. Otherwise, the probability mass function is returned.

=weibull(105, 20, 100, true) → 0.92958139006928

---

zTest(array, x, [deviation])

Performs the Z-test and returns the two-tailed probability that **x** does not belong to the population represented by the n-element **array** sample.

The returned value is calculated as 1 - normsDist((m - x)/deviation/sqrt(n)), where m is the sample mean.

The **deviation** argument is the known population standard deviation. If it is omitted, the sample deviation is used.

=zTest({3, 6, 7, 8, 6, 5, 4, 2, 1, 9}, 4,) → 0.09057420261958

---

zTest2(array, x, [deviation])

Returns the z-statistic for a given variable and a given data set. The population is represented by the n-element **array** sample.

The returned value is calculated as z = (m - x)/deviation/sqrt(n), where m is the sample mean.

The **deviation** argument is the known population standard deviation. If it is omitted, the sample deviation is used.

=zTest2({3, 6, 7, 8, 6, 5, 4, 2, 1, 9}, 4,) → 1.33722748248063

Function Index

Financial Functions

cumIPMT(rate, nper, pv, n1, n2)

Returns the cumulative interest paid on a loan between the periods startPeriod and endPeriod, based on a given interest rate (relative to one period), the total number of

periods nper, the present value pv and payments made in arrears.

=cumIPMT(0.0075, 360, 125000, 13, 24) → -11135.2321307508

---

cumPRINC(rate, period, pv, n1, n2)

Returns the cumulative principal paid on a loan between the periods startPeriod and endPeriod, based on a given interest rate (relative to one period), the total number of periods nper, the present value pv and payments made in arrears.

=cumPRINC(0.0075, 360, 125000, 13, 24) → -934.107123420876

---

db(cost, salvage, life, period, [month])

Returns the depreciation of an asset for a given period using the fixed-declining balance method, where cost is the initial cost of an asset, salvage is the value at the end of the depreciation, life is the total number of depreciation periods, period is the period for which the depreciation is calculated, month is the number of months in the first year. The default value of month is 12.

=db(1000000, 100000, 6, 1, 7) → 185912.959716189
=db(1000000, 100000, 6, 7, 7) → 15867.888384391

---

ddb(cost, salvage, life, period, [factor])

Returns the depreciation of an asset for a given period using the double-declining balance method or other method determined by the optional factor declining rate, where cost is the initial cost of an asset, salvage is the value at the end of the depreciation, life is the total number of depreciation periods, period is the period for which the depreciation is calculated, factor is the rate at which the balance declines. The default value of factor is 2.

=ddb(2400, 300, 3650, 1) → 1.31506849315068
=ddb(2400, 300, 10, 2, 1.5) → 306

---

dollarDe(x, denominator)

Converts a dollar price represented by a fraction with a given denominator value to a decimal number.

=dollarDe(1.02, 16) → 1.125 (=1 2/16)

---

dollarFr(x, denominator)

Converts a decimal number to a dollar price represented by a fraction with a given denominator value.

=dollarFr(1.125, 16) → 1.02 (=1 2/16)

---

effect(rate, nper)

Calculates and returns the annual effective interest rate based on the nominal annual interest rate and the number of compounding periods per year nper.

=effect(5.25%, 4) → 0.05354266737076

---

fv(rate, pmt, [pv], [type])

Returns the future value of an investment based on constant periodic payments and interest rate where rate is the interest rate (relative to one period), pmt is the periodic payment, pv is the present value of the investment and type specifies the payment type: 0 – at the end of each period, 1 – at the beginning of each period. If pmt is omitted, pv must be specified. If pv is omitted, it is assumed to be 0 and pmt must be specified. The default value of type is 0.

=fv(0.5%, 10, -200, -500, 1) → 2581.40337406014

---

fvSchedule(pv, v1, v2, ...)

Returns the future value of an initial principal after the specified number of periods at a variable interest rate in each period. The v_ arguments are numbers or array(s) of numbers representing the subsequent interest rates. Empty cells are treated as zeros.

=fvSchedule(1, {0.09, 0.11, 0.1}) → 1.33089

---

ipmt(rate, period, nper, pv, [fv])

Returns the interest payment for a given period based on constant periodic payments and a constant interest rate where nper is the total number of periods, pv is the present value or the current value of the future payments, fv is the future value (after the last payment is made) and payments occur at the end of each period. If fv is omitted, it is assumed to be 0.

=ipmt(0.1/12, 1, 36, 8000) → -66.6666666666667

irr(pv1, pv2, ..., guess)

Returns the internal interest rate for a series of (negative and positive) cash flows occurring at regular intervals. The pv(n) arguments can be numbers or arrays of numbers, guess is the value of the rate that irr will use as the initial approximation of the actual rate. Empty cells in arrays are ignored.

=irr(-70000, 12000, 15000, 18000, 21000, 26000, 0%) → 0.08663094803653

mirr(flowsArray, rate)

Returns the modified internal rate of return for a series of (negative and positive) cash flows assuming that the obtained cash can be reinvested at a given interest rate (relative to one period). Empty cells in arrays are ignored. To evaluate an investment, compare that value with the interest rate paid on the capital.

=mirr({-120000, 39000, 30000, 21000, 37000, 46000}, 12%) → 0.12609413036591

nominal(rate, nper)

Calculates and returns the annual nominal interest rate, based on a given effective rate and the number of compounding periods per year.

=nominal(5.3543%, 4) → 0.05250031986836

nper(rate, pmt, pv, [fv], [type])

Returns the total number of periods for an investment based on constant periodic payments and a constant interest rate. The pmt argument is the periodic payment, pv is the present value, fv is a cash balance after the last payment is made and type specifies the payment type: 0 – at the end of each period, 1 – at the beginning of each period. If fv is omitted, it is assumed to be 0. The default value of type is 0.

=nper(12%/12, -100, -1000, 10000, 1) → 59.6738656742946
=nper(1%, -100, 1000) → 10.5886444594232

npv(rate, v1, v2, ...)

Returns the net present value of an investment, given the rate discount rate and a series of future payments and income occurring at regular intervals in arrears. The rate argument

is the discount rate (relative to one period), the v_ values are numbers or arrays of numbers representing payments or income. Empty cells are ignored.

=npv(10%, -10000, 3000, 4200, 6800) → 1188.44341233522

---

pmt(rate, nper, pv, [fv], [type])

Returns the payment for a loan based on constant payments and constant interest rate. The rate argument is the interest rate, nper is the total number of payment periods, pv the present value, fv is a cash balance after the last payment is made and type specifies the payment type: 0 – at the end of each period, 1 – at the beginning of each period. If fv is omitted, it is assumed to be 0. The default value of type is 0.

=pmt(8%/12, 10, 10000) → -1037.03208935916
=pmt(8%/12, 10, 10000, 0, 1) → -1030.16432717798

---

ppmt(rate, period, nper, pv, [fv], [type])

Returns the principal portion of a given payment. The rate argument is the interest rate (relative to one period), period is the period that the payment refers to, nper is the total number of periods, pv is the present value, fv is a cash balance after the last payment is made and type specifies the payment type: 0 – at the end of each period, 1 – at the beginning of each period. If fv is omitted, it is assumed to be 0. The default value of type is 0.

=ppmt(10%/12, 1, 24, 2000) → -75.6231860083666

---

pv(rate, nper, pmt, [fv], [type])

Returns the present value of an investment where rate is the interest rate (relative to one period), nper is the total number of compounding periods, pmt is the periodic payment, fv is the future value and type specifies the payment type: 0 – at the end of each period, 1 – at the beginning of each period. If fv is omitted, it is assumed to be 0. The default value of type is 0.

=pv(0.08/12, 12*20, 500, , 0) → -59777.1458511878

---

rate(nper, pmt, pv, [fv], [type])

Returns the average interest rate per period. The nper argument is the total number of periods, pmt is the periodic payment, pv is the present value, fv is the future value, type is

the payment type: 0 – at the end of each period, 1 – at the beginning of each period. If fv is omitted, it is assumed to be 0. The default value of type is 0.

=rate(48, -200, 8000) → 0.0077014724882

---

sln(cost, salvage, life)

Returns the straight-line depreciation of an asset for one period. The cost argument is initial value of an asset, salvage is the value at the end of the depreciation, life is the total number of depreciation periods.

=sln(30000, 7500, 10) → 2250

---

syd(cost, salvage, life, period)

Returns the sum-of-years' digits depreciation of an asset for a given period. The cost argument is the initial value of an asset, salvage is the value at the end of the depreciation period, life is the total number of depreciation periods, period is the period for which the SYD value is calculated.

=syd(30000, 7500, 10, 1) → 4090.909090909

---

xirr(flows, dates, [guess])

Returns the internal rate of return for a schedule of cash flows without the periodicity constraint. The flows argument is an array of numbers representing cash flows and the dates argument is an array of generic date/time strings indicating when the corresponding cash flows occur. The first element of the flows array is optional and represents the initial cost/investment. The first element of the dates array specifies the beginning of the schedule; the remaining dates can be placed in any order but they must be later than the first one. The guess argument is the value of the rate that xirr will use as the initial approximation of the actual rate. The default value of guess is 10%.

=xirr({-10000, 2750, 4250, 3250, 2750}, {"1992-01-01", "1992-03-01", "1992-10-30", "1993-02-15", "1993-04-01"}, 0.1) → 0.37336253351883

---

xnpv(rate, flows, dates)

Returns the net present value of an investment for a schedule of cash flows without the periodicity constraint. The rate argument is the discount rate, flows is an array of numbers representing cash flows and the dates argument is an array of generic date/time strings indicating when the corresponding cash flows occur. The first element of the flows array is

optional and represents the initial cost/investment. The first element of the dates array specifies the beginning of the schedule; the remaining dates can be placed in any order but they must be later than the first one.

=xnpv(0.09, {-10000, 2750, 4250, 3250, 2750}, {"1992-01-01", "1992-03-01", "1992-10-30", "1993-02-15", "1993-04-01"}) → 2086.64760205153

Function Index

Lookup and Reference Functions

array(rows, columns, v1, v2, ...)

Creates an array out of the **v** values, which can be field names, numbers, or text strings. The **rows** and **columns** parameters specify the dimensions of that array.

=array(2, 1, "abc", 10) → {"abc"; 10}
=array(1, 3, 1, 5, 10) → {1, 5, 10}
=array(2, 2, 1, 5, 10, field1) → {1, 5; 10, v}, where v is the value of field1

---

fields(start, end) / fieldNames(start, end)

Creates an array with field values (or field names, for **fieldNames()**) from field **start** to field **end** within a given record. Binary fields are not included. The **start** and **end** parameters are 1-based field indexes (start <= end).

The arrays returned by this function can be used in any built-in or external (e.g., **UDF()** Python) function that accepts ranges/arrays of values. For example:

=fields(1, 2) → e.g. {"abc"; 10}
=median(fields(2, 15)) → the median for field values from the 2nd field to the 15th field
=sum(fields(1, 100)) → the sum for field values from the 1st field to the 100th field
=udf("e:\python_module.py", "my_file_function", 3 + 32, fields(2, 30)) → a CSV data block with the specified field values, to be inserted in a LongText field

---

columns(array)

Returns the number of columns in **array**.

=columns({1, 2, 3; 4, 5, 6}) → 3

---

hLookUp(v, array, n, [type])

Searches the top row of the specified **array** for the **v** value and, if found, returns a value from the same column and the n-th row of **array**.

The **type** argument specifies how the searching procedure should be performed:

- 0 – hLookUp searches for an exact match; **v** can be a search pattern containing "?" (any single character) and "*" (any string); to search for a literal "?" or "*", place a tilde (~) before it,
- 1 – if an exact match is not found, hLookUp searches for the largest value that is not greater than **v**,
- -1 – if an exact match is not found, hLookUp searches for the smallest value that is not smaller than **v**.

If the match is not found, the function returns the #N/A! error value. If **type** is omitted, it is assumed to be 1.

=hLookUp(2, {1, 2, 3; "a", "b", "c"}, 2, 0) → "b"
=hLookUp(2.5, {1, 2, 3; "a", "b", "c"}, 2, -1) → "c"
=hLookUp(2.5, {1, 2, 3; "a", "b", "c"}, 2, 1) → "b"
=hLookUp("*bc??", {"abc", "abcde", "ac"; 1, 2, 3}, 2, 0) → 2

---

index(x, [m], [n])

If **x** is a cell reference (a range of cells) and **m** and **n** are numbers, the function returns the reference of the cell in the m-th row and the n-th column of **x**. If either **m** or **n** is omitted or is 0, the function returns a reference to, respectively, the n-th column or the m-th row. If **x** is not a reference, the function returns cell value(s) instead of references.

If either **m** or **n** is an array of numbers, the function returns, respectively, a set of rows (of cell values) of **x** or a set of columns. If **m** is an array, the **n** argument must be omitted. If **n** is an array, the **m** argument must be omitted.

=index(A1:D4, 2, 2) → B2
=index({1, 3, 5; 2, 4, 6}, 2, 2) → 4
=index({1, 3, 5; 2, 4, 6}, , 2) → {3; 4}
=index({1, 3, 5; 2, 4, 6}, {2, 1}) → {2, 4, 6; 1, 3, 5}

---

lookUp(v, array)
lookUp(v, searchArray, resultArray)

The first version searches either (1) the top row of **array** (if **array** has more columns than rows) or (2) the leftmost column of **array** (if **array** has more rows than columns), and returns a value, respectively, from either (1) the same column and the last row, or (2) the same row and the last column.

The second version of the **lookUp** function uses the **searchArray** and **resultArray** arguments, which are one-column or one-row arrays with the same number of cells. After finding the **v** value in **searchArray**, the corresponding value from **resultArray** is returned.

If an exact match cannot be found, the largest value not greater than **v** is returned. On failure, both versions return the #N/A error value.

=lookUp(3, {1, 5, 3, 4; "a", "b", "c", "d"}) → "c"
=lookUp(3, {1, 5, 3, 4}, {"a", "b", "c", "d"}) → "c"

---

match(v, array, [type])

Returns the relative position of the **v** value in the specified **array**.

The **type** argument specifies how the searching procedure should be performed:

- 0 – the function searches for an exact match; **v** can be a search pattern containing "?" (any single character) and "*" (any string); to search for a literal "?" or "*", place a tilde (~) before it,
- 1 – if an exact match is not found, the function searches for the largest value that is not greater than **v**,
- -1 – if an exact match is not found, the function searches for the smallest value that is not smaller than **v**.

If **type** is omitted, it is assumed to be 1. If the searched **v** value is not found, **match** returns the #N/A! error value.

=match(2, {1, 2, 3}, 0) → 2
=match("b", {1, 2, 3; "a", "b", "c"}, 0) → 5
=match("*bc??", {"abc", "abcde", "ac"}, 0) → 2

---

rows(array)

Returns the number of rows in **array**.

=rows({1, 2, 3; 4, 5, 6}) → 2

---

rtd(prog_id, machine, theme1, ..., themeN)

Receives data from a COM (automation) server identified by its registered **prog_id** name.

The COM server application can run on a remote computer or, if the **machine** argument is omitted, locally.

The **theme1**, ..., **themeN** arguments specify the names of the requested properties. Each property can return a variant of the following types:

- (1) VT_EMPTY (empty cells)
- (2) VT_UI1 (error codes)
- (3) VT_R8 (numbers)
- (4) VT_BSTR (text strings)
- (5) SAFEARRAY: an array of (1)–(4)

If two or more names are specified, the function returns a vector (a one-column array) of values. If a given property returns an array, it must be the only **theme** argument.

=rtd("MSComCtl2.MonthView.2", , "Day")

---

sort(array, column1, order1, column2, order2, column3, order3)

Sorts **array** based on the specified column/order pairs and returns the result as a vector of relative row indices.

The **column(n)** arguments specify the relative position of the columns that should be used as the sort keys, and the **order(n)** argument specifies the sort order:

- 0 – descending order,
- 1 – ascending order.

Use the **index** function to obtain the final sorted array.

=sort({3; 1; 5; 2; 4}, 1, 1) → {2; 4; 1; 5; 3}
=sort({3; 1; 5; 2; 4}, 1, 0) → {3; 5; 1; 4; 2}
=sort({2, "b"; 2, "a"; 3, "d"; 3, "c"; 1, "e"}, 1, 1, 2, 1) → {5; 2; 1; 4; 3}
=index({2, "b"; 2, "a"; 3, "d"; 3, "c"; 1, "e"}, {5; 2; 1; 4; 3}) → {1, "e"; 2, "a"; 2, "b"; 3, "c"; 3, "d"}

---

transpose(array)

Transposes a given array.

=transpose({1, 2, 3; 4, 5, 6}) → {1, 4; 2, 5; 3, 6}

---

vLookUp(v, array, n, [type])

Searches for the **v** value in the leftmost column of the specified **array** and, if found, returns a value from the same row and the n-th column of **array**.

The **type** argument specifies how the searching procedure should be performed:

- 0 – vLookUp searches for an exact match; **v** can be a search pattern containing "?" (any single character) and "*" (any string); to search for a literal "?" or "*", place a tilde (~) before it,
- 1 – if an exact match is not found, vLookUp searches for the largest value that is not greater than **v**,
- -1 – if an exact match is not found, vLookUp searches for the smallest value that is not smaller than **v**.

If **type** is omitted, it is assumed to be 1. If the match is not found, the function returns the #N/A! error value.

=vLookUp(2, {1, "a"; 2, "b"; 3, "c"}, 2, 0) → "b"
=vLookUp(2.5, {1, "a"; 2, "b"; 3, "c"}, 2, -1) → "c"
=vLookUp(2.5, {1, "a"; 2, "b"; 3, "c"}, 2, 1) → "b"
=vLookUp("*bc??", {"abc", 1; "abcde", 2; "ac", 3}, 2, 0) → 2

---

unique(vector, [type])

Searches **vector** (a one-column array) for unique values.

If **type** is 0, the **unique** function returns a vector of indices to the subsequent unique values found in **vector**.

If **type** is 1, the **unique** function returns a vector of the unique values found in **vector**.

If **type** is 2, the **unique** function returns a two-column array. The first column contains the unique values, and the second contains the number of occurrences of the corresponding value.

If **type** is omitted, it is assumed to be 1.

=unique({2; 3; 1; 2; 5; 3; 1}, 1) → {1; 2; 3; 5}
=unique({"a"; "b"; "cd"; "a"; "cd"; "b"; "d"}, 0) → {4; 2; 5; 7}
=unique({"a"; "b"; "cd"; "a"; "cd"; "b"; "d"}, 2) → {"a", 2; "b", 2; "cd", 2; "d", 1}

---

udf(python_module, function, type, arg1, arg2, ...)

Executes a function from the specified Python module. The **type** parameter specifies the type of the returned value:

- 0 – number
- 2 – text string (up to 1024 characters)
- 3 – string / block of CSV data (of any size) to be pasted, without parsing numbers, into LongText/Memo fields
- 5 – an image (binary data returned as a standard Python "memoryview" object) to be inserted into Images/Files binary fields

Additionally, you can add the following flags to the **type** parameter:

- 32 – all field ranges passed from GS-Base to Python will be saved and passed to Python as single blocks/strings of CSV data,
- 64 – same as 32, except that CSV data is created and passed instead of a 1D numeric array only if there are any non-numeric (text) fields within that record field range.

For details, please see the "Python integration" help topic.

Function Index

Logical Functions

and(x1, x2, ...)

Returns the logical AND of all its arguments. All numbers are rounded to the nearest integers. Positive values are treated as **true**, and 0 is treated as **false**. Text strings are converted to numbers. Negative numbers, or text strings that

cannot be converted to numbers, cause errors. Empty cells are ignored. If there are no non-empty cells, the function returns the #N/A! error value.

=and(1, 2, true, "10") → 1
=and({true, true, 1, 1}, {true, false}) → 0

---

false()

Returns 0.

---

if(testValue, ifTrueValue, ifFalseValue)

If **testValue** is a positive number, the function returns the **ifTrueValue** argument. If it is 0, the **ifFalseValue** argument is returned. Negative values cause errors. If **testValue** is not an integer, it is rounded to the nearest integer.

=if((1 > 0)*(2 > 0), {1, 2}, false) → {1, 2}
=if("ab" > "a", 1, #N/A!) → 1

---

not(x)

Returns the logical negation of **x**.

=not("ab" > "a") → 0

---

or(x1, x2, ...)

Returns the logical OR of all its arguments. All numbers are rounded to the nearest integers. Positive values are treated as **true**, and 0 is treated as **false**. Text strings are converted to numbers. Negative numbers, or text strings that cannot be converted to numbers, cause errors. Empty cells are ignored. If there are no non-empty cells, the function returns the #N/A! error value.

=or(false, "10") → 1
=or({0, , , 1}, {1, false}) → 1

true()

Returns 1.

Function Index

Engineering Functions

besselJ(x, n)

Returns the Bessel function of the n-th order for **x**.

=besselJ(1.9, 2) → 0.32992572769239

besselY(x, n)

Returns the Bessel/Weber function of the n-th order for **x**.

=besselY(2.5, 1) → 0.14591813796679

bin2dec(n)

Converts **n**, representing a binary number, to a decimal number. The **n** argument can contain up to 10 characters. The most significant bit in **n** determines its sign. Negative numbers are represented in two's complement format.

=bin2dec("1100100") → 100
=bin2dec("1111111111") → -1

---

bin2hex(n, [l])

Converts **n**, representing a binary number, to a string representing a hexadecimal number consisting of **l** characters. The **n** argument can contain up to 10 characters. The most significant bit in **n** determines its sign. Negative numbers are represented in two's complement format. The **l** argument must be in the range [0, 10]. If it is omitted, it is assumed to be 10.

=bin2hex(11111011, 4) → "00FB"
=bin2hex("1111111111",) → "FFFFFFFFFF"

---

bin2oct(n, [l])

Converts **n**, representing a binary number, to a string representing an octal number consisting of **l** characters. The **n** argument can contain up to 10 characters. The most significant bit in **n** determines its sign. Negative numbers are represented in two's complement format. The **l** argument must be in the range [0, 10]. If it is omitted, it is assumed to be 10.

=bin2oct(1001, 3) → "011"
=bin2oct(1111111111,) → "7777777777"

---

complex(x, y, [i])

Converts real and imaginary coefficients into a string representing a complex number. The optional **i** argument specifies the suffix of the imaginary **y** coefficient. The default value of **i** is "i".

=complex(1, 2,) → "1 + 2i"
=complex(3.5, 4.6, "j") → "3.5 + 4.6j"

---

dec2bin(n, [l])

Converts **n** to a string representing a binary number consisting of **l** characters. The **n** argument must be in the range [-512, 511]. The most significant bit in **n** determines its sign. Negative numbers are represented in two's complement format. The **l** argument must be in the range [0, 10]. If it is omitted, it is assumed to be 10.

=dec2bin(9, 4) → "1001"
=dec2bin(-1,) → "1111111111"

---

dec2hex(n, [l])

Converts **n** to a string representing a hexadecimal number consisting of **l** characters. The most significant bit in **n** determines its sign. Negative numbers are represented in two's complement format. The **l** argument must be in the range [0, 10]. If it is omitted, it is assumed to be 10.

=dec2hex(100, 4) → "064"
=dec2hex(-1,) → "FFFFFFFFFF"

---

dec2oct(n, [l])

Converts **n** to a string representing an octal number consisting of **l** characters. The **n** argument must be in the range [-536870912, 536870911]. The most significant bit in **n** determines its sign. Negative numbers are represented in two's complement format. The **l** argument must be in the range [0, 10]. If it is omitted, it is assumed to be 10.

=dec2oct(58, 3) → "072"
=dec2oct(-1,) → "7777777777"

---

delta(x, [y])

Returns 1 if **x** = **y**, 0 otherwise. If **y** is omitted, it is assumed to be 0.

=delta(12, 11) → 0
=delta(0,) → 1

---

erf(x1, [x2])

Returns the value of the error function integrated between the specified limits. If the **x2** argument is omitted, the integration is performed between 0 and **x1**.

=erf(0.745,) → 0.70792891807065

---

erfc(x)

Returns the value of the complementary error function integrated between **x** and infinity.

=erfc(1) → 0.15722921001143

---

getStep(x, [y])

Returns 1 if **x** > **y**, 0 otherwise. The default value of **y** is 0.

=getStep(1.4,) → 1

---

hex2bin(n, [l])

Converts **n**, representing a hexadecimal number, to a string representing a binary number consisting of **l** characters. The **n** argument can contain up to 10 characters and must be in the range [-512, 511]. The most significant bit in **n** determines its sign. Negative numbers are represented in two's complement format. If **l** is omitted, it is assumed to be 10.

=hex2bin("F", 8) → "00001111"
=hex2bin("FFFFFFFFFF",) → "1111111111"

---

hex2dec(n)

Converts **n**, representing a hexadecimal number, to a decimal number. The **n** argument can contain up to 10 characters. The most significant bit in **n** determines its sign. Negative numbers are represented in two's complement format.

=hex2dec("A5") → 165
=hex2dec("FFFFFFFFFF") → -1

---

hex2oct(n, [l])

Converts **n**, representing a hexadecimal number, to a string representing an octal number consisting of **l** characters. The **n** argument can contain up to 10 characters and must be in the range [-536870912, 536870911]. The most significant bit in **n** determines its sign. Negative numbers are represented in two's complement format. If **l** is omitted, it is assumed to be 10.

=hex2oct("F", 3) → "017"
=hex2oct("FFFFFFFFFF",) → "7777777777"

---

imAbs(z)

Returns the absolute value of a given complex number. The **z** argument represents a complex number in the form "x + yi", x, or "yi".

=imAbs("5 + 12i") → 13

---

imaginary(z)

Returns the imaginary coefficient of a given complex number. The **z** argument represents a complex number in the form "x + yi", x, or "yi".

=imaginary("5 + 12i") → 12

---

imArgument(z)

Returns the angle t such that:

z = |z| · (cos(t) + sin(t)i)

The **z** argument represents a complex number in the form "x + yi", x, or "yi".

=imArgument("3 + 4i") → 0.92727521800161

---

imConjugate(z)

Returns the complex conjugate of **z**. The **z** argument represents a complex number in the form "x + yi", x, or "yi".

=imConjugate("3 + 4i") → "3 - 4i"

---

imCos(z)

Returns the cosine of **z**. The **z** argument represents a complex number in the form "x + yi", x, or "yi".

=imCos("1 + 1i") → "0.83373002513115 + 0.98889770576287i"

---

imDiv(z1, z2)

Divides **z1** by **z2** and returns the result. The **z1** and **z2** arguments represent complex numbers in the form "x + yi", x, or "yi".

=imDiv("1 + 2i", "2 + 1i") → "0.8 + 0.6i"

---

imExp(z)

Raises e to the power of **z**. The **z** argument represents a complex number in the form "x + yi", x, or "yi".

=imExp("1 + 1i") → "1.43869393991589 + 2.28735528717884i"

---

imLn(z)

Returns the natural logarithm of **z**. The **z** argument represents a complex number in the form "x + yi", x, or "yi".

=imLn("3 + 4i") → "1.6094379124341 + 0.92729521800161i"

---

imLog10(z)

Returns the base-10 logarithm of **z**. The **z** argument represents a complex number in the form "x + yi", x, or "yi".

=imLog10("3 + 4i") → "0.69897000433601 + 0.40271919627337i"

---

imLog2(z)

Returns the base-2 logarithm of **z**. The **z** argument represents a complex number in the form "x + yi", x, or "yi".

=imLog2("3 + 4i") → "2.32192809488732 + 1.33780421245095i"

---

imPower(z, n)

Raises **z** to the power of **n** and returns the result. The **z** argument represents a complex number in the form "x + yi", x, or "yi".

=imPower("2 + 3i", 3) → "-46 + 9i"

---

imProduct(z1, z2, ...)

Multiplies all the specified arguments, which can be complex numbers or arrays of complex numbers in the form "x + yi", x, or "yi". Empty cells in arrays are skipped.

=imProduct("3 + 4i", "5 - 3i") → "27 + 11i"

---

imReal(z)

Returns the real coefficient of **z**. The **z** argument represents a complex number in the form "x + yi", x, or "yi".

=imReal("3 + 4i") → 3

---

imSin(z)

Returns the sine of **z**. The **z** argument represents a complex number in the form "x + yi", x, or "yi".

=imSin("3 + 4i") → "3.85373803791938 - 27.0168132580039i"

---

imSqrt(z)

Returns the square root of **z**. The **z** argument represents a complex number in the form "x + yi", x, or "yi".

=imSqrt("1 + i") → "1.09868411346781 + 0.45508986056223i"

---

imSub(z1, z2)

Returns the difference between **z1** and **z2**. The **z1** and **z2** arguments represent complex numbers in the form "x + yi", x, or "yi".

=imSub("10 + 5i", "9 + 4i") → "1 + 1i"

---

imSum(z1, z2, ...)

Adds all the specified arguments, which can be complex numbers or arrays of complex numbers in the form "x + yi", x, or "yi".

=imSum("1 + 1i", "2 + 2i", {"1 + 1i", "1 + 1i"}, 1, "1i") → "6 + 6i"

---

oct2bin(n, [l])

Converts **n**, representing an octal number, to a string representing a binary number consisting of **l** characters. The **n** argument can contain up to 10 characters and must be in the range [-512, 511]. The most significant bit in **n** determines its sign. Negative numbers are represented in two's complement format. The **l** argument must be in the range [0, 10]. If it is omitted, it is assumed to be 10.

=oct2bin(3, 3) → "011"
=oct2bin(7777777000,) → "1000000000"

---

oct2dec(n)

Converts **n**, representing an octal number, to a decimal number. The **n** argument can contain up to 10 characters. The most significant bit in **n** determines its sign. Negative numbers are represented in two's complement format.

=oct2dec(54) → 44
=oct2dec(7777777777) → -1

---

oct2hex(n, [l])

Converts **n**, representing an octal number, to a string representing a hexadecimal number consisting of **l** characters. The **n** argument can contain up to 10 characters. The most significant bit in **n** determines its sign. Negative numbers are represented in two's complement format. The **l** argument must be in the range [0, 10]. If it is omitted, it is assumed to be 10.

=oct2hex(100, 4) → "0040"
=oct2hex(7777777777,) → "FFFFFFFFFF"

Function Index

Optimization/Solver Functions

LProg(A, s, b, c, vector, [epsilon], [BigM])

1. Purpose

Finds one or more solutions for a linear programming problem with **m** constraints of the form:

(1) $\Sigma_{j=1..n}(x[j]\cdot a[i,j]) \leq b[i]$
(2) $\Sigma_{j=1..n}(x[j]\cdot a[i,j]) = b[i]$
(3) $\Sigma_{j=1..n}(x[j]\cdot a[i,j]) \geq b[i]$

and a maximized objective function:

$\Sigma_{j=1..n}(x[j]\cdot c[j])$

2. Arguments

- **A** — m×n matrix of coefficients a[i,j]
- **b** — m-element column vector
- **c** — n-element column vector
- **s** — m-element column vector specifying constraint types:
  - s[i] = 1 → ≤
  - s[i] = 0 → =
  - s[i] = −1 → ≥
- **vector** — return mode:
  - 0 → number of optimum vectors found

    - i > 0 → i-th optimum vector
- **epsilon** — precision threshold; coefficients with absolute value < epsilon are treated as 0 (default: 1e−8)
- **BigM** — large artificial-variable coefficient used in the Big-M method; if omitted, an appropriate Big-M value is chosen automatically

3. Behavior

- All returned x[i] values are non-negative.
- If the model allows negative variables, each such variable must be represented by a pair of non-negative variables.
- To minimize the objective function, negate it and use the default maximization.
- If the objective function is unconstrained, LProg returns **#N/A!**.
- If the constraints are inconsistent and no solution exists, LProg returns **#NULL!**.

4. Example

Initial constraints:

$2 \cdot x1 + x2 + 2 \cdot x3 \leq 20$
$3 \cdot x1 - x2 + x3 = 10$

Objective function:

$3 \cdot x1 + x2 - x3 \rightarrow \max$

Formula:

=LProg({2, 1, 1; 1, 1, 3}, {1; 0}, {5; 10}, {4; 2; 2}, 1,,)

returns **{1; 0; 3}**

LProgBin(A, s, b, c, bin, vector, [epsilon], [BigM])

1. Purpose

Finds one or more solutions for a linear programming problem with constraints of the same form as LProg, where some or all variables x[i] are required to be binary (0 or 1).

2. Arguments

- **A**, **b**, **c**, **s**, **epsilon**, **BigM** — as in LProg.
- **bin** — n-element column vector defining variable types:
    - bin[i] = 1 → $x[i] \in \{0, 1\}$
    - bin[i] = 0 → $x[i] \geq 0$
- **vector** — return mode:

- 0 → number of optimum vectors found
- i > 0 → i-th optimum vector

3. Behavior

- All returned x[i] values are non-negative; binary variables are restricted to 0 or 1.
- To minimize the objective function, negate it and use the maximization procedure.
- If the objective function is unconstrained, LProgBin returns **#N/A!**.
- If the constraints are inconsistent and no solution exists, LProgBin returns **#NULL!**.

4. Example

Initial constraints:

2.5·x1 + 1.5·x2 + x3 ≤ 15
x1 + 1.5·x2 + 3·x3 = 10

Objective function:

x1 + x2 + 2·x3 → max

Formula:

=LProgBin({2.5, 1.5, 1; 1, 1.5, 3}, {1; 0}, {15; 10}, {1; 1; 2}, {1; 0; 1}, 1,,)

returns **{1; 4; 1}**

LProgInt(A, s, b, c, int, vector, [epsilon], [BigM])

1. Purpose

Finds one or more solutions for a linear programming problem with constraints of the same form as LProg, where some or all variables x[i] are required to be integers.

2. Arguments

- **A**, **b**, **c**, **s**, **epsilon**, **BigM** — as in LProg.
- **int** — n-element column vector defining variable types:
  - int[i] = 1 → x[i] must be integer
  - int[i] = 0 → x[i] ≥ 0
- **vector** — return mode:
  - 0 → number of optimum vectors found
  - i > 0 → i-th optimum vector

3. Behavior

- All returned x[i] values are non-negative; selected variables must be integers.
- To minimize the objective function, negate it and use the maximization procedure.
- If the objective function is unconstrained, LProgInt returns **#N/A!**.
- If the constraints are inconsistent and no solution exists, LProgInt returns **#NULL!**.

4. Example

Initial constraints:

2.5·x1 + 1.5·x2 + x3 ≤ 15
x1 + 1.5·x2 + 3·x3 = 10

Objective function:

x1 + x2 + 2·x3 → max

Formula:

=LProgInt({2.5, 1.5, 1; 1, 1.5, 3}, {1; 0}, {15; 10}, {1; 1; 2}, {1; 0; 1}, 1,,)

returns **{4; 0; 2}**

QProg(A, s, b, p, C, vector, [epsilon])

1. Purpose

Finds one or more solutions for a quadratic programming problem with **m** constraints:

(1) $\Sigma_{j=1..n}(x[j]\cdot a[i,j]) \le b[i]$
(2) $\Sigma_{j=1..n}(x[j]\cdot a[i,j]) = b[i]$
(3) $\Sigma_{j=1..n}(x[j]\cdot a[i,j]) \ge b[i]$

and a maximized objective function:

$p^T\cdot x - x^T\cdot C\cdot x$

where **p** is an n-element column vector and **C** is a positive-definite n×n matrix representing a quadratic form.

2. Arguments

- **A** — m×n matrix of coefficients a[i,j]
- **b** — m-element column vector
- **p** — n-element column vector

- **C** — n×n positive-definite matrix
- **s** — m-element column vector specifying constraint types:
  - s[i] = 1 → ≤
  - s[i] = 0 → =
  - s[i] = −1 → ≥
- **vector** — return mode:
  - 0 → number of optimum vectors found
  - i > 0 → i-th optimum vector
- **epsilon** — precision threshold (default: 1e−8)

3. Behavior

- All x[i] are non-negative.
- If negative variables are needed, each must be represented by a pair of non-negative variables.
- To minimize the objective function, change its sign and use the maximization procedure.
- If the objective function is unconstrained, QProg returns **#N/A!**.
- If the constraints are inconsistent and no solution exists, QProg returns **#NULL!**.

4. Example

Constraints:

$x1 + 2 \cdot x2 \le 10$
$x1 + x2 \le 9$
$x1 \ge 0, x2 \ge 0$

Objective function:

$f(x1, x2) = 10 \cdot x1 + 25 \cdot x2 - 10 \cdot x1^2 - x2^2 - 4 \cdot x1 \cdot x2 \rightarrow \max$

Data:

A = {1, 2; 1, 1}
s = {1; 1}
b = {10; 9}
p = {10; 25}
C = {10, 2; 2, 1}

Formula:

=QProg({1, 2; 1, 1}, {1; 1}, {10; 9}, {10; 25}, {10, 2; 2, 1}, 1,)

returns **{0; 5}**

Function Index

Matrix and Equation System Functions

mtxBkSubst(U, B)

Performs backsubstitution for a given upper triangular **n × n U** matrix and an **n × l B** matrix containing **l** vectors of right-hand values.

The **mtxBkSubst** function returns **l** column vectors that are—typically—solutions of the equation **L·U·x = B**.

Example:
=mtxBkSubst({1, 0.5, 0.3333333; 0, 0.28867507685978, 0.2886752500649; 0, 0, 0.07453530110679}, {0.58333; -0.25979035258533; -0.03740870232733}) returns {0.94965127674073; -0.39804764314169; -0.50189241569889}

mtxChl(A, B, [iterations], result)

Performs Cholesky decomposition of a given symmetric, positively determined **n × n A** matrix (**x^T·A·x > 0** for each **x ≠ 0**). The **mtxChl** function can be used to:

1. solve a linear equation set **A·x = B**, where **A** is an **n × n** matrix, **x** is an **n**-element column vector and **B** is an **n × l** matrix,
2. find the upper triangular **U** matrix such that **U^T·U = A**,
3. find the inverse and determinant of **A**.

The **A** and **B** arguments specify the matrices used in the **A·x = B** equation. The **B** argument is used only when **result = 1**. If **B** has **l** columns (**l > 1**), the procedure will use each column as different right-hand values and it will return **l** solution (column) vectors.

The **iterations** argument specifies the number of additional improving iterations that should be performed when finding the **x** vector(s). The default value is **0**.

The **result** argument specifies what should be returned:

If **result = 1**, the function returns **l** solution vectors of the **A·x = B** equation system.
If **result = 2**, the function returns an upper triangular **n × n U** matrix such that $\mathbf{U^T \cdot U = A}$.
If **result = 3**, the function returns a transposed upper triangular **n × n** $\mathbf{U^T}$ matrix such that $\mathbf{U^T \cdot U = A}$.
If **result = 4**, the function returns the inverse of **A**.
If **result = 5**, the function returns the determinant of **A**.

Examples:
=mtxChl({1, 0.5, 0.3333333; 0.5, 0.3333333, 0.25; 0.33333333, 0.25, 0.2}, {0.58333; 0.21667; 0.11666}, 3, 1) returns {0.9496504220468; -0.3980425149853; -0.5018975438522}

=mtxChl({1, 0.5, 0.3333333; 0.5, 0.3333333, 0.25; 0.33333333, 0.25, 0.2},,,2) returns {1, 0.5, 0.3333333; 0, 0.28867507685978, 0.2886752500649; 0, 0, 0.07453530110679}

---

mtxDiag(n, x) / mtxDiag(n, A)

Creates and returns an **n × n** diagonal matrix.

The first version of the function uses the **x** value for all diagonal elements. The second version uses the subsequent elements of the **A** vector/matrix to set the diagonal elements in the returned matrix.

Examples:
=mtxDiag(3, 1) returns {1, 0, 0; 0, 1, 0; 0, 0, 1}

=mtxDiag(3, {1, 2, 3}) returns {1, 0, 0; 0, 2, 0; 0, 0, 3}

---

mtxFwSubst(L, B)

Performs forward substitution for a given lower triangular **n × n L** matrix and an **n × l B** matrix containing **l** vectors of right-hand values.

The **mtxFwSubst** function returns **l** (column) vectors.

Example:
=mtxFwSubst({1, 0, 0; 0.5, 0.28867507685978, 0; 0.3333333, 0.2886752500649, 0.07453530110679}, {0.58333; 0.21667; 0.11666}) returns {0.58333; -0.25979035258533; -0.03740870232733}

---

mtxGauss(A, B, [iterations], [scaling], result)

Performs Gauss transformation with partial pivoting, optional improving iterations and scaling. The **mtxGauss** function can be used to:

1. solve a linear equation set **A·x = B**, where **A** is an **n × n** matrix, **x** is an **n**-row column vector and **B** is an **n × l** matrix,
2. find the LU decomposition of **A**,
3. find the inverse of **A**,
4. obtain the determinant of **A**.

The **A** and **B** arguments specify the matrices used in the **A·x = B** equation. The **B** argument is used only when **result = 1**. If **B** has **l** columns (**l > 1**), the procedure will use each column as different right-hand values and it will return **l** solution vectors.

The **iterations** argument specifies the number of improving iterations that should be performed when finding the **x** vector(s). The default value is **0**.

The **scaling** argument specifies whether the initial coefficients should be scaled (to minimize roundoff problems if the equations are not well balanced). If **scaling = 1**, **mtxGauss** will scale the coefficients. If **scaling = 0**, no scaling will be performed. The default value is **0**.

The **result** argument specifies what should be returned:

If **result = 1**, the function solves **A·x = B** and returns **l** solution vectors, where **l** is the number of columns in **B**.
If **result = 2**, the function returns an **n × n** lower triangular matrix **L** such that **L·U = A'** where **A'** is a row-wise permutation of **A**.
If **result = 3**, the function returns an **n × n** upper triangular matrix **U** such that **L·U = A'** where **A'** is a row-wise permutation of **A**.
If **result = 4**, the function returns an **n × n** permutation matrix **P** such that **P·A = A'** where **A'** is obtained from **A** by exchanging rows in the partial pivoting procedure.
If **result = 5**, the function returns the inverse of **A**.
If **result = 6**, the function returns the determinant of **A**.

Initial equation system:
2·x1 − x2 + 3·x3 = 6
1·x1 + 3·x2 + x3 = 8
4·x1 + x2 + x3 = 2

Examples:
=mtxGauss({2, -1, 3; 1, 3, 1; 4, 1, 1}, {6; 8; 2},,,1) returns {-0.75; 1.875; 3.125}

=mtxGauss({2, -1, 3; 1, 3, 1; 4, 1, 1},,,,6) returns -32

=mtxGauss({2, -1, 3; 1, 3, 1; 4, 1, 1},,,,5) returns {-0.0625, -0.125, 0.3125; -0.09375, 0.3125, -0.03125; 0.34375, 0.1875, -0.21875}

---

mtxLSC(A, b, E, d, [minMaxRatio], result)

Solves a given over-determined equation set **A·x ≈ b** with constraints in the form of **E·x = d**, where **E** is an **l × n** matrix and **l < n**.

The **A** argument is an **m × n** matrix and the **b** argument is a column vector with **m** rows. The **E** argument is an **l × n** matrix and the **d** argument is a column vector with **l** rows.

The **minMaxRatio** argument specifies the minimum allowable ratio between the minimum and maximum singular value below which a given singular value will be treated as 0.

The **result** argument specifies what should be returned:

If **result = 1**, the function returns the **n**-row column vector **x**.
If **result = 2**, the function returns the rest value **r** such that $\mathbf{(A \cdot x - b)^T \cdot (A \cdot x - b) = r}$.

If the SVD decomposition is not convergent, the **#NUM!** error value is returned.

Examples:
=mtxLSC({1, 2; 2, 3; 7, 1}, {1; 2; 3}, {1, 3}, {2},,1) returns {0.2; 0.6}

=mtxLSC({1, 2; 2, 3; 7, 1}, {1; 2; 3}, {1, 3}, {2},,2) returns 0.319512195121195

---

mtxLSW(A, b, G, [minMaxRatio], result)

Solves a given over-determined equation set **A·x ≈ b** with weights specified by the **G** matrix so that the minimized function has the form:

$\mathbf{r = (A \cdot x - b)^T \cdot G \cdot (A \cdot x - b) \rightarrow min}$

The **A** argument is an **m × n** matrix and the **b** argument is a column vector with **m** rows.

The **G** argument is an **n × n** matrix with weights. If **G = I**, the function is an equivalent of **mtxSVD**.

The **minMaxRatio** argument specifies the minimum allowable ratio between the minimum and maximum singular value below which a given singular value will be treated as 0.

The **result** argument specifies what should be returned:

If **result = 1**, the function returns the **n**-row column vector **x**.
If **result = 2**, the function returns the rest value **r** such that $\mathbf{(A \cdot x - b)^T \cdot G \cdot (A \cdot x - b) = r}$.

If the SVD decomposition is not convergent, the **#NUM!** error value is returned.

Examples:
=mtxLSW({1, 2; 2, 3; 7, 1}, {1; 2; 3}, {1,0,0;0,1,0;0,0,1},,1) returns {0.37476459510358; 0.38418079096045}

=mtxLSW({1, 2; 2, 3; 7, 1}, {1; 2; 3}, {1, 0.5 , 0.2; 0.5, 1, 0.7; 0.2, 0.7, 1},,1) returns {0.38927943760984; 0.3585237258348}

---

mtxMlt(v1, v2, ...)

Multiplies the subsequent specified matrices. The number of rows in each next matrix and the number of columns in the previous result must be the same.

Example:
=mtxMlt({1;2;3}, {1, 2, 3}, {2; 2; 2}) returns {12; 24; 36}

---

mtxRand(n, [seed1], [seed2])

Generates a column vector (a one-column matrix) of **n** random floating point numbers from the range **(0, 1)**. The function uses two combined MLCG generators to create a series of about **2.3×10^{18}** numbers.

The **n** argument specifies the size of the returned vector.

The **seed1** and **seed2** arguments determine the starting numbers for the first and the second generator.

If both those values are omitted, the first call to **mtxRand** will always start from the same hard-coded values of **seed1** and **seed2** and subsequent calls will use the previously generated values, returning partial series occurring one after another.

If both **seed1** and **seed2** are specified, **mtxRand** will always be generating one and the same partial series.

To generate two or more independent partial series that do not occur one after another, specify a fixed **seed1** value and skip the **seed2** argument.

Example:
=mtxRand(4, 10000, 25000) returns {0.71261246808435; 0.28457538373252; 0.42105025462307; 0.93072516522912}

---

mtxRand2(n, [seed1], [seed2], [type], [v1], [v2])

Generates a column vector (a one-column matrix) of random floating point numbers for the specified distribution type.

The **n** argument specifies the size of the returned vector.

The **seed1** and **seed2** arguments determine the starting numbers for the first and the second MLCG generator.

If both those values are omitted, the first call to **mtxRand2** will always start from the same hard-coded values of **seed1** and **seed2** and subsequent calls will use the previously generated values, returning partial series occurring one after another.

If both **seed1** and **seed2** are specified, **mtxRand2** will always be generating one and the same partial series.

To generate two or more independent partial series that do not occur one after another, specify a fixed **seed1** value and skip the **seed2** argument.

The **type** argument specifies the distribution type:

0 – uniform (0, 1)
1 – normal with the **v1** mean and the **v2** standard deviation
2 – exponential with the **v1** mean
3 – Poisson with the **v1** mean
4 – Bernoulli with the **v1** probability
5 – geometric with the **v1** probability

If **type** is omitted, it's assumed to be **0**.

If a given distribution type doesn't require the **v1** and/or **v2** arguments, they should be omitted.

Example:
=mtxRand2(4, 10000, 25000,,,) returns {0.71261246808435; 0.28457538373252; 0.42105025462307; 0.93072516522912}

---

mtxSeries(n, x, [step])

Generates a column vector (a one-column matrix) of **n** subsequent numbers or generic date/time strings.

The **x** argument specifies the first element of the series. This can be a number or date/time string.

The **step** argument specifies the value by which the subsequent numbers/dates are incremented. If it's omitted, it's assumed to be **1** for numeric series and **"P1D"** (one day) for date series.

The general form of the date/time period is:
[+|-]PnYnMnDTnHnMnS.nnn

Examples:
P1Y2M10DT11H5M4S.355 represents a period of 1 year, 2 months, 10 days, 11 hours, 5 minutes, 4 seconds, 355 thousandths.
PT12H7M represents a period of twelve hours and seven minutes.

Examples:
=mtxSeries(5, 1, 0.1) returns {1; 1.1; 1.2; 1.3; 1.4}

=mtxSeries(5, "2005-12-01", "P1D") returns {"2005-12-01"; "2005-12-02"; "2005-12-03"; "2005-12-04"; "2005-12-05"}

=mtxSeries(5, "12:50:00", "PT2M") returns {"12:50:00"; "12:50:02"; "12:50:04"; "12:50:06"; "12:50:08"}

---

mtxSVD(A, B, [minMaxRatio], result)

Performs SVD decomposition of a given matrix **A[m,n]**. The **mtxSVD** function can be used to:

1. solve an over-determined equation system **A·x ≈ B** where **B** is an **m × l** matrix containing **l** right-hand values vectors,
2. find the number of non-zero singular values,
3. find **l** rest values **r[i]** such that $(A \cdot x - B_i)^T \cdot (A \cdot x - B_i) = r[i]$ where B_i is the i-th column of **B**,
4. find the (pseudo-)inverse A^+ of **A**,
5. find the **U[m,n]** matrix, **S[m,n]** matrix and **V[n,n]** matrix such that $U \cdot S \cdot V^T = A$.

The **A** and **B** arguments are matrices used in the equation system **A·x ≈ B**. The **B** argument is ignored for **result** values other than **1** or **3**.

The **minMaxRatio** argument specifies the minimum allowable ratio between the minimum and maximum singular value below which a given singular value will be treated as 0. The default value (and the smallest possible value) is **1.0e−15**.

The **result** argument specifies what should be returned:

If **result = 1**, the function returns **l n**-row column vectors that are solutions of **A·x ≈ B**.
If **result = 2**, the function returns a column vector of **l** rest values.
If **result = 3**, the function returns the number of non-zero singular values.
If **result = 4**, the function returns the inverse A^+ of **A**.
If **result = 5**, **6** or **7**, the function returns respectively the **U**, **S** and **V** matrices described above.

If the SVD decomposition is not convergent, the **#NUM!** error value is returned.

Examples:
=mtxSVD({1, 2; 2, 3; 7, 1}, {1; 2; 3},,1) returns {0.37476459510358; 0.38418079096045}

=mtxSVD({1, 2; 2, 3; 7, 1}, {1; 2; 3},,3) returns 0.030131826742

=mtxSVD({1, 2; 2, 3; 7, 1},,,6) returns {7.68114574786861, 0; 0, 3; 0, 0}

---

mtxT(A)

Transposes a given matrix and returns the result.

Example:
=mtxT({1, 2, 3; 4, 5, 6}) returns {1, 4; 2, 5; 3, 6}

Using the built-in UDF() (user-defined-function) function you can call any Python modules/functions in formulas in GS-Base.
This can help you significantly increase GS-Base functionality thanks to the large number of various Python modules and libraries, including, for example, science, finance and graphics.

From GS-Base you can pass to Python functions any type of parameters: (double precision) numbers, text strings, ranges of field values as numeric arrays/matrices, and automatically saved blocks of CSV data from field ranges.

GS-Base can accept from Python functions any type of data it supports in formulas: (double precision) numbers, text strings, numeric arrays/matrices, CSV data blocks, and images/graphics/charts to be inserted in the LongText, Images/Files or Code binary fields.

The UDF() Function

UDF(module, function, type, parameter1, parameter2, ...)

- module - Python module name with full path, for example `c:\my_modules\some_functions.py`
- function - function name as it appears in the above module, for example `my_function`
- type - an index (0-5) specifying what type of data GS-Base should expect from the Python procedure. Possible values are:
  - 0 - a number
  - 1 - a matrix/array of numbers
  - 2 - a text string (up to 8192 characters)
  - 3 - a string/block of CSV data (of any size) to be inserted in the LongText/Code field

- 5 - an image (binary data returned as standard Python "memoryview" binary data) to be inserted in the Images/Files field

(Type 4 is reserved for the corresponding GS-Calc UDF() implementation and does not apply in GS-Base.)

Additionally, you can add to the "type" parameter the following flags:

32 - all ranges/arrays (of field) values passed from GS-Base to Python will be saved and passed to Python as blocks/strings of CSV data
64 - same as 32, except that CSV data is created and passed instead of an array only if there are any non-numeric/text field values within that GS-Base field range.

If neither 32 nor 64 is used, if you specify a range as a parameter in the UDF() function, GS-Base will pass it as an array of numbers, ignoring any text data. Rules used to save and parse such CSV data blocks are the same as for handling regular CSV files in GS-Base.

Python modules are plain text files with the "*.py" extension. Below is an example of such a file containing six simple functions.

`e:\example.py`:

```
def function1():
   return 10

def function2(a, b):
  return a + b

def function3(text):
   return 'A new text with ' + text + ' inserted in the
middle'

def function4(csvdata):
   return csvdata

def function5(csvFilePath):
   f = open(csvFilePath, 'rb')
   file_content = f.read()
   f.close()
   return file_content

def function6(imageFilePath):
   f = open(imageFilePath, 'rb')
```

```
    file_content = f.read()
    f.close()
    return bytearray(file_content)
```

In GS-Base, the above functions can be used as follows:

1. `UDF("e:\example.py", "function1", 0)`

   takes no arguments and returns 10.

2. `UDF("e:\example.py", "function2", 0, 10, 20)`

   returns the sum of two numbers.

3. `UDF("e:\example.py", "function3", 2, "a word")`

   takes a text string, inserts it in another string, and returns the obtained new string to GS-Base.

4. `UDF("e:\example.py", "function4", 3 + 32, fields(10, 100))`

   accepts from GS-Base a CSV data block (of fields 10 to 100) and returns it back to GS-Base.
   If this formula is used with the **Insert > Generating Function** command, it can be used to populate binary LongText/Code fields sequentially in all records.

5. `UDF("e:\example.py", "function5", 3, "e:\some_csv_file.csv")`

   loads a CSV file and returns its contents. It can be used in calculated formulas to insert the returned result in plain record text fields, or such formulas can be used with the **Insert > Generating Function** command to populate binary LongText/Code fields sequentially in all records.

6. `UDF("e:\example.py", "function6", 5, "e:\some_image.jpg")`

   returns a jpeg/png/gif/bmp/tiff image loaded from the specified file and inserts it in a binary field. Returning graphics is applicable to such formulas when they are used with the **Insert > Generating Function**

command to populate binary LongText/Code fields sequentially in all records.

Related Topics

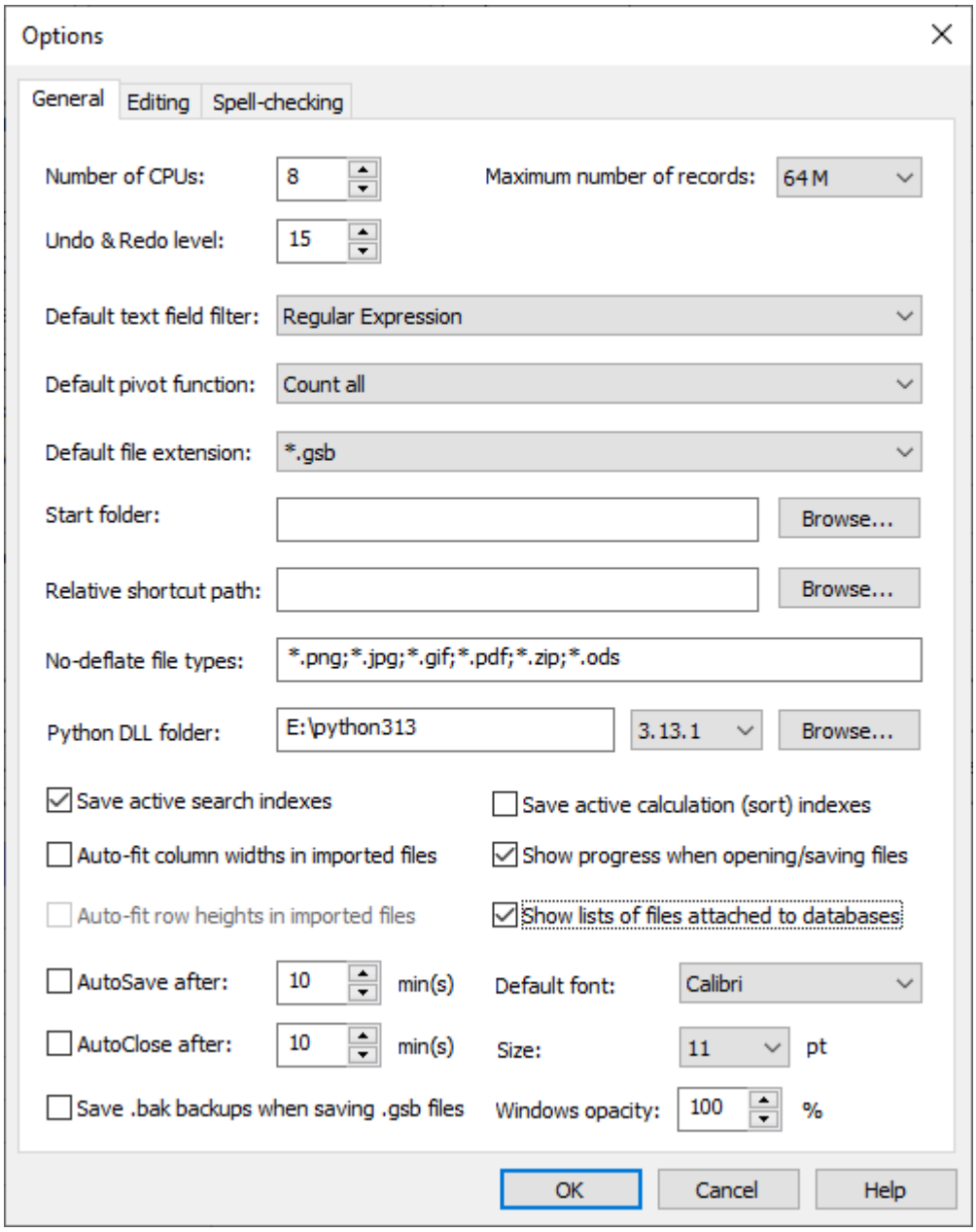
Installing Python and integrating it with GS-Base

GS-Base Help > *Installing Python and integrating it with GS-Base*

You can download the standard Python package, among others, from the Microsoft Store or directly from the Python project home page.

You must specify which Python version you would like to use with GS-Base, by selecting a given version in the **Settings > Options > General** dialog box.

Unless you already have Windows system variables pointing to the Python installation folder, in the **Settings > Options > General** dialog box you must also specify the folder where the Python core library DLL file (for example, `python39.dll` for version 3.9) is located.

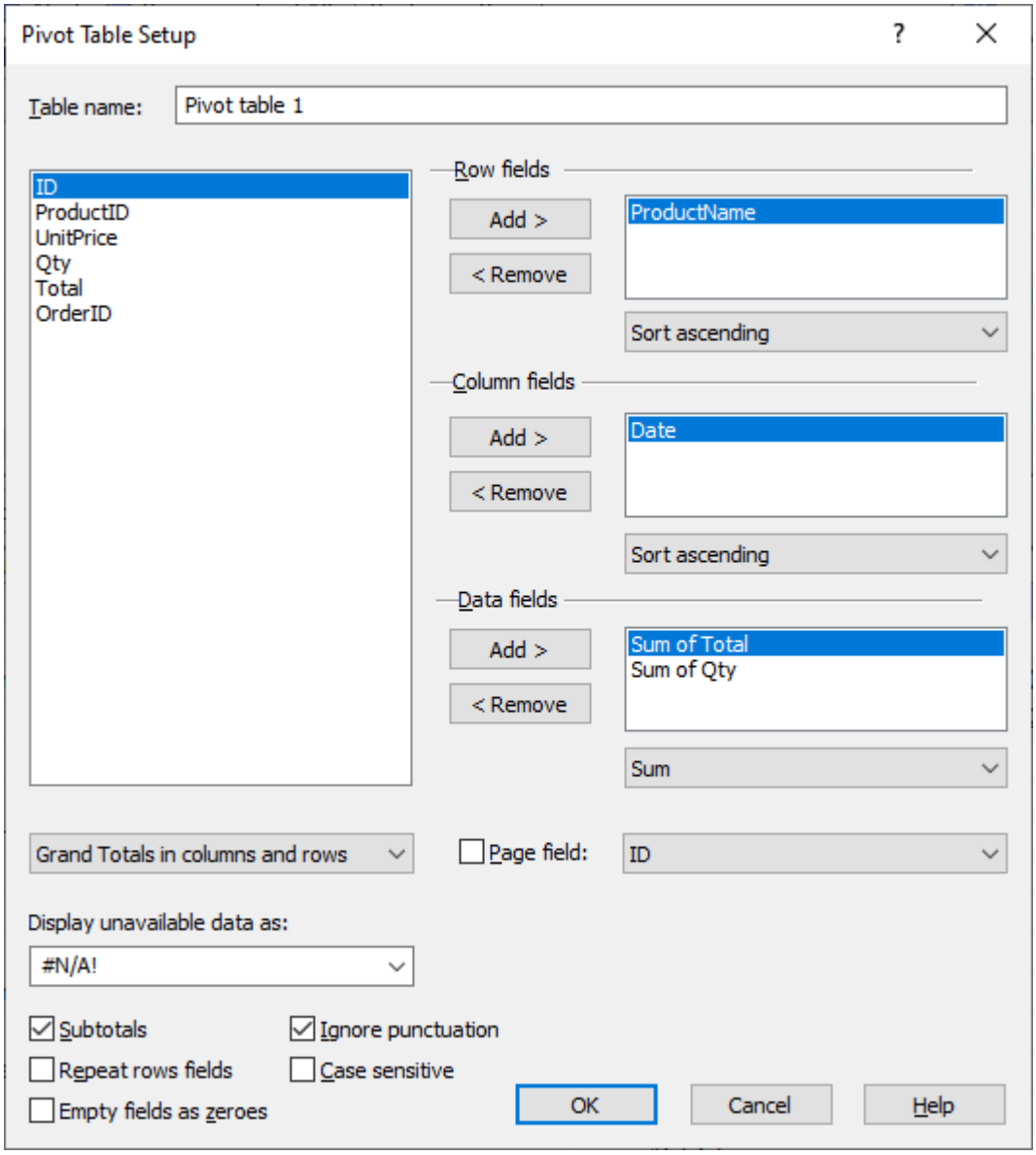


Pivot tables enable you to quickly and easily obtain and visualize various statistical information about the data stored in flat source record tables. This includes (but is not limited to) counting, summarizing, sorting, or finding partial maximum/minimum and mean values (see: Pivot Table Functions).

Tip: you can create up to 100 pivot tables for each table in the database file, with up to 256 million rows and up to 16K columns each. Once a pivot table is created, clicking the **Update** button always applies to the current filtered record set. If your current filtered pivot table category column has more unique values than the above ~16K limit:

1. Use **Tools > Find Unique Values and Frequencies** (disregard frequencies).
2. Select a range of records within the above limit and use **Tools > Show Current Selection** to "find" the selection as the record set.
3. Update the table.

To create a new pivot table, use the **View > Pivot Table View** command and, in the opened pivot table pane, click **Pivot Table > Add New Pivot Table**, or - if this is the first pivot table - simply click the **Setup** button:



Quick Start

Open the included sample "sample.zip" database and choose the "customers" table.
The records contain (among others) the "Country" and "City" fields.

To obtain the number of customers from particular countries:

1. Open the **Pivot Table Setup** dialog.
2. Choose the "Country" database field as the only **Row field**.
3. Click **OK**.

	Country	Grand Total
1	Argentina	3
2	Austria	2
3	Belgium	2
4	Brazil	9
5	Canada	3
6	Denmark	2
7	Finland	2
8	France	11
⋮		
22	**Grand Total**	**91**

To obtain the number of customers from particular countries and to find out how many of them come from given cities in that country:

1. Open the **Pivot Table Setup** dialog.
2. Choose the "Country" database field as the first **Row field**.
3. Choose the "City" database field as the second **Row field**.
4. Click **OK**.

	Country	City	Grand Total
1	Argentina	Buenos Aires	3
2	Argentina Total		3
3	Austria	Graz	1
4		Salzburg	1
5	Austria Total		2
6	Belgium	Bruxelles	1
7		Charleroi	1
8	Belgium Total		2
⋮			
91	**Grand Total**		**91**

Open the included sample "sample.zip" database and choose the "orders" table. The records contain (among others) the "ProductName" and (order) "Date" fields.

To obtain the numbers of sold products and to find out how these numbers changed over subsequent days:

1. Open the **Pivot Table Setup** dialog.
2. Choose the "ProductName" database field as the only **Row field**.
3. Choose the "Date" database field as the only **Column field**.
4. Click **OK**.

	ProductName	2011-02-27	2011-02-28	2011-03-01	2011-03-04	2011-03-05	Grand Total
1	Aniseed Syrup			1			1
2	Chef Anton's Cajun Seasoning					1	1
3	Chef Anton's Gumbo Mix		1				1
4	Gustaf's Knäckebröd	1					1
5	Queso Cabrales				1		1
6	Sir Rodney's Marmalade				1		1
7	Teatime Chocolate Biscuits		1				1
8	Tofu			1			1
9	**Grand Total**	**1**	**2**	**2**	**2**	**1**	**8**

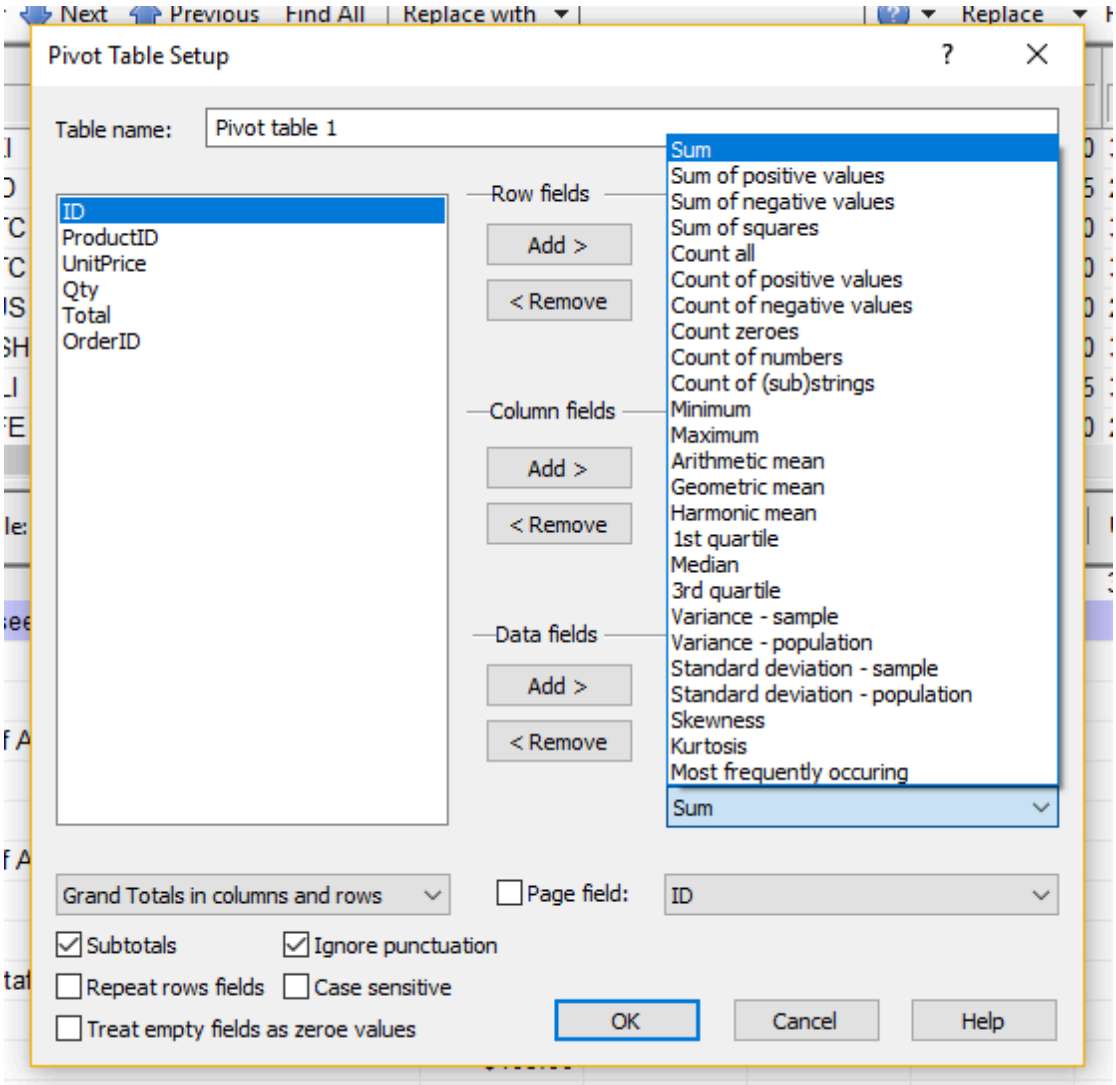
To obtain the numbers of sold products and to find out how both these numbers and the sales totals changed over subsequent days:

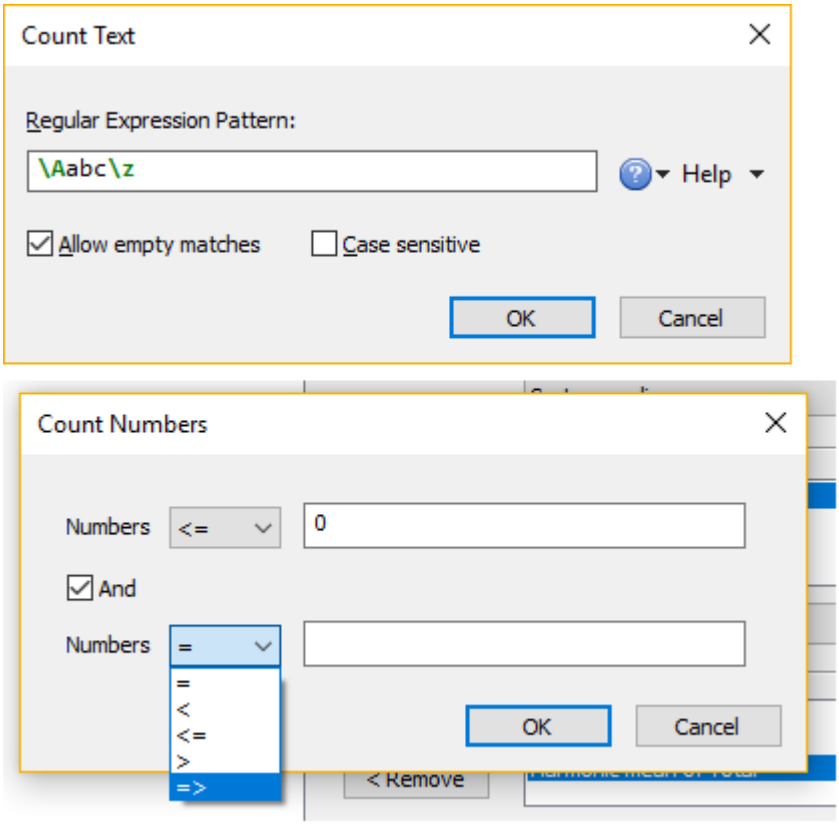
1. Open the **Pivot Table Setup** dialog.
2. Choose the "ProductName" database field as the only **Row field**.
3. Choose the "Date" database field as the only **Column field**.
4. Choose the "Total" database field as the first **Data field** and select the **Count** function for it.
5. Choose the "Qty" database field as the second **Data field** and select the **Sum** function for it.
6. Click **OK**.

	ProductName	2011-02-27	2011-02-28	2011-03-01	2011-03-04	2011-03-05	Grand Total
1	Aniseed Syrup			$20.00			$20.00
2				2			2
3	Chef Anton's Cajun Seasoning					$44.00	$44.00
4						2	2
5	Chef Anton's Gumbo Mix		$21.35				$21.35
6			1				1
7	Gustaf's Knäckebröd	$105.00					$105.00
8		5					5
9	Queso Cabrales				$63.00		$63.00
10					3		3
11	Sir Rodney's Marmalade				$81.00		$81.00
12					1		1
13	Teatime Chocolate Biscuits		$27.60				$27.60
14			3				3
15	Tofu			$23.25			$23.25
16				1			1
17	**Grand Total**	**$105.00**	**$48.95**	**$43.25**	**$144.00**	**$44.00**	**$385.20**
18		**5**	**4**	**3**	**4**	**2**	**18**

GS-Base pivot tables can use a number of "counting" functions:

- Sum
- Sum of positive values
- Sum of negative values
- Sum of squares
- Count all
- Count negative values
- Count positive values
- Count numbers
- Count (sub)strings
- Minimum
- Maximum
- Arithmetic mean
- Geometric mean
- Harmonic mean
- 1st quartile
- Median
- 3rd quartile
- Variance - sample
- Variance - population
- Standard deviation - sample
- Standard deviation - population
- Skewness
- Kurtosis
- Most frequently occurring

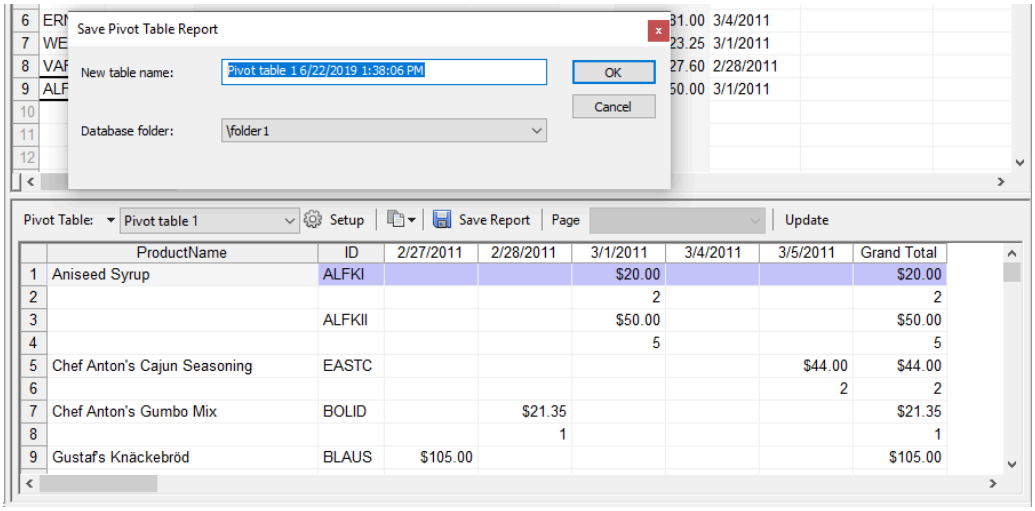

Next
Previous
Find All
Replace with
Replace
Pivot Table Setup
Table name:
Pivot table 1
ID
ProductID
UnitPrice
Qty
Total
OrderID
Row fields
Add >
< Remove
Column fields
Add >
< Remove
Data fields
Add >
< Remove
Sum
Sum of positive values
Sum of negative values
Sum of squares
Count all
Count of positive values
Count of negative values
Count zeroes
Count of numbers
Count of (sub)strings
Minimum
Maximum
Arithmetic mean
Geometric mean
Harmonic mean
1st quartile
Median
3rd quartile
Variance - sample
Variance - population
Standard deviation - sample
Standard deviation - population
Skewness
Kurtosis
Most frequently occuring
Sum
Grand Totals in columns and rows
Page field:
ID
Subtotals
Ignore punctuation
Repeat rows fields
Case sensitive
Treat empty fields as zeroe values
OK
Cancel
Help

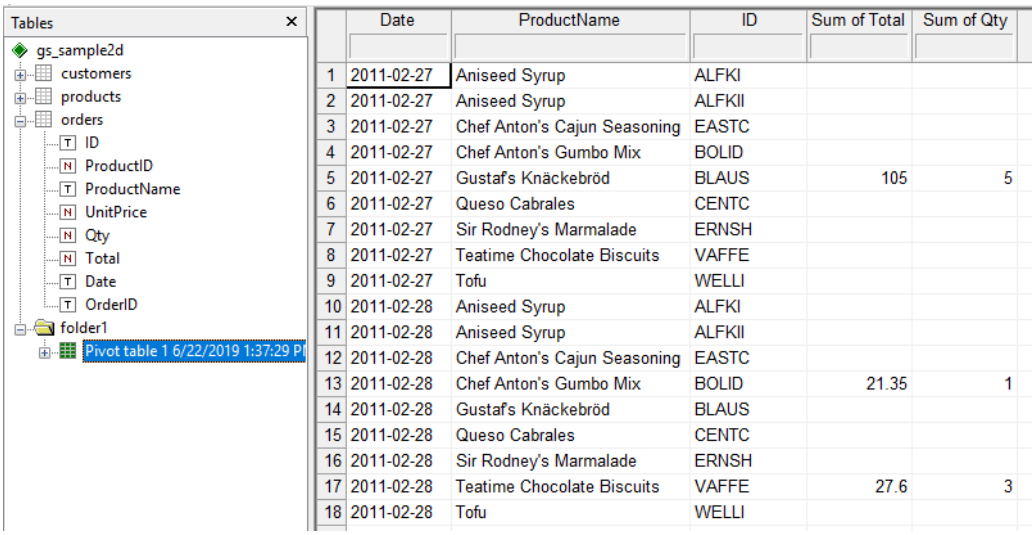


GS-Base Help > Creating pivot table reports

Clicking the **Save** icon in the Pivot Table View enables you to save a new time-stamped report for a given pivot table.

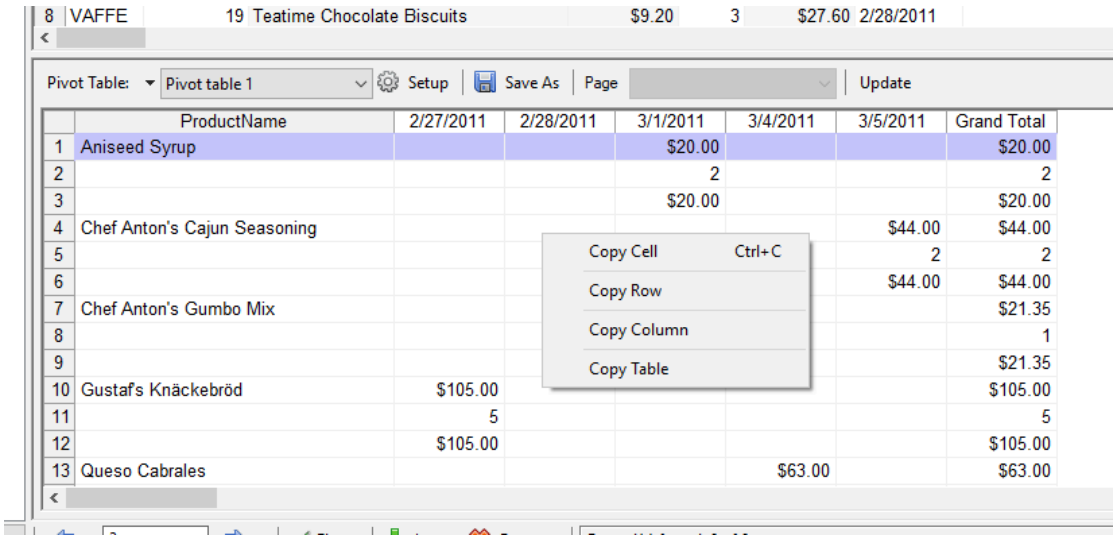
The procedure looks as follows:

1. 

2. 

	Date	ProductName	ID	Sum of Total	Sum of Qty
1	2011-02-27	Aniseed Syrup	ALFKI		
2	2011-02-27	Aniseed Syrup	ALFKII		
3	2011-02-27	Chef Anton's Cajun Seasoning	EASTC		
4	2011-02-27	Chef Anton's Gumbo Mix	BOLID		
5	2011-02-27	Gustaf's Knäckebröd	BLAUS	105	5
6	2011-02-27	Queso Cabrales	CENTC		
7	2011-02-27	Sir Rodney's Marmalade	ERNSH		
8	2011-02-27	Teatime Chocolate Biscuits	VAFFE		
9	2011-02-27	Tofu	WELLI		
10	2011-02-28	Aniseed Syrup	ALFKI		
11	2011-02-28	Aniseed Syrup	ALFKII		
12	2011-02-28	Chef Anton's Cajun Seasoning	EASTC		
13	2011-02-28	Chef Anton's Gumbo Mix	BOLID	21.35	1
14	2011-02-28	Gustaf's Knäckebröd	BLAUS		
15	2011-02-28	Queso Cabrales	CENTC		
16	2011-02-28	Sir Rodney's Marmalade	ERNSH		
17	2011-02-28	Teatime Chocolate Biscuits	VAFFE	27.6	3
18	2011-02-28	Tofu	WELLI		

You can also simply copy the pivot table data to other programs, either as an image or using a few **Copy** commands from the context menu:



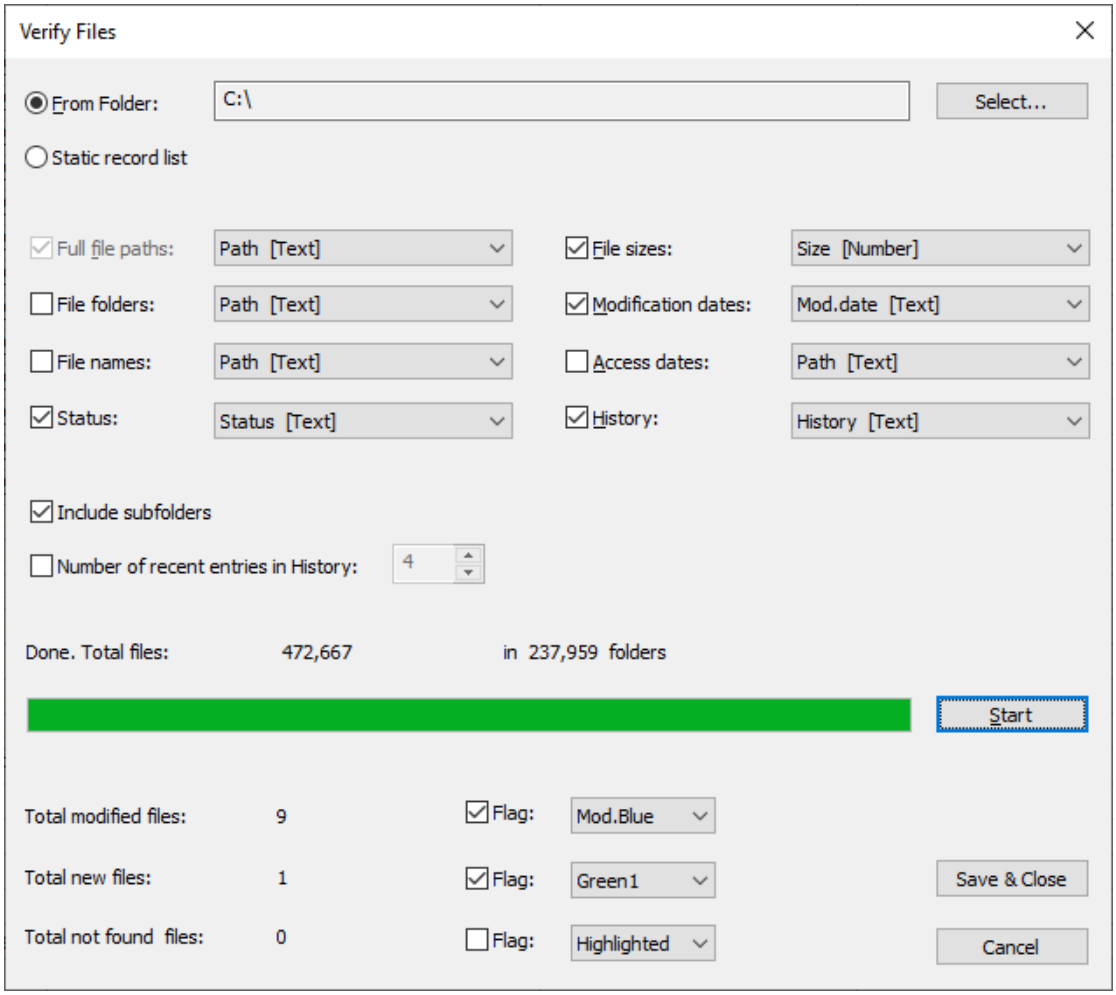
	ProductName	2/27/2011	2/28/2011	3/1/2011	3/4/2011	3/5/2011	Grand Total
1	Aniseed Syrup			$20.00			$20.00
2				2			2
3				$20.00			$20.00
4	Chef Anton's Cajun Seasoning					$44.00	$44.00
5						2	2
6						$44.00	$44.00
7	Chef Anton's Gumbo Mix						$21.35
8							1
9							$21.35
10	Gustaf's Knäckebröd	$105.00					$105.00
11		5					5
12		$105.00					$105.00
13	Queso Cabrales				$63.00		$63.00

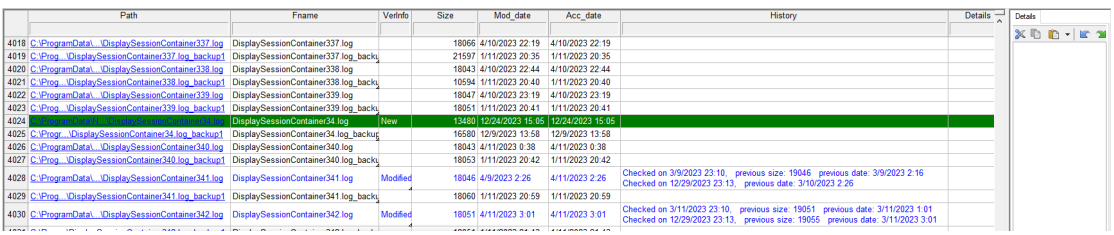
GS-Base Help > Verifying file (path) lists in tables

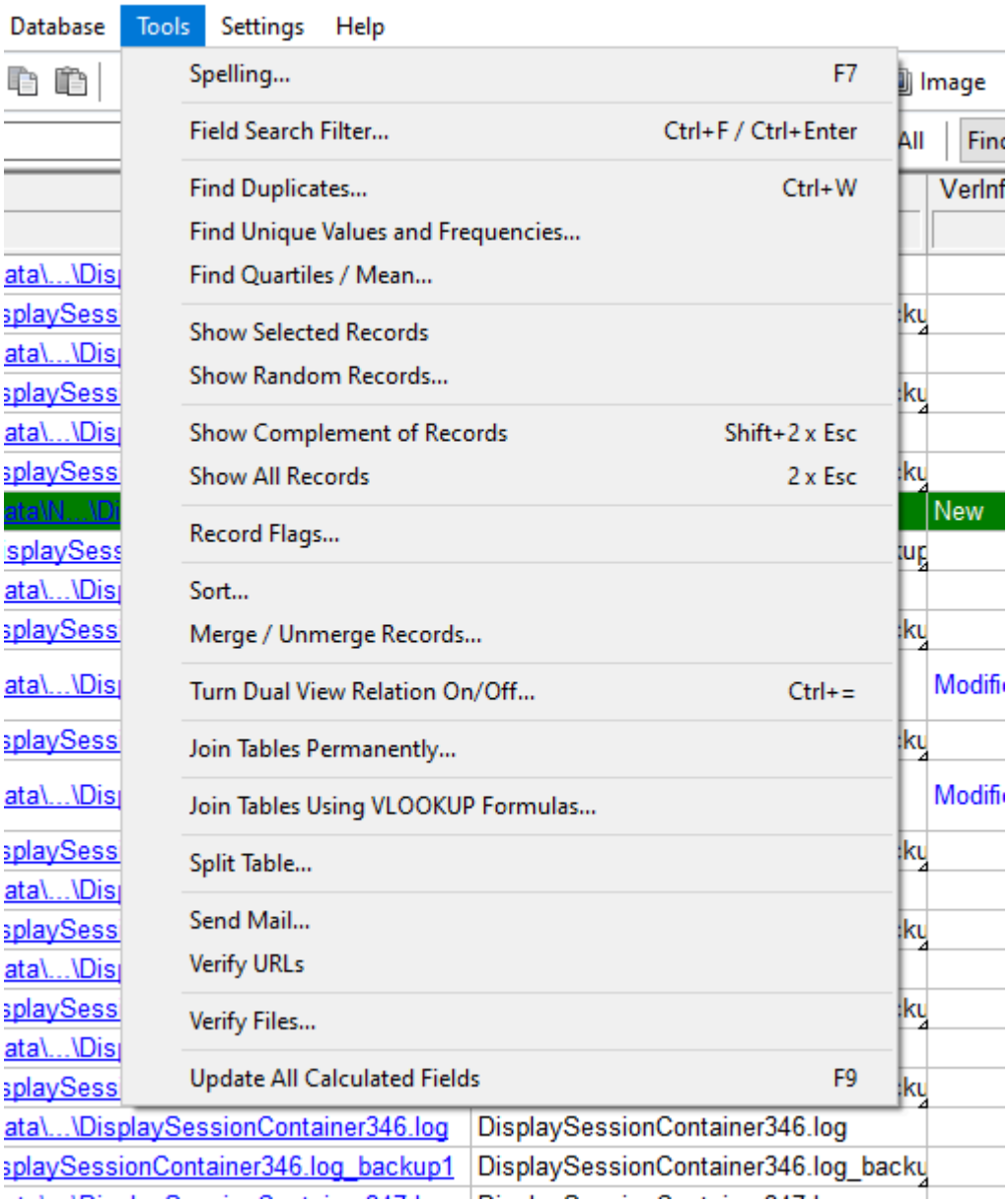
With GS-Base you can create tables/databases with all file paths and the related file details from folders and entire disks, with up to 256 million files in a single table. Using this functionality, you can, for example:

- scan, verify and monitor file changes on your disks, find changed files, new files and deleted files,
- add and maintain any type of file metadata for each file,

- view and search the automatically created history of file changes for each file,
- search for the same or similar files on your disks using regular expressions, search for duplicates, fuzzy searches,
- filter full paths, folders, file names, extensions, modification dates, access dates, file sizes, also using historical data,
- analyze large sets of files (sizes, duplicates, substrings in names/paths, dates) using, e.g., pivot tables,
- perform all of the above (and more) only for specific static lists of files you choose.







To verify/scan files in folders or disks:

1. Enter the location(s) (folder(s)/disk(s)) to scan. Multiple locations should be separated by a semicolon (;), for example:

   ```
   c:\folder1;d:\;e:\folder2
   ```

2. Create a table with fields that will contain the file parameters you want to save and any other fields you want to maintain for each file. The **File path** field must always be selected. When you open the **Verify Files** dialog for the first time, the fields can be preselected if their names contain the following substrings:

"path", "folder", "file", "ver", "size", "mod", "access", "history".
You can also later freely add or remove any fields, including calculated fields.

3. Click **Start** to scan a selected folder or disk. If you choose, e.g., an entire system partition with a large number of files and folders on a slower HDD disk, the process can take anything from seconds to minutes. As Windows caches already-accessed file information for some short period of time, if you start scanning the same location again, it'll be much faster.

4. Click **Save & Close** to fill the table with the newly generated results, replacing the previous ones. If you click **Cancel**, the table won't be updated.
Note: old entries are safe; none of the existing entries in the database table are removed until you manually delete such a record in GS-Base.
New files are added automatically.

5. To mark files as modified, added or deleted since the last verification, use either flags (font and background colors) that you can define using the **Tools > Record Flags** command, or use the **Verification** field that will contain one of the state strings "Modified", "New", "Not Found", or will be empty. Each new verification/scanning overwrites the previous flags and states.

6. To save subsequent size or modification date changes for files, select the **History** field. Each such change detected during verification will be added as a new line in that field (and the row heights will be adjusted automatically) in the following form:

   ```
   On [date], size: ... mod: [date]
   ```

   Older entries are moved to the lines below within a table cell. If you want to keep them in one line, click the column, then in the **Find** edit field on the **Find** toolbar enter (using the default regex find/replace mode):
   `\R`
   and in the **Replace** field enter any character/string you would like to use to separate entries. Alternatively, if these **Text** fields become large, you can also convert them to the **Long Text** type, with data displayed in separate panes.

7. Use the standard GS-Base functions to search for field values, filter flagged records, etc. If you would like to verify files in multiple locations/disks, for example, to find file duplicates, create a few tables and use the **Merge Records** commands.

8. You can use the **Format > Hyperlink** command to format the **File path** and **Folder** fields. Clicking or pressing Enter on a file name opens it in the associated program. Clicked folders are displayed for browsing.

9. By default, dates are saved in the standard YYYY-MM-DDTHH:MM:SS format. To display them in one of the many available date/time formats, use the **Format > Style** command.

10. **Note:** only folders and disks that are accessible (e.g. can be browsed in File Manager) from your Windows account can be scanned.

11. In calculated fields you can use the

    ```
    =fileStats(file_path, tag)
    ```

    formula/function to extract tens of various EXIF tags from your photos and images from entire disks and filter them, e.g., by the date a given photo was taken, its author, the place (via GPS), the camera settings, the software used to process it, etc.
    It's a counterpart to the objectStats() function that does the same for objects/files inserted in Images/Files binary fields. Both are listed in the **informational** formula category.
    Using objectStats() Functions to Obtain File/Image Data

GS-Base Help > *Mass-renaming, copying and deleting files*

Along with the "Verify Files" command, GS-Base introduces a few additional "file management" commands to mass-rename, copy/move, and delete files on your disks.

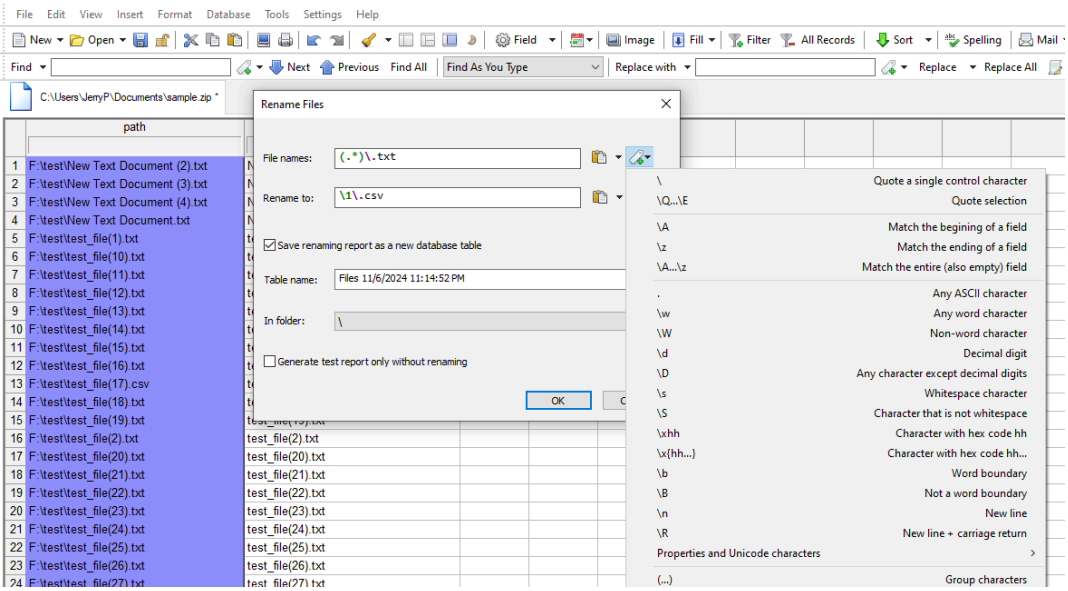
Renaming Files

To mass-rename files on your disks, create a table with file paths from given folders or disks.
You can load it using, for example, the **Tools > Verify Files** command. Then select the column with file paths or some subset/range of paths from that column

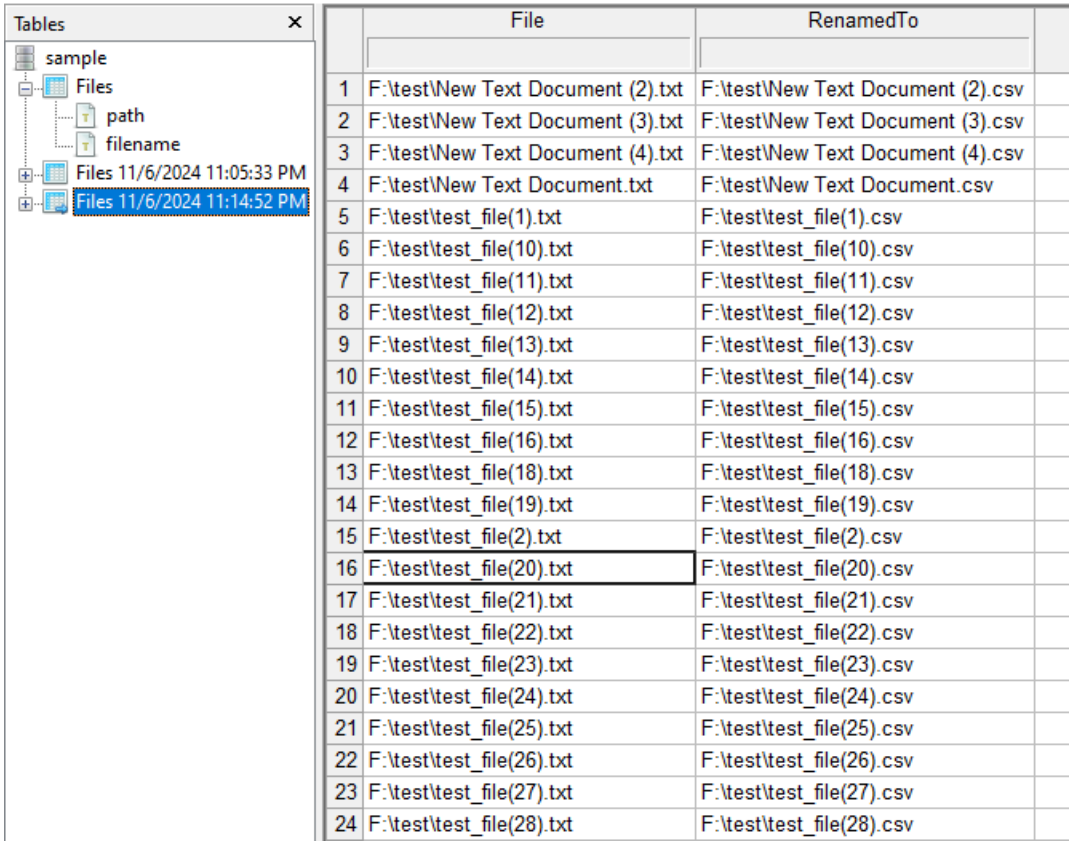
and use the **Tools > Manage Files > Rename** command.
You can filter files to be renamed any way you want: substrings in their file names/paths using regular expressions, the standard file metadata, your own added custom attributes, or EXIF (photos/images/mp4) tags.

Renaming uses a regular expression to specify which patterns to look for and how to modify the found text. Renaming concerns strictly the file name with its extension only, excluding the rest of the path.

For example, to change the file extensions for all files from "txt" to "csv" on a disk or in some folders, you can use the following expressions:



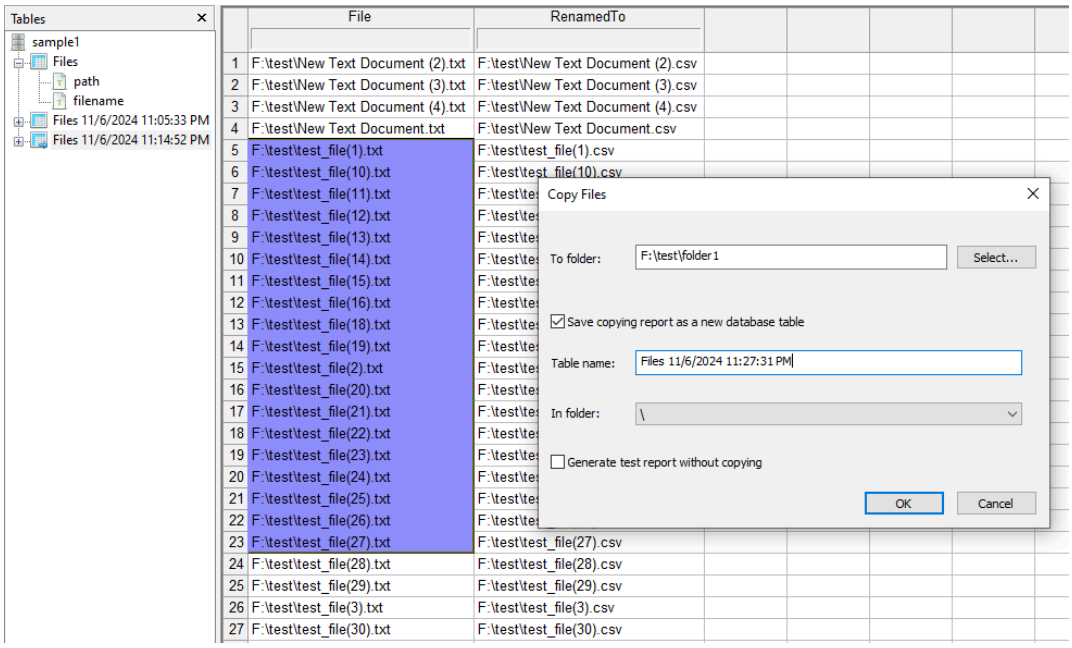
All performed changes can be saved in time-stamped tables, for example:

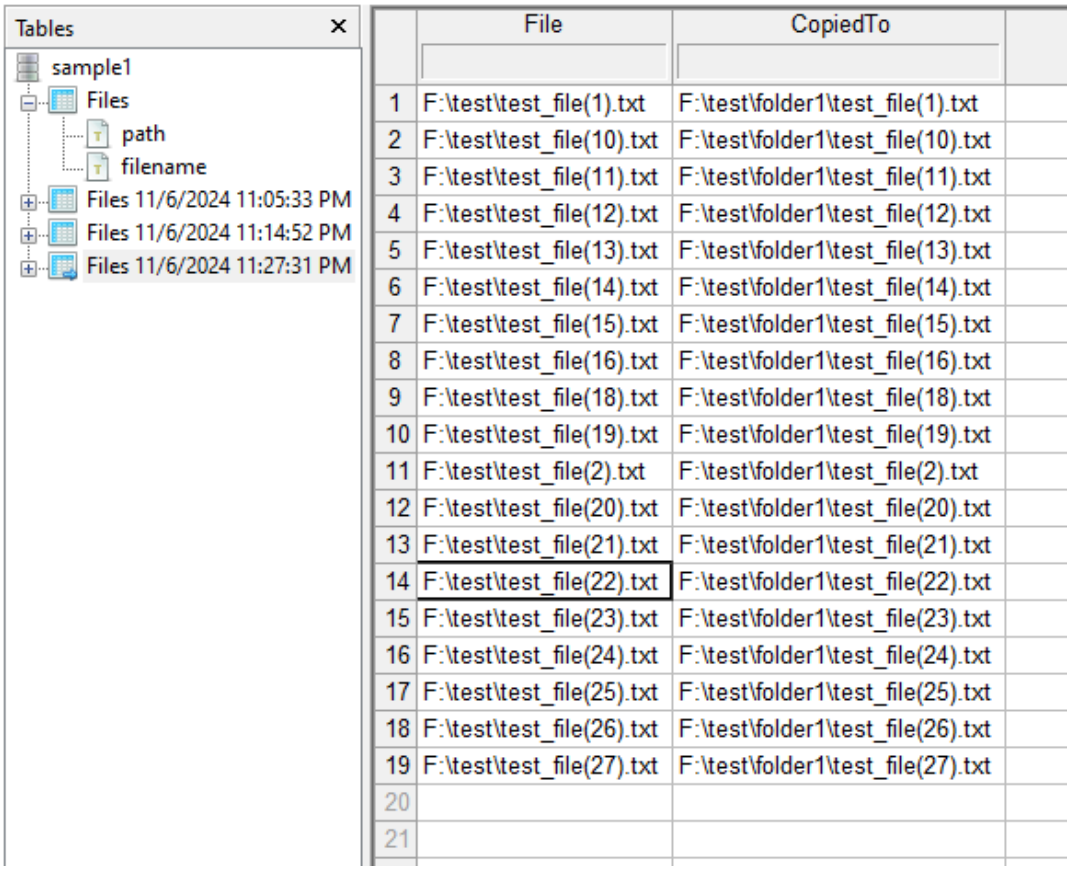


	File	RenamedTo
1	F:\test\New Text Document (2).txt	F:\test\New Text Document (2).csv
2	F:\test\New Text Document (3).txt	F:\test\New Text Document (3).csv
3	F:\test\New Text Document (4).txt	F:\test\New Text Document (4).csv
4	F:\test\New Text Document.txt	F:\test\New Text Document.csv
5	F:\test\test_file(1).txt	F:\test\test_file(1).csv
6	F:\test\test_file(10).txt	F:\test\test_file(10).csv
7	F:\test\test_file(11).txt	F:\test\test_file(11).csv
8	F:\test\test_file(12).txt	F:\test\test_file(12).csv
9	F:\test\test_file(13).txt	F:\test\test_file(13).csv
10	F:\test\test_file(14).txt	F:\test\test_file(14).csv
11	F:\test\test_file(15).txt	F:\test\test_file(15).csv
12	F:\test\test_file(16).txt	F:\test\test_file(16).csv
13	F:\test\test_file(18).txt	F:\test\test_file(18).csv
14	F:\test\test_file(19).txt	F:\test\test_file(19).csv
15	F:\test\test_file(2).txt	F:\test\test_file(2).csv
16	F:\test\test_file(20).txt	F:\test\test_file(20).csv
17	F:\test\test_file(21).txt	F:\test\test_file(21).csv
18	F:\test\test_file(22).txt	F:\test\test_file(22).csv
19	F:\test\test_file(23).txt	F:\test\test_file(23).csv
20	F:\test\test_file(24).txt	F:\test\test_file(24).csv
21	F:\test\test_file(25).txt	F:\test\test_file(25).csv
22	F:\test\test_file(26).txt	F:\test\test_file(26).csv
23	F:\test\test_file(27).txt	F:\test\test_file(27).csv
24	F:\test\test_file(28).txt	F:\test\test_file(28).csv

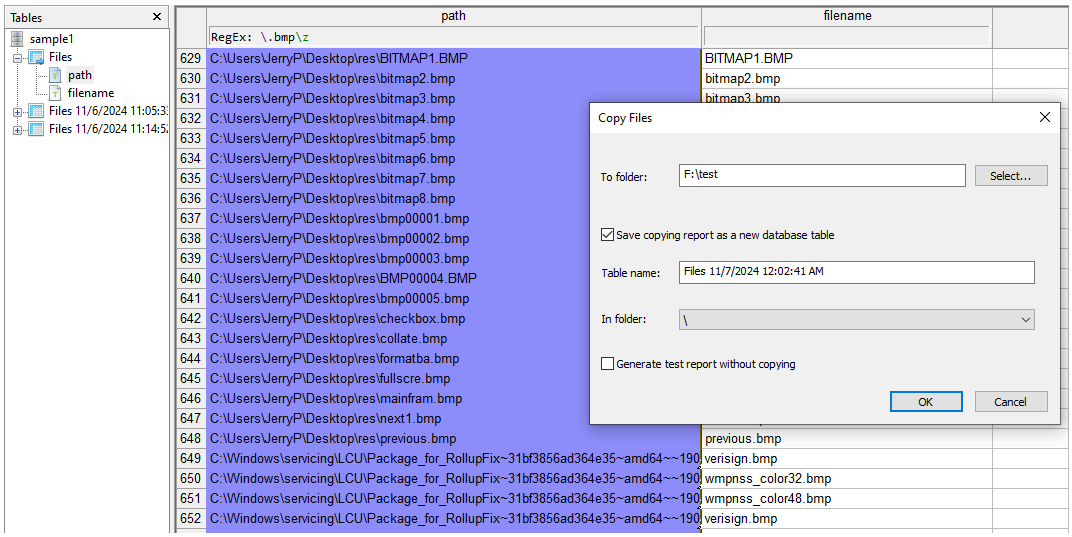
Copying Files

Similarly, use the **Tools > Manage Files > Copy** command to mass-copy any number of filtered files from your disk to a given single location and save the report. Repeated file names are automatically modified by adding the "(n)" suffix, like fname.png → fname(1).png, fname(2).png, etc.



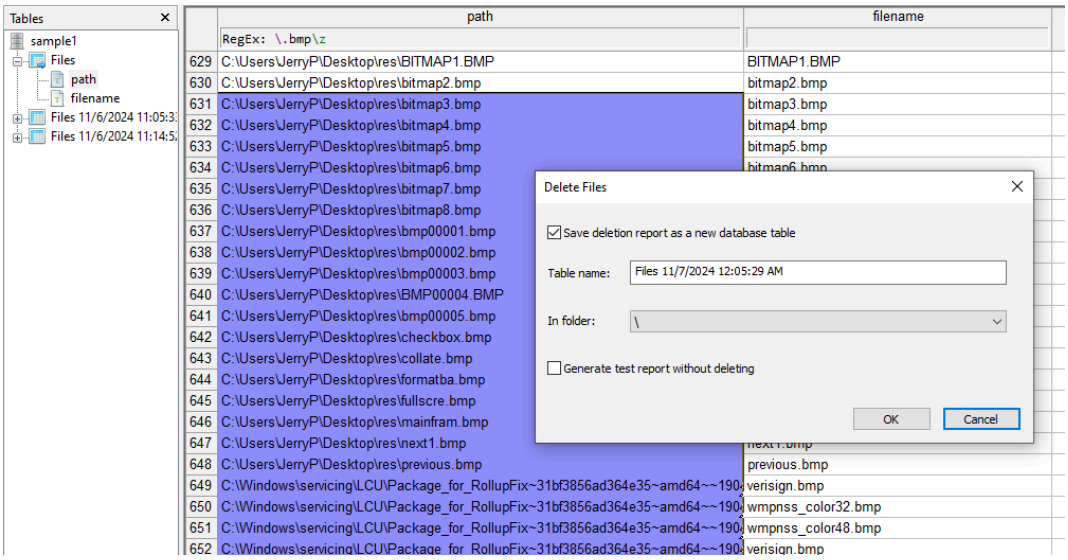


You can copy groups of files from any locations, applying any filters:



Deleting Files

The "Delete Files" command can be useful, e.g., to remove original files after making their copies in a given folder, or to remove duplicated files, etc. Reports about deleted files are created optionally, like for the previous two commands.

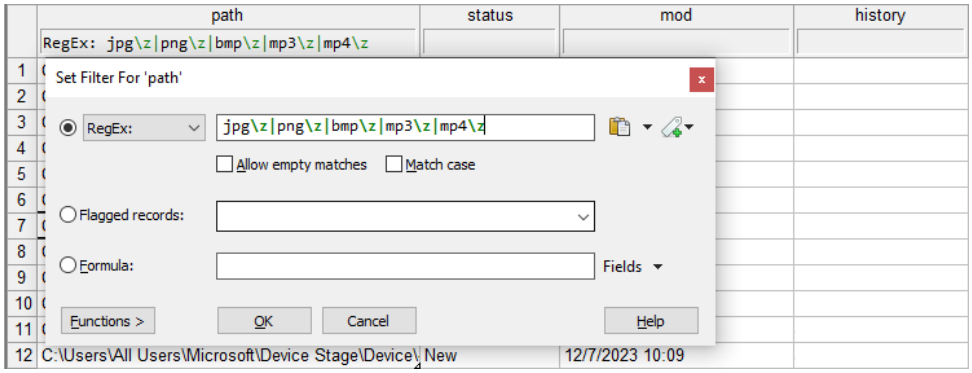


You can also check the "Generate test report" check box to verify/browse the report data first, without performing any actual renaming, copying, or deleting of files.
Note: test reports don't include/generate the "(n)" suffixes mentioned above for the **Tools > Manage Files > Copy** command.

1. Use the Verify File command and choose a folder or disk to scan and load the file listing to selected table fields.

2. As the above command by default loads all file paths, for clarity you can filter out from the displayed listing any paths for file types that you don't need:

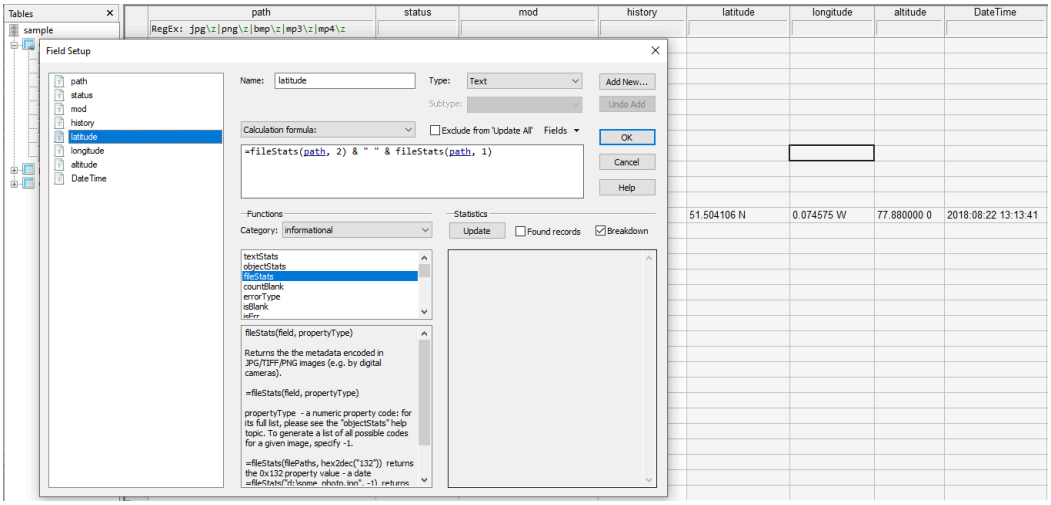
   a. Scroll to the path field, press Ctrl+Enter to enter a filter for file extensions, for example, enter:
   jpg**\z**
   mp3**\z**
   mp4**\z**
   You can combine multiple extension patterns with "or", for example:
   jpg**\z|**mp3**\z|**mp4**\z**



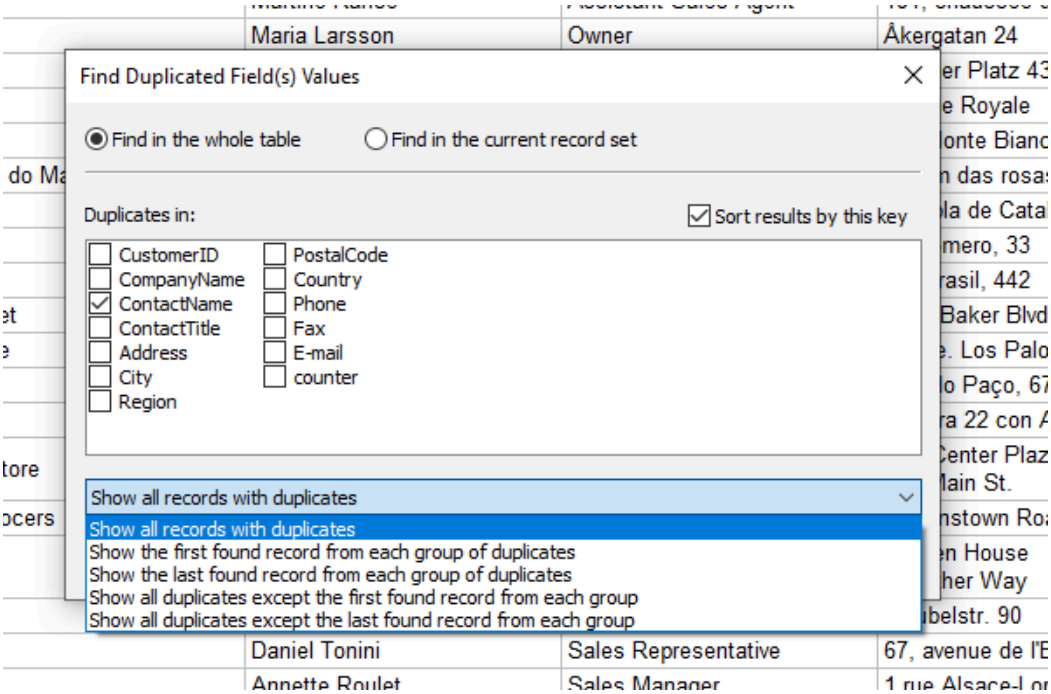
   b. Additionally, to permanently remove the unwanted file paths from the table:
   (a) use the **Tools > Show Complement** command to display the filtered-out ones first,
   (b) click any column to select all these records (or enter something like 1:10000 in the record number field at the bottom) and press Ctrl+D to delete them. Double-press Esc (the **Tools > Show All** command) to go back to the found recordset.

3. Choose which EXIF tags or multimedia tags you want to compare to find duplicates (or, more generally, some specific sets of files), such as the GPS

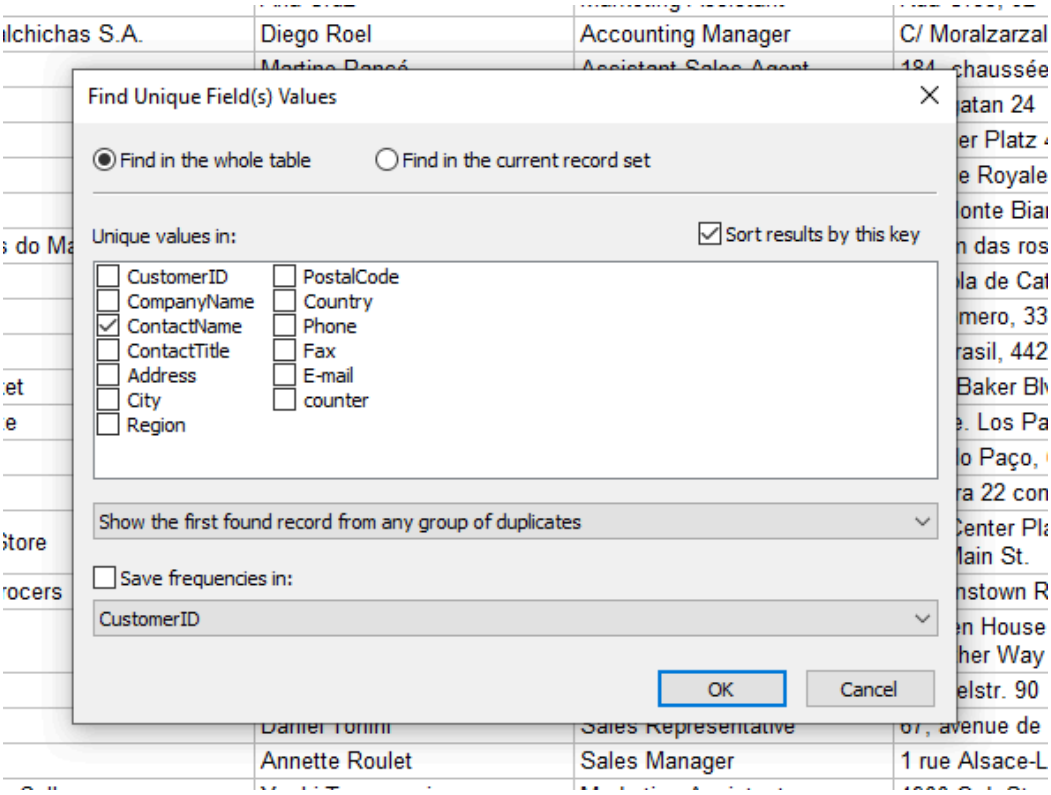
location where the photo was taken, author or artist, dates, camera type, track name or length, file size, etc. There are about 60 EXIF tags and 70 multimedia sound/video tags; please see: Using fileStats() function.
For each one you choose to check, add a calculated text field with this formula and the tag code:



4. The specified values will be automatically extracted/calculated. You can search/sort them any way. You can use various methods to find duplicates. The direct one is to use the Tools > Find Duplicates command.



Alternatively, you can use the Tools > Find Unique command, then the **Tools > Show Complement** command and delete all displayed records, which will result in leaving one occurrence of each path.

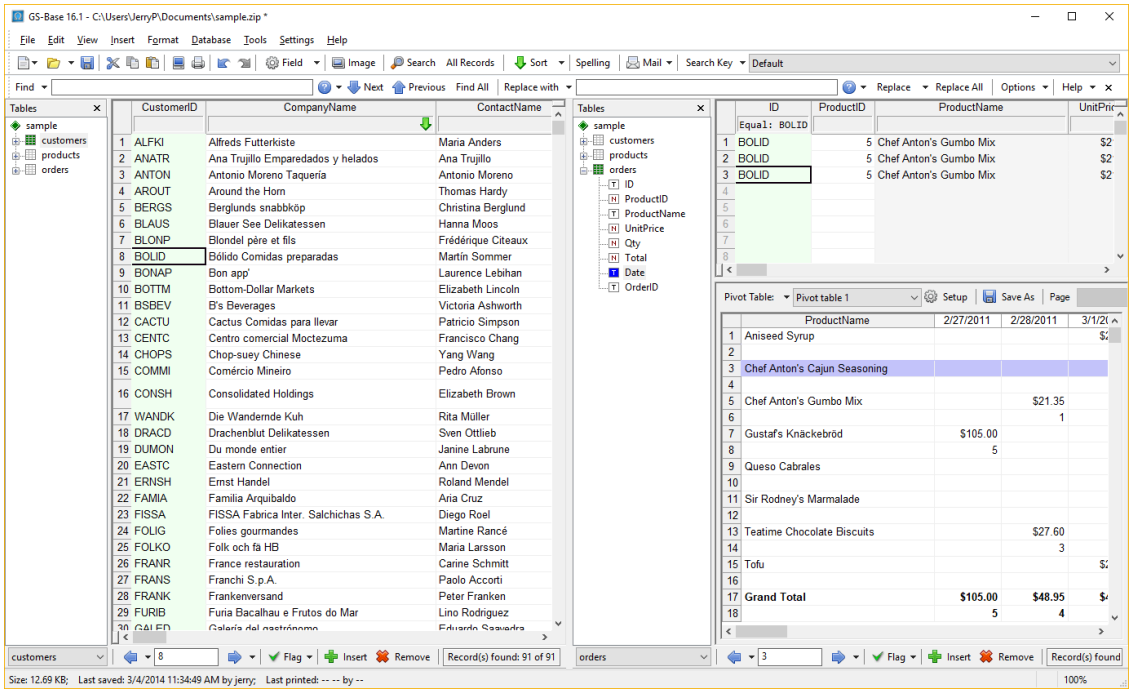


5. You can perform various other searches, such as for photos taken between certain dates, sound files with specific bitrates, videos using specific encoding, songs or videos by a given artist and many more; see: Using fileStats() function.

6. To delete the files you found, use any method involving the "del" command in the command line. For example, insert "del" before each path (that is, define a field operation as `"del" & " " & path` and complete it with one click) and save or copy this column (or two columns) to a *.bat file to execute in CMD. You can also use, for example, "copy" to move found files to a different location.

7. In one scan you can create a list of 256 million file paths - at the moment this is the maximum number of records per table. If you need to compare files between multiple disks/folders, repeat this procedure and use one of the table merging options (or, if you don't use binary fields in your tables, simply select the entire 2nd table, copy it to the clipboard, place the cursor below the 1st field of the last record in the 1st table and paste the data).

Use the **View > Dual View** command to split the main GS-Base window into two panes that display two database views independently. This enables you to browse and edit two different tables at the same time.



GS-Base Help > *Linking tables in dual views by the 1-N relation*

To Create a Link (the One-to-Many Relation) Between Two Fields in Different Tables

1. Use the **View > Dual View** command to open the database in two panes: the left and right panes.
2. With the current table as the "master" table in the left pane, select a desired "slave/search" table in the right pane, then place the current selections in fields (columns) that are to be linked.
3. Use the **Tools > Set Relation (Ctrl+=)** command to turn the relation on and off.

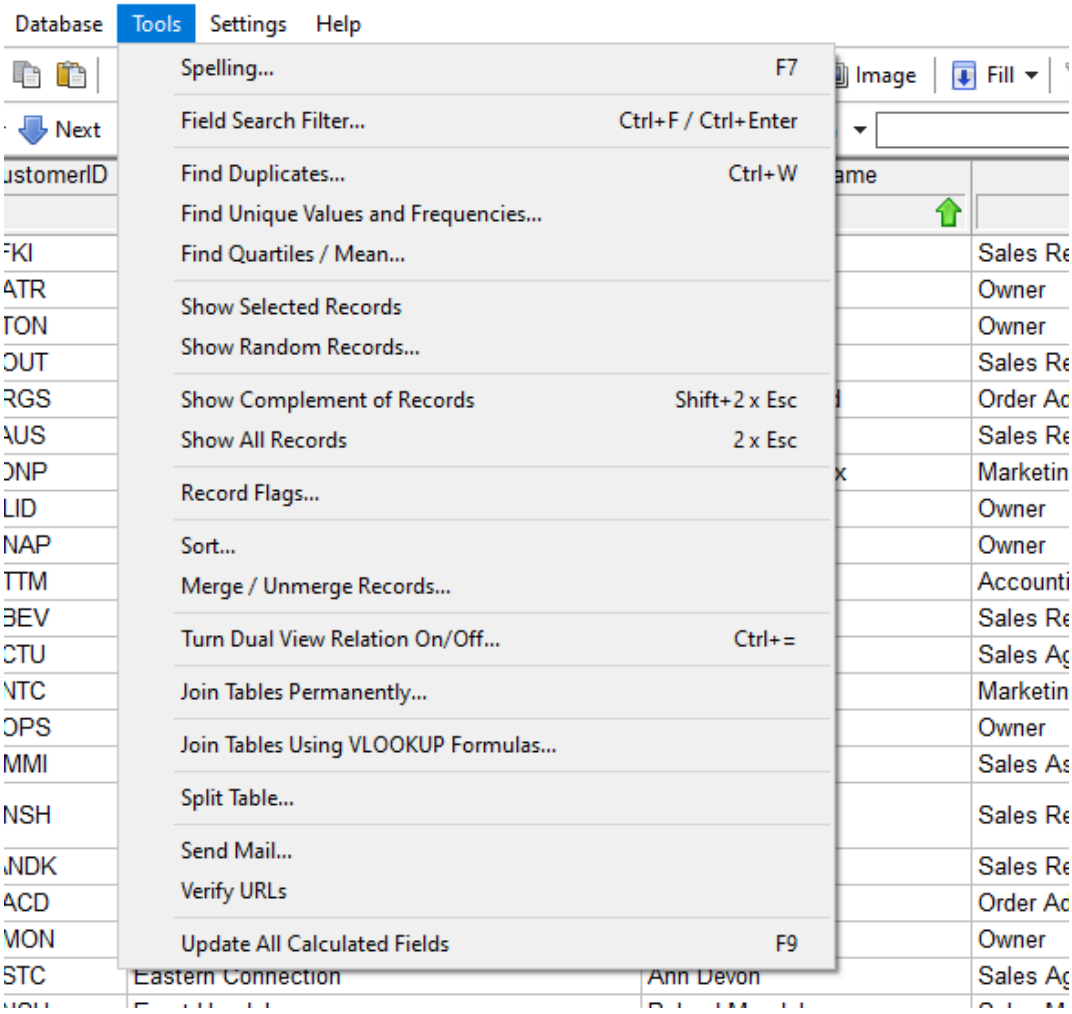
The linked fields are displayed with a highlighted background color. Scrolling to another record in the "master" table in the left pane causes automatic filtering of the "search" table to find matching values in the linked field in the right pane.

Closing the dual view turns off the active relation.

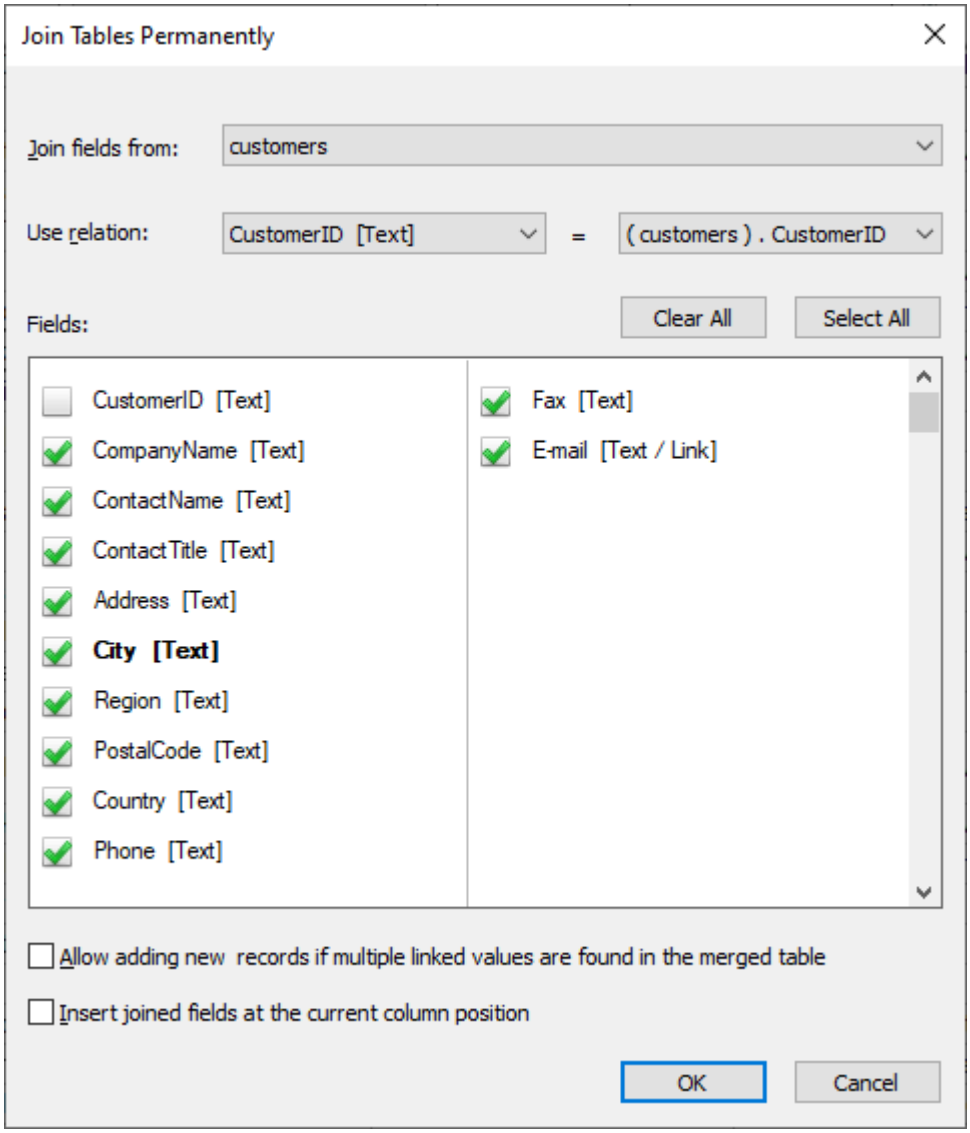
Executing the **Tools > Set Relation (Ctrl+=)** command again turns off the active relation, no matter which tables are currently selected in the dual view.

Joining Tables

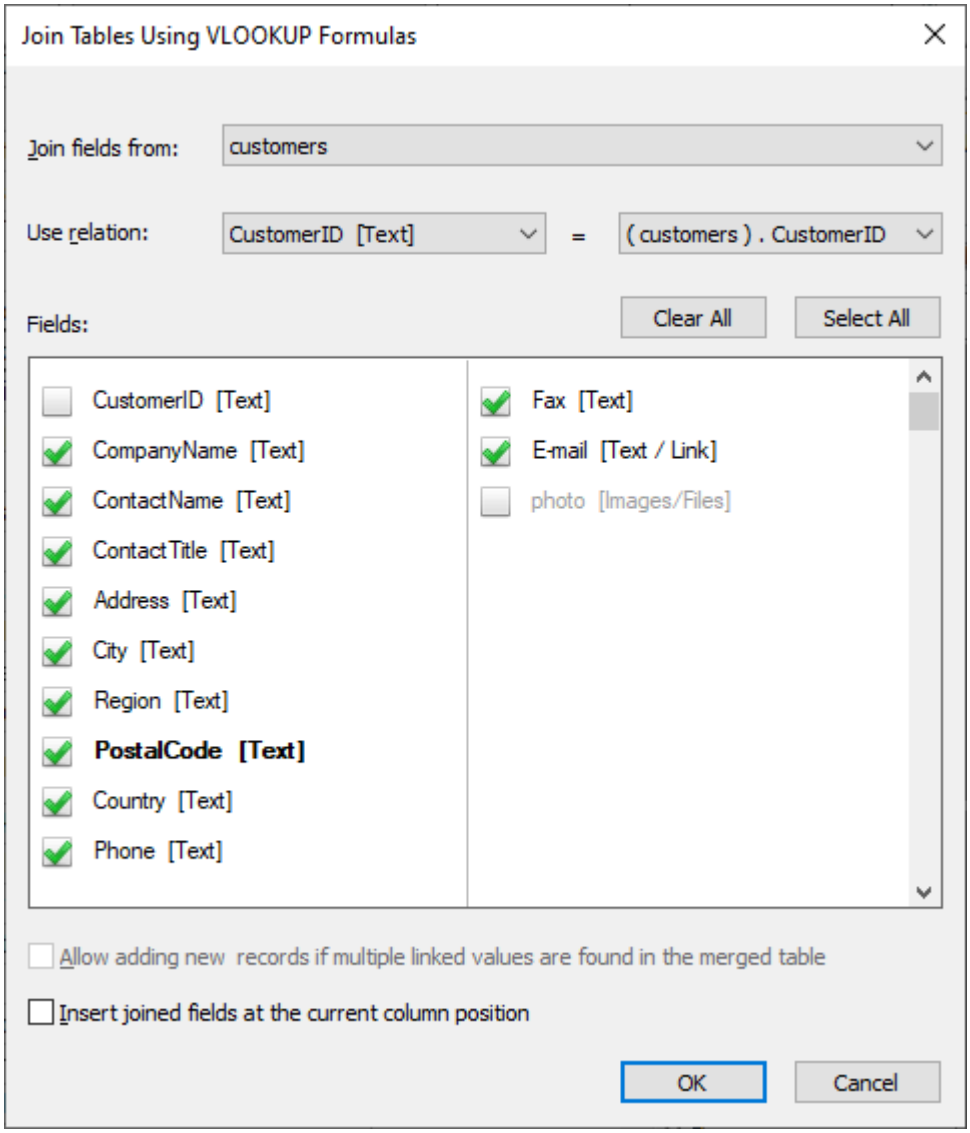
To perform table joins and the opposite operations (table splitting / normalizing), use the **Tools > Join (...)** commands.



You can create joins either permanently or dynamically (based on the fast binary vlookup calculation fields) with single clicks, without entering any formulas.
In the "Join Tables Permanently" dialog box you can now choose which fields should be merged and whether they should be inserted at the current column/field position:

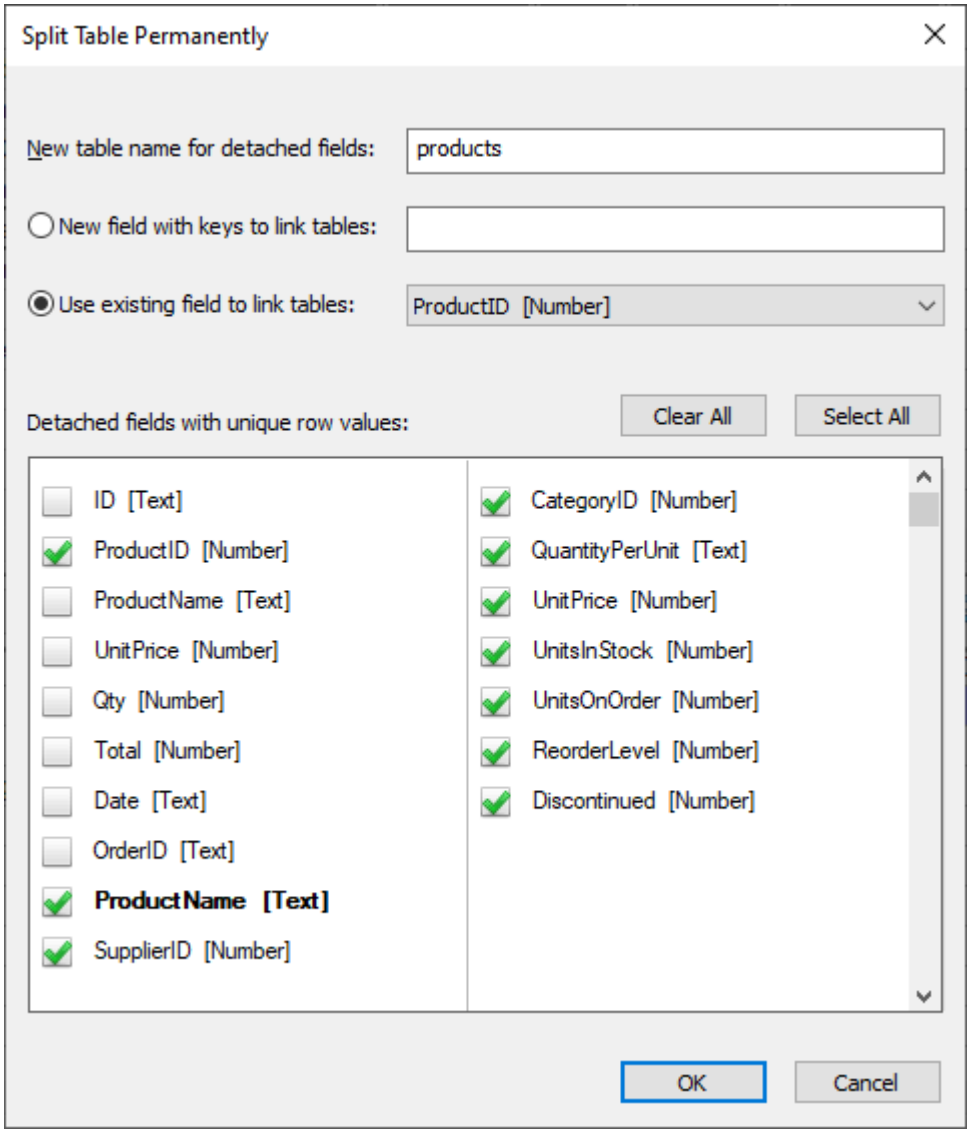


In the "Join Tables Using VLOOKUP Formulas" dialog box you can choose which calculation fields should be added to the current table to perform lookups in the dynamically "joined" table(s). You no longer have to manually enter long formulas in these calculation fields. No matter how many lookup fields are added or how many tables are joined, everything is done automatically.



Splitting (Normalizing) Tables

You can automatically split a table where the same group of fields occurs in multiple records. In general, this is the opposite of the above "join" actions. For example, if the sample "orders" table includes many orders of the same group of products and each "order" record includes the full specification of a given product, potentially duplicating these specifications, that "orders" table can be split to include only, e.g., the order date and amount fields, and the list of unique product specifications will be kept in the newly created "products" table.



This may often result in much smaller database files and memory usage, and make editing product descriptions easier. For any reporting purposes, with a few clicks you can add selected product description fields back either permanently or as calculation fields with fast vlookups.

For example, to transform a single large table containing both the order and repeated product details, you just need to specify in the above dialog box a new table name (e.g. "Products"), a link field name (e.g. "ProductID"), deselect the order fields, and GS-Base will automatically modify the original table and add a new "Products" table:

	Qty	Total	Date	OrderID	ProductName	UnitPrice	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
1	2	$20.00	3/1/2023	NIEoDmOJ	Aniseed Syrup	$10.00	12 - 550 ml bottles	$10.00	13	70	25	0
2	1	$21.35	3/2/2023	Z460cNyG	Chef Anton's Gumbo Mix	$21.35	36 boxes	$21.35	0	0	0	1
3	3	$63.00	3/5/2023	ywElFzWA	Queso Cabrales	$21.00	1 kg pkg.	$21.00	22	30	30	0
4	2	$44.00	3/7/2023	HFPfuyAZ	Chef Anton's Cajun Seasoning	$22.00	48 - 6 oz jars	$22.00	53	0	0	0
5	5	$105.00	3/11/2023	5t413Mty	Gustaf's Knäckebröd	$21.00	24 - 500 g pkgs.	$21.00	104	0	25	0
6	1	$81.00	3/14/2023	0FXFBthT	Sir Rodney's Marmalade	$81.00	30 gift boxes	$81.00	40	0	0	0
7	1	$23.25	3/14/2023	4sYrAcsz	Tofu	$23.25	40 - 100 g pkgs.	$23.25	35	0	0	0
8	3	$27.60	3/14/2023	nPGuPhsT	Teatime Chocolate Biscuits	$9.20	10 boxes x 12 pieces	$9.20	25	0	5	0
9	3	$64.05	3/18/2023	ZcdswJTf	Chef Anton's Gumbo Mix	$21.35	36 boxes	$21.35	0	0	0	1
10	2	$42.00	3/19/2023	zdX2exNg	Queso Cabrales	$21.00	1 kg pkg.	$21.00	22	30	30	0
11	1	$22.00	3/19/2023	CWBwTUBG	Chef Anton's Cajun Seasoning	$22.00	48 - 6 oz jars	$22.00	53	0	0	0
12	4	$85.40	3/19/2023	ehpgMZ54	Chef Anton's Gumbo Mix	$21.35	36 boxes	$21.35	0	0	0	1
13	1	$21.00	3/20/2023	uHxdsEx5	Queso Cabrales	$21.00	1 kg pkg.	$21.00	22	30	30	0
14	3	$66.00	3/20/2023	Ekl7Ab7i	Chef Anton's Cajun Seasoning	$22.00	48 - 6 oz jars	$22.00	53	0	0	0

Orders & Products

Orders

	Qty	Total	Date	OrderID	ProductID
1	2	$20.00	3/1/2023	NIEoDmOJ	1
2	1	$21.35	3/2/2023	Z460cNyG	3
3	3	$63.00	3/5/2023	ywElFzWA	5
4	2	$44.00	3/7/2023	HFPfuyAZ	2
5	5	$105.00	3/11/2023	5t413Mty	4
6	1	$81.00	3/14/2023	0FXFBthT	6
7	1	$23.25	3/14/2023	4sYrAcsz	8
8	3	$27.60	3/14/2023	nPGuPhsT	7
9	3	$64.05	3/18/2023	ZcdswJTf	3
10	2	$42.00	3/19/2023	zdX2exNg	5
11	1	$22.00	3/19/2023	CWBwTUBG	2
12	4	$85.40	3/19/2023	ehpgMZ54	3
13	1	$21.00	3/20/2023	uHxdsEx5	5
14	3	$66.00	3/20/2023	Ekl7Ab7i	2

+

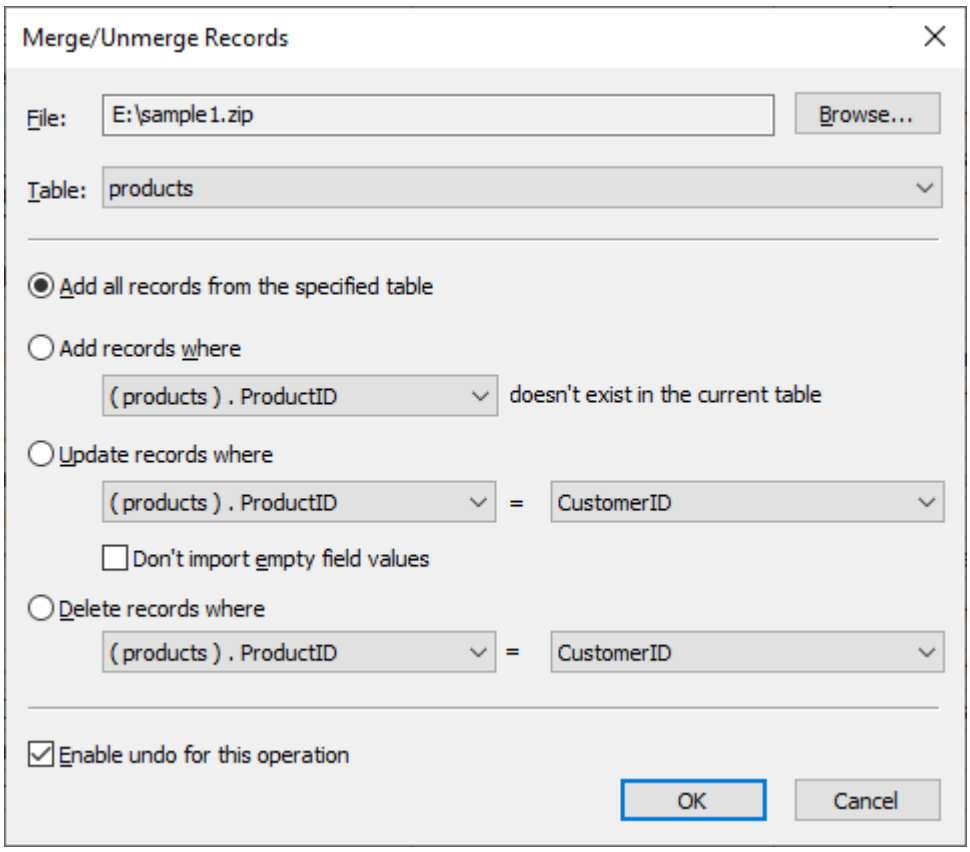
Products

	ProductID	ProductName	UnitPrice	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
1	1	Aniseed Syrup	$10.00	12 - 550 ml bottles	$10.00	13	70	25	0
2	3	Chef Anton's Gumbo Mix	$21.35	36 boxes	$21.35	0	0	0	1
3	4	Gustaf's Knäckebröd	$21.00	24 - 500 g pkgs.	$21.00	104	0	25	0
4	6	Sir Rodney's Marmalade	$81.00	30 gift boxes	$81.00	40	0	0	0
5	8	Tofu	$23.25	40 - 100 g pkgs.	$23.25	35	0	0	0
6	7	Teatime Chocolate Biscuits	$9.20	10 boxes x 12 pieces	$9.20	25	0	5	0
7	5	Queso Cabrales	$21.00	1 kg pkg.	$21.00	22	30	30	0
8	2	Chef Anton's Cajun Seasoning	$22.00	48 - 6 oz jars	$22.00	53	0	0	0

GS-Base Help > *Merging and unmerging records*

Merging/Unmerging Records

To merge (add) or "unmerge" (mass-delete) records from other databases, use the **Tools > Merge/Unmerge** command. The displayed dialog box enables you to specify how adding or removing records should be carried out:



Option 1, "Add all records from the specified table", unconditionally adds all records from the external table and performs field name matching in the current and the merged tables to assign imported values to the correct fields. You can bypass this matching requirement by using this command in the form of a script - please see the examples in the Scripting section of this documentation.

Options 2 to 4 cause adding records, updating records, or deleting records in the current table where the specified field values match in both tables.

Note: to perform mass-merging of tables from a given folder or multiple folders, use a script as shown in the "Merging" scripting examples further in this manual.

Spell-Checking Availability

GS-Base 64-bit can use either the system spell checking functionality provided by Windows 8.x and later, or the Hunspell spell checking engine (available for all systems).
GS-Base 32-bit can use Hunspell spell checking only.
The current selection can be changed either in the **Settings > Options > Spelling** dialog box or directly in the **Tools > Spelling** dialog box.

Adding Hunspell Dictionaries

To install a Hunspell dictionary for a given language:

1. Go to one of the websites which distribute Hunspell dictionaries, for example:

   https://addons.mozilla.org/en-US/firefox/language-tools/

   The dictionaries are distributed as zip files, typically with *.xpi or *.oxt extensions. After downloading the desired language file, rename it to a *.zip file.

2. Each downloaded dictionary archive contains two text files with the *.aff and *.dic extensions and a name that (usually) represents the corresponding language ISO code, e.g. en-US.aff and en-US.dic (English US).

   Copy those files to some folder and specify that folder in the GS-Base spelling options. GS-Base will automatically find all installed Hunspell dictionaries and list them in the **Spelling** dialog box the next time you execute the **Tools > Spelling** command.
   To remove a given Hunspell dictionary, delete its *.aff and *.dic files from the above folder.

Adding Custom Words to Dictionaries

Users can add their own words to existing dictionaries (or, when using the Hunspell spell checking engine, create their own full dictionaries).
The added words are saved in a text file in a folder specified in the GS-Base spelling options. The file extension is *.txt, and the file name is the language ISO code corresponding to the dictionary the file is linked to. For example, if the dictionary files are

`en-US.aff` and `en-US.dic`

then the custom words will automatically (after closing the dialog box) be saved to

`en-US.txt`

New words are added by clicking the **Add Word** button when performing spell checking. The *.txt files can also be edited in any text editor to add a larger number of words at once.
The text files containing added words must be saved using UTF-8 encoding.

Other ISO codes include:

af-ZA	cs-CZ	gd-IE	pt-PT
sq-AL	da-DK	de-DE	pt-BR
ar-AE	nl-NL	de-AT	ro-RO
ar-BH	nl-BE	de-LI	ru-RU
ar-DZ	en-AU	de-LU	sr-SR
ar-EG	en-BZ	de-CH	sl-SI
ar-IQ	en-CA	el-GR	sk-SK
ar-JO	en-CB	he-IL	es-ES
ar-KW	en-IE	hi-IN	es-AR
ar-LB	en-JM	hu-HU	es-BO
ar-LY	en-NZ	is-IS	es-CO
ar-MA	en-PH	id-ID	es-CR
ar-OM	en-ZA	it-IT	es-DO
ar-QA	en-US	it-CH	es-EC
be-BY	en-GB	ja-JP	es-GT
bg-BG	et-EE	ko-KR	es-HN
zh-CN	fi-FI	lv-LV	es-MX
zh-HK	fr-FR	lt-LT	es-NI
zh-MO	fr-BE	mk-MK	es-PA
zh-SG	fr-CA	ms-MY	es-PE
zh-TW	fr-LU	no-NO	es-PR
hr-HR	fr-CH	pl-PL	es-PY

es-UY	sv-FI	uk-UA	yi-IL
es-VE	th-TH	uz-UZ	
sv-SE	tr-TR	vi-VN	

Fonts

The Font dialog box enables you to specify all basic font attributes, including its name, size, weight, slant (the italic style), the **underline** and **strikeout** options and the font color.

To revert to the default settings, click the **Reset** button.

Style

You can use the following standard numeric styles and their options:

- Default
  (No specific formatting. Numbers with more than 15 digits are displayed in the exponential-scientific format.)
- General
  - decimal places (0 to 14)
  - leading zeros (0 to 14)
  - thousand separator
  - displaying negative numbers in red or in parenthesis
- Currency
  - currency symbol position
  - currency symbol
  - displaying negative numbers in red or in parenthesis
- Accounting
  - currency symbol

- Date
  - date pattern
  - using fixed or system dependent year/month/day order
- Time
  - time/period pattern
- Date/Time
  - date pattern
  - time pattern
  - using fixed or system dependent year/month/day order
  - date/time order
- Percent
  - decimal places (0 to 14)
- Fraction
  - either a fixed denominator value (1 to 9,223,372,036,854,775,807) or a fixed number of denominator digits (1 to 19)
- Scientific
  - decimal places (0 to 14)
  - a fixed exponent value

Alternatively, you can use a **user-defined** numeric format. To do this, simply enter the desired style pattern in the Style dialog box. If you need to re-use it, you can save it as a new user style with a new name. To learn more about style patterns, please see how the standard styles are defined. For example, the accounting format is defined as:

```
_(\$* #,##0.00_);"($"* #,##0.00\);_(\$* \-??_);_(@_)
```

and the fraction format with the fixed denominator (4) can be defined as:

```
?\ ?/\4
```

User-defined formats allow you to specify font colors for numbers that meet some conditions. For example:

```
[green]#.#;[red]\-#.#;[blue]#.#;[gray]@
```

This displays positive numbers in a green font, negative numbers in red, zeroes in blue and text labels in gray.

```
[red][<10]#0.00;[yellow][<=50]#0.0;[green][<400]##0;[magenta][>=400]#00
```

This displays numbers less than 10 in a red font, numbers less than or equal to 50 in a yellow font, numbers less than 400 in a green font and numbers greater than or equal to 400 in a magenta font.

Color values can be expressed as RGB values (for example, `[red][<10]#0.00` is the same as `[#FF0000][<10]#0.00`) or by one of the following color names:

black	[#000000]	**maroon**	[#800000]
green	[#008000]	**olive**	[#808000]
navy	[#000080]	**purple**	[#800080]
teal	[#008080]	**gray**	[#808080]
silver	[#C0C0C0]	**red**	[#FF0000]
lime	[#00FF00]	**yellow**	[#FFFF00]
blue	[#0000FF]	**fuchsia**	[#FF00FF]
aqua	[#00FFFF]	**white**	[#FFFFFF]

Alignment

The Alignment dialog box enables you to specify the following options:

- how to display text strings that exceed the cell width,
- horizontal and vertical indents (the distance measured from the cell contents to the cell boundaries),
- text rotation,
- horizontal (left, center, right) and vertical (top, center, bottom) alignment; unformatted cells display vertically centered, right-aligned numbers and vertically centered, left-aligned text strings.

To revert to the default settings, click the **Reset** button.

Drop-Down Lists

You can use the Drop-down list dialog box to set the drop-down list format and/or to manage (create, edit, delete) lists in the current workbook.

When editing a field with a drop-down list attached to it, you can quickly choose a predefined cell value instead of re-typing it.
To set that format, simply choose an existing list or click the **New List** button.

To add list items, enter them in the **List Items** edit field: one item per line.

If you choose the **Append new items** option, each unique value entered in the corresponding field(s) will be added to the list automatically.
If the **Sort items** option is not selected, the list will display all its elements in the same order as they were added/entered.
If the **Multiple selection** option is selected, you can select more than one list item. Use the Ctrl and Shift keys (along with clicking) to make such selections. Multiple items are copied to the corresponding cells as comma-separated lists. Multiple selections are not treated as new items and are not added to the list.

One list can be used by any number of fields in any table. You may create up to 255 lists in one database file. To move lists between files, you must copy/paste the data from the **List Items** edit field.

GS-Base Help > *Formatting - hyperlinks*

Use the **Format > Hyperlinks** command to set the hyperlink style for a given field. Clicking such a field or pressing Space will open that link. Hyperlinks can be links to another record/table, web addresses, e-mail addresses, or regular files/folders on your local disk, etc.
You can explicitly specify the link type/protocol by adding one of the following prefixes:

- table://
- http://
- https://

- file://
- mailto:
- ftp://
- gopher://
- nntp://
- news://
- wais://
- telnet://
- prospero://
- res://

Links between individual records can be easily created with the **Edit > Copy As Hyperlink** command. After it's copied, you can paste it in any other text field in any record/table. After the first use of that copy command variant, you can use the **Ctrl+C** shortcut to repeat it.
Links between records can also be entered manually as text. The format is:

```
some-table-name, record-number
```

where the record-number is the physical record number in a given table.
For example:

`table://orders, 4555` (a link to the 4555th record in the "orders" table)

`table://folder1\receipts, 2227` (a link to the 2227th record in the "receipts" table in the folder1 subfolder)

`table://11455` (a link to the 11455th record in the same table)

The **Format > Images** style enables you to enter any sequence of image names in a text field and then display the result in one line in a cell.

The predefined image names (available in every database) include the following buttons, flags and symbols:

- tick
- cross
- flag-green
- flag-red
- flag-white
- star
- star-half
- star-empty
- heart
- heart-half
- heart-empty

For example, entering the following in a cell:

```
star, star, star, star, star-half, star-empty, star-empty
```

will display a 7-star rating with 4.5 stars filled in:

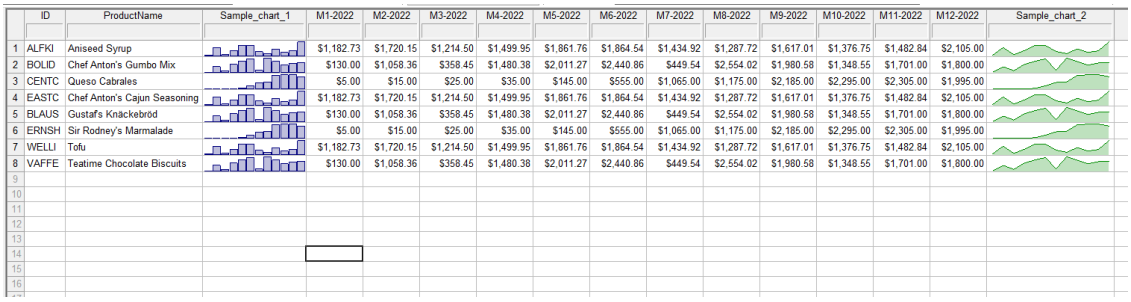
no Taquería		Owner
orn		Sales Repre
abbköp		Order Admir
elikatessen		Sales Repre
et fils		Marketing M

GS-Base Help > *Formatting: charts in record fields*

The **Format > In-Field Chart** command enables you to create any number of cell mini-charts with a single click.
All parameters and display options are set up automatically. You can change the data series color by changing the font color for a given field. The number of charts is practically unlimited: it's 256 million times the number of fields using this format. The maximum series size equals the maximum number of table columns (16K) minus one. By default, you apply this in-field format to field(s) at the end of a data series placed in

a group of subsequent fields/columns. However, the chart(s) can also be displayed at the beginning of such a series by specifying field text alignment to the right.

Sample in-field charts:



	ID	ProductName	Sample_chart_1	M1-2022	M2-2022	M3-2022	M4-2022	M5-2022	M6-2022	M7-2022	M8-2022	M9-2022	M10-2022	M11-2022	M12-2022	Sample_chart_2
1	ALFKI	Aniseed Syrup		$1,182.73	$1,720.15	$1,214.50	$1,499.95	$1,861.76	$1,864.54	$1,434.92	$1,287.72	$1,617.01	$1,376.75	$1,482.84	$2,105.00	
2	BOLID	Chef Anton's Gumbo Mix		$130.00	$1,058.36	$358.45	$1,480.38	$2,011.27	$2,440.86	$449.54	$2,554.02	$1,980.58	$1,348.55	$1,701.00	$1,800.00	
3	CENTC	Queso Cabrales		$5.00	$15.00	$25.00	$35.00	$145.00	$555.00	$1,065.00	$1,175.00	$2,185.00	$2,295.00	$2,305.00	$1,995.00	
4	EASTC	Chef Anton's Cajun Seasoning		$1,182.73	$1,720.15	$1,214.50	$1,499.95	$1,861.76	$1,864.54	$1,434.92	$1,287.72	$1,617.01	$1,376.75	$1,482.84	$2,105.00	
5	BLAUS	Gustaf's Knäckebröd		$130.00	$1,058.36	$358.45	$1,480.38	$2,011.27	$2,440.86	$449.54	$2,554.02	$1,980.58	$1,348.55	$1,701.00	$1,800.00	
6	ERNSH	Sir Rodney's Marmalade		$5.00	$15.00	$25.00	$35.00	$145.00	$555.00	$1,065.00	$1,175.00	$2,185.00	$2,295.00	$2,305.00	$1,995.00	
7	WELLI	Tofu		$1,182.73	$1,720.15	$1,214.50	$1,499.95	$1,861.76	$1,864.54	$1,434.92	$1,287.72	$1,617.01	$1,376.75	$1,482.84	$2,105.00	
8	VAFFE	Teatime Chocolate Biscuits		$130.00	$1,058.36	$358.45	$1,480.38	$2,011.27	$2,440.86	$449.54	$2,554.02	$1,980.58	$1,348.55	$1,701.00	$1,800.00	

GS-Base Help > *Searching / filtering records*

To Set/Modify a Field Filter

Scroll to a given field and press **Ctrl+F** or **Ctrl+Enter** or click the **Search** button or use the **Tools > Set Search Filter** command.
Filters specified for more than one field at the same time are assumed to be merged by the AND logical operator and narrow down the search results.

- The displayed Search dialog box enables you to specify the filter and its type.
  - Regular expressions
  - **Starts with**, text patterns with the * and ? masks
  - Equal, Not Equal, Greater than, Smaller than
  - Between
  - AND, OR
  - IsEmpty (also after removing whitespaces)
  - Similar ("fuzzy" searches)
  - Flagged records (with user-defined flags)
  - Formula (including over 300 functions and references to fields in one record)

To search for the currently selected field value, click the **"="** button or press **Ctrl + =**.

- The default filter type for text fields is **Regular Expression**, and for numeric fields: **Equal**.
- After a given filter type is used for some field, it becomes its default type.
- The default initial filter type can be changed in the **Settings > Options** dialog box.
- **Allow empty matches** and **Match case** are two options used by the **Regular Expression** filter type only.
  Empty matches occur for **Regular Expression** patterns like `\A\z` or `a?`.
- Selecting the default **Search All Now** option causes instant re-searching of all fields after you click OK. Turning it off can be helpful and can prevent unnecessary re-searching, especially when entering relatively many filters (with the limit being the same as the field/column limit: 16K) for large data sets. After you enter them, you just use the **Search All Now (Ctrl+F9)** command.

There are four filter types available for numeric and text fields:

Regular Expressions (RegEx)

The filter string is a data pattern based on standard regular expression syntax. You can click the **"?"** button to display a list of commonly used expressions. Some examples of regular expressions:

Pattern	*Matches*
`abc`	substrings "abc"
`.bc`	three-character substrings consisting of any character followed by "bc"
`\Aabc`	field contents starting with "abc"
`abc\z`	field contents ending with "abc"
`\Aabc.*123\z`	field contents starting with "abc" and ending with "123", with any number of other characters in between

Pattern	*Matches*
`abc\d\z`	substrings ending with "abc" and one digit
`^a\d+`	field contents containing a line starting with "a" and at least one digit
`^a\d*`	field contents containing a line starting with "a" and zero or more digits
`[ab]+c`	substrings "abc", "aabc", "abbabc", etc., but not "c"
`[ab]*c`	substrings "abc", "aabc", "abbabc", etc., and "c"
`[^ab]+c`	substrings ending with "c" and not containing "a" or "b"
`\w\d{2,3}`	substrings consisting of one letter followed by 2 or 3 digits
`\Aab\d{2,3}c`	field contents beginning with "ab", two or three digits, and "c"
`abc|xyz`	substrings "abc" or "xyz"
`\A\z`	empty fields

For more information, see PCRE regular expression syntax summary.

Plain Text (Starts With, Equal, Not Equal, ...)

The filter string represents a plain text string that is compared against the whole field contents. This filter category includes a few variants: **Pattern**, **Equal**, **Not equal**, **Greater than**, **Less than**, **Between**, **And** and **Or**.
For the **Starts with**, **And** and **Or** variants, the filter string can be a prefix of the compared strings and can contain the wildcard "?" and "*" characters. Any non-wildcard "?" and "*" characters must be prefixed with a tilde (~).
The remaining variants perform exact text string comparisons. For **Between**, **And** and **Or**, the entered filter must be a list of, respectively, two or more elements separated by spaces.

Filter string	*Matches*
`abc`	field contents beginning with "abc" ("Starts with")
`?bc`	field contents beginning with any character followed by "bc" ("Starts with")
`*abc`	field contents ending with any character followed by "abc" ("Starts with")
`*`	all non-empty field contents ("Starts with")
`50.1 100.2`	values equal to or between 50.1 and 100.2 ("Between", US regional numeric settings)
`12/1/2012 12/31/2012`	dates equal to or between the specified ones ("Between", US regional date settings)
`ab cd *ef`	field contents equal to "ab", "cd", or ending with "ef" ("Or")

Formulas

The filter string represents a formula expression that can make use of all the built-in functions, operators and references to other record fields. The formulas are similar to those used in spreadsheets, with a few exceptions:

- cell references are replaced by field names,
- the range ":" operator is not available,
- direct references to other tables via the "!" and "_" operators are not available.

When searching, such a formula is evaluated for each record, and a given record is considered to meet the searching criteria if the formula returns a non-zero value.

To learn more about building formulas, see: Entering formulas.

Flagged Records

Searching for records flagged/marked by a given flag, which is a unique combination of field font and background colors.

Long Text, **Images/Files** and **Code** fields always use **Regular Expressions** to filter records. When filtering **Images/Files** fields, the file names are searched.

Clicking the **Paste** button displays a list of recently entered search strings.

To remove all specified search filters, click the **(None)** search key on the toolbar or use the **Tools > Reset Searching** command.

To Filter Records and Search for Duplicated Field Values

Scroll to a field or select a range of fields which should be searched and use the **Tools > Find Duplicates** command.
To find duplicated (whole) records, select an entire row by clicking the row heading or pressing **Shift+Space**.

To Filter Records and Display the Complement of the Current Record Set

To find the complement of the current record set (that is, all the records that are not included in the current record set), use the **Tools > Find Complement** command.

To Perform Full-Text Searches and Search for Patterns Occurring in Any Text and Numeric Fields

Use the **Edit > Find (F3)** command and, in the displayed **Search Toolbar**, enter the desired substring. The **Find Previous/Next** commands simply scroll to the subsequent found table or form cell values. The **Find All** command performs full-text filtering of **Text** and **Numeric** fields (**Long Text** fields are not included).
The **Search Toolbar** also enables you to replace found substrings. If you want to

perform such a find-and-replace action for one field only, select the column by clicking its header.

pcresyntax man page

Return to the PCRE index page.

This page is part of the PCRE HTML documentation. It was generated automatically from the original man page. If there is any nonsense in it, please consult the man page, in case the conversion went wrong.

PCRE REGULAR EXPRESSION SYNTAX SUMMARY

The full syntax and semantics of the regular expressions that are supported by PCRE are described in the **pcrepattern** documentation. This document contains a quick-reference summary of the syntax.

QUOTING

```
  \x         where x is non-alphanumeric is a literal x
  \Q...\E    treat enclosed characters as literal
```

CHARACTERS

```
  \a         alarm, that is, the BEL character (hex 07)
  \cx        "control-x", where x is any ASCII character
  \e         escape (hex 1B)
  \f         form feed (hex 0C)
  \n         newline (hex 0A)
  \r         carriage return (hex 0D)
  \t         tab (hex 09)
  \ddd       character with octal code ddd, or backreference
  \xhh       character with hex code hh
  \x{hhh..}  character with hex code hhh..
```

CHARACTER TYPES

```
  .          any character except newline;
               in dotall mode, any character whatsoever
  \C         one data unit, even in UTF mode (best avoided)
  \d         a decimal digit
  \D         a character that is not a decimal digit
  \h         a horizontal white space character
  \H         a character that is not a horizontal white space character
  \N         a character that is not a newline
  \p{xx}     a character with the xx property
  \P{xx}     a character without the xx property
  \R         a newline sequence
  \s         a white space character
  \S         a character that is not a white space character
  \v         a vertical white space character
  \V         a character that is not a vertical white space character
  \w         a "word" character
  \W         a "non-word" character
  \X         an extended Unicode sequence
```

In PCRE, by default, \d, \D, \s, \S, \w, and \W recognize only ASCII characters, even in a UTF mode. However, this can be changed by setting the PCRE_UCP option.

GENERAL CATEGORY PROPERTIES FOR \p and \P

```
  C          Other
  Cc         Control
  Cf         Format
  Cn         Unassigned
  Co         Private use
  Cs         Surrogate

  L          Letter
  Ll         Lower case letter
  Lm         Modifier letter
  Lo         Other letter
  Lt         Title case letter
  Lu         Upper case letter
  L&         Ll, Lu, or Lt
```

```
M          Mark
Mc         Spacing mark
Me         Enclosing mark
Mn         Non-spacing mark

N          Number
Nd         Decimal number
Nl         Letter number
No         Other number

P          Punctuation
Pc         Connector punctuation
Pd         Dash punctuation
Pe         Close punctuation
Pf         Final punctuation
Pi         Initial punctuation
Po         Other punctuation
Ps         Open punctuation

S          Symbol
Sc         Currency symbol
Sk         Modifier symbol
Sm         Mathematical symbol
So         Other symbol

Z          Separator
Zl         Line separator
Zp         Paragraph separator
Zs         Space separator
```

PCRE SPECIAL CATEGORY PROPERTIES FOR \p and \P

```
Xan        Alphanumeric: union of properties L and N
Xps        POSIX space: property Z or tab, NL, VT, FF, CR
Xsp        Perl space: property Z or tab, NL, FF, CR
Xwd        Perl word: property Xan or underscore
```

SCRIPT NAMES FOR \p AND \P

Arabic, Armenian, Avestan, Balinese, Bamum, Batak, Bengali, Bopomofo, Brahmi, Braille, Buginese, Buhid, Canadian_Aboriginal, Carian, Chakma, Cham, Cherokee, Common, Coptic, Cuneiform, Cypriot, Cyrillic, Deseret, Devanagari, Egyptian_Hieroglyphs, Ethiopic, Georgian, Glagolitic, Gothic, Greek, Gujarati, Gurmukhi, Han, Hangul, Hanunoo, Hebrew, Hiragana, Imperial_Aramaic, Inherited, Inscriptional_Pahlavi, Inscriptional_Parthian, Javanese, Kaithi, Kannada, Katakana, Kayah_Li, Kharoshthi, Khmer, Lao, Latin, Lepcha, Limbu, Linear_B, Lisu, Lycian, Lydian, Malayalam, Mandaic, Meetei_Mayek, Meroitic_Cursive, Meroitic_Hieroglyphs, Miao, Mongolian, Myanmar, New_Tai_Lue, Nko, Ogham, Old_Italic, Old_Persian, Old_South_Arabian, Old_Turkic, Ol_Chiki, Oriya, Osmanya, Phags_Pa, Phoenician, Rejang, Runic, Samaritan, Saurashtra, Sharada, Shavian, Sinhala, Sora_Sompeng, Sundanese, Syloti_Nagri, Syriac, Tagalog, Tagbanwa, Tai_Le, Tai_Tham, Tai_Viet, Takri, Tamil, Telugu, Thaana, Thai, Tibetan, Tifinagh, Ugaritic, Vai, Yi.

CHARACTER CLASSES

```
[...]       positive character class
[^...]      negative character class
[x-y]       range (can be used for hex characters)
[[:xxx:]]   positive POSIX named set
[[:^xxx:]]  negative POSIX named set

alnum       alphanumeric
alpha       alphabetic
ascii       0-127
blank       space or tab
cntrl       control character
digit       decimal digit
graph       printing, excluding space
lower       lower case letter
print       printing, including space
punct       printing, excluding alphanumeric
space       white space
upper       upper case letter
word        same as \w
xdigit      hexadecimal digit
```

In PCRE, POSIX character set names recognize only ASCII characters by default, but some of them use Unicode properties if PCRE_UCP is set. You can use \Q...\E inside a character class.

QUANTIFIERS

```
?           0 or 1, greedy
?+          0 or 1, possessive
??          0 or 1, lazy
*           0 or more, greedy
*+          0 or more, possessive
*?          0 or more, lazy
+           1 or more, greedy
++          1 or more, possessive
+?          1 or more, lazy
{n}         exactly n
{n,m}       at least n, no more than m, greedy
{n,m}+      at least n, no more than m, possessive
{n,m}?      at least n, no more than m, lazy
{n,}        n or more, greedy
{n,}+       n or more, possessive
{n,}?       n or more, lazy
```

ANCHORS AND SIMPLE ASSERTIONS

```
\b          word boundary
\B          not a word boundary
^           start of subject
             also after internal newline in multiline mode
\A          start of subject
$           end of subject
             also before newline at end of subject
             also before internal newline in multiline mode
\Z          end of subject
             also before newline at end of subject
\z          end of subject
\G          first matching position in subject
```

MATCH POINT RESET

```
\K          reset start of match
```

ALTERNATION

```
expr|expr|expr...
```

CAPTURING

```
(...)           capturing group
(?<name>...)    named capturing group (Perl)
(?'name'...)    named capturing group (Perl)
(?P<name>...)   named capturing group (Python)
(?:...)         non-capturing group
(?|...)         non-capturing group; reset group numbers for
                 capturing groups in each alternative
```

ATOMIC GROUPS

```
(?>...)         atomic, non-capturing group
```

COMMENT

```
(?#....)        comment (not nestable)
```

OPTION SETTING

```
(?i)            caseless
(?J)            allow duplicate names
(?m)            multiline
(?s)            single line (dotall)
(?U)            default ungreedy (lazy)
(?x)            extended (ignore white space)
(?-...)         unset option(s)
```

The following are recognized only at the start of a pattern or after one of the newline-setting options with similar syntax:

```
(*NO_START_OPT) no start-match optimization (PCRE_NO_START_OPTIMIZE)
(*UTF8)         set UTF-8 mode: 8-bit library (PCRE_UTF8)
(*UTF16)        set UTF-16 mode: 16-bit library (PCRE_UTF16)
(*UCP)          set PCRE_UCP (use Unicode properties for \d etc)
```

LOOKAHEAD AND LOOKBEHIND ASSERTIONS

```
(?=...)         positive look ahead
(?!...)         negative look ahead
(?<=...)        positive look behind
(?<!...)        negative look behind
```

Each top-level branch of a look behind must be of a fixed length.

BACKREFERENCES

```
\n              reference by number (can be ambiguous)
\gn             reference by number
\g{n}           reference by number
\g{-n}          relative reference by number
\k<name>        reference by name (Perl)
\k'name'        reference by name (Perl)
\g{name}        reference by name (Perl)
\k{name}        reference by name (.NET)
(?P=name)       reference by name (Python)
```

SUBROUTINE REFERENCES (POSSIBLY RECURSIVE)

```
(?R)            recurse whole pattern
(?n)            call subpattern by absolute number
(?+n)           call subpattern by relative number
(?-n)           call subpattern by relative number
(?&name)        call subpattern by name (Perl)
(?P>name)       call subpattern by name (Python)
\g<name>        call subpattern by name (Oniguruma)
\g'name'        call subpattern by name (Oniguruma)
\g<n>           call subpattern by absolute number (Oniguruma)
\g'n'           call subpattern by absolute number (Oniguruma)
\g<+n>          call subpattern by relative number (PCRE extension)
\g'+n'          call subpattern by relative number (PCRE extension)
\g<-n>          call subpattern by relative number (PCRE extension)
\g'-n'          call subpattern by relative number (PCRE extension)
```

CONDITIONAL PATTERNS

```
(?(condition)yes-pattern)
(?(condition)yes-pattern|no-pattern)

(?(n)...        absolute reference condition
(?(+n)...       relative reference condition
(?(-n)...       relative reference condition
(?(<name>)...   named reference condition (Perl)
(?('name')...   named reference condition (Perl)
(?(name)...     named reference condition (PCRE)
(?(R)...        overall recursion condition
(?(Rn)...       specific group recursion condition
(?(R&name)...   specific recursion condition
(?(DEFINE)...   define subpattern for reference
(?(assert)...   assertion condition
```

BACKTRACKING CONTROL

The following act immediately they are reached:

```
(*ACCEPT)       force successful match
(*FAIL)         force backtrack; synonym (*F)
(*MARK:NAME)    set name to be passed back; synonym (*:NAME)
```

The following act only when a subsequent match failure causes a backtrack to reach them. They all force a match failure, but they differ in what happens afterwards. Those that advance

the start-of-match point do so only if the pattern is not anchored.

```
(*COMMIT)       overall failure, no advance of starting point
(*PRUNE)        advance to next starting character
(*PRUNE:NAME)   equivalent to (*MARK:NAME)(*PRUNE)
(*SKIP)         advance to current matching position
(*SKIP:NAME)    advance to position corresponding to an earlier
                (*MARK:NAME); if not found, the (*SKIP) is ignored
(*THEN)         local failure, backtrack to next alternation
(*THEN:NAME)    equivalent to (*MARK:NAME)(*THEN)
```

NEWLINE CONVENTIONS

These are recognized only at the very start of the pattern or after a (*BSR_...), (*UTF8), (*UTF16) or (*UCP) option.

```
(*CR)           carriage return only
(*LF)           linefeed only
(*CRLF)         carriage return followed by linefeed
(*ANYCRLF)      all three of the above
(*ANY)          any Unicode newline sequence
```

WHAT \R MATCHES

These are recognized only at the very start of the pattern or after a (*...) option that sets the newline convention or a UTF or UCP mode.

```
(*BSR_ANYCRLF)  CR, LF, or CRLF
(*BSR_UNICODE)  any Unicode newline sequence
```

CALLOUTS

```
(?C)      callout
(?Cn)     callout with data n
```

SEE ALSO

pcrepattern(3), **pcreapi**(3), **pcrecallout**(3), **pcrematching**(3), **pcre**(3).

AUTHOR

Philip Hazel
University Computing Service
Cambridge CB2 3QH, England.

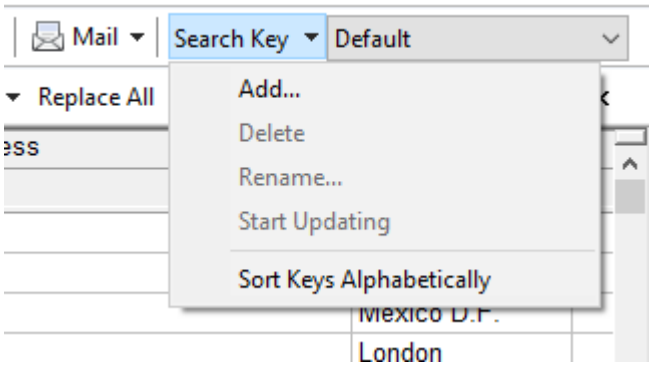
REVISION

Last updated: 10 January 2012

Return to the PCRE index page.

To Save the Current Field Filters as a Named Search Key

Click the **Search Keys** toolbar button and use the **Add...** command from the displayed menu.



To Apply Filters Saved as a Named Search Key

Choose the key name from the **Search Keys** combo box on the main toolbar.

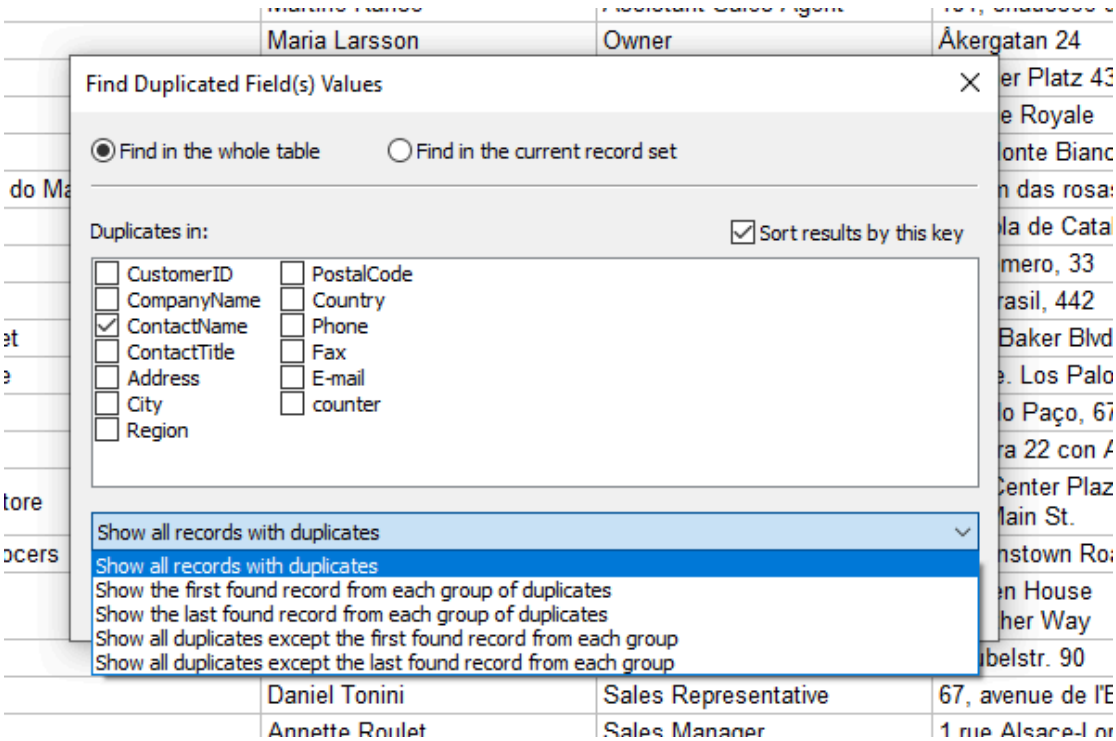
To Modify Filters Saved as a Named Search Key

Select that key, then click the **Search Keys** toolbar button and choose the **Start Updating** command. After adding/deleting/modifying the desired filters, end the updating mode.

To find duplicated values in one or more columns, use the **Tools > Find Duplicates** command. In the displayed dialog box you can specify:

- whether to search for duplicates within the entire table or the current record set

- which record order is to be used for displaying: the physical table order or any sort order
- up to 100 fields to be used as a duplicate key
- how to display the results:
    - the first (entered in the table) records from each group of duplicates,
    - the last (entered in the table) records from each group of duplicates,
    - all records from each group of duplicates except the first one,
    - all records from each group of duplicates except the last one.

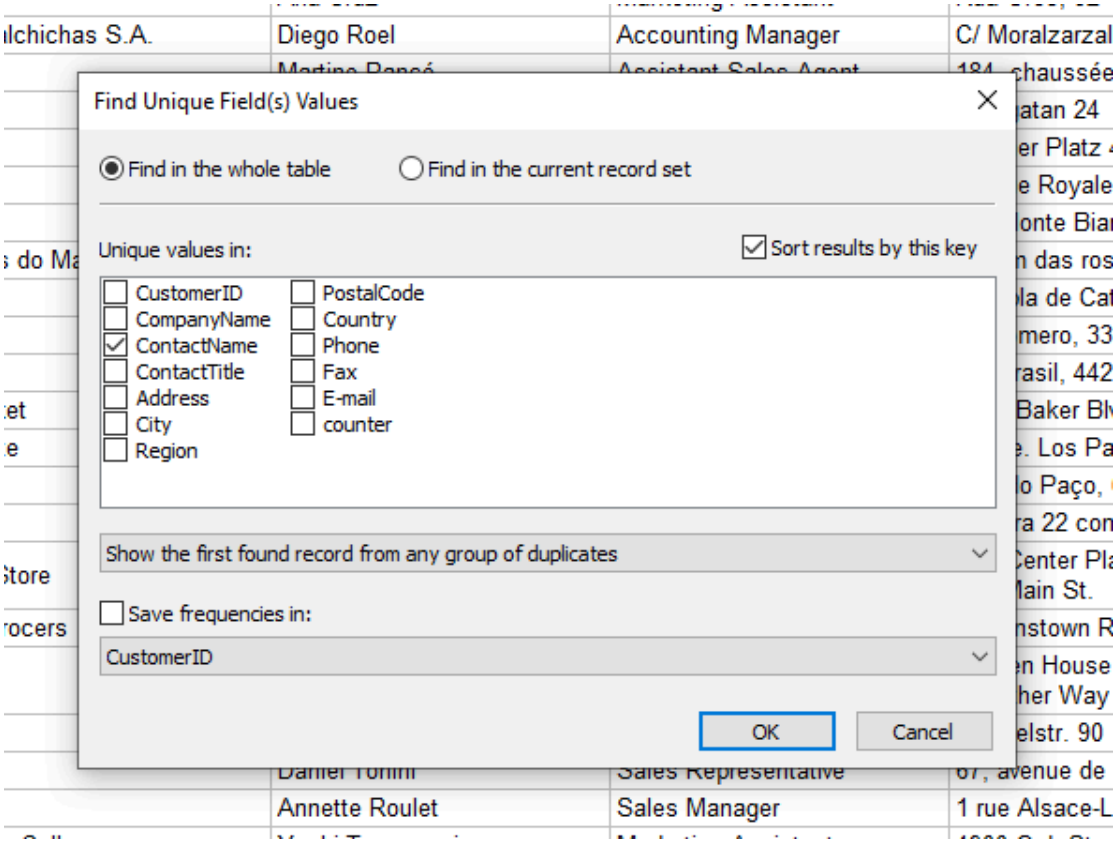


GS-Base Help > *Searching - unique field values*

To find/list all unique values in one or more columns, use the **Tools > Find Unique Values** command. In the displayed dialog box you can specify:

- whether to search for unique values within the entire table or the current record set,

- which record order is to be used for displaying: the physical table order or any sort order,
- up to 100 fields to be used as a duplicate/unique key,
- how to display the results:
  - the first (entered in the table) records from each group of duplicates,
  - the last (entered in the table) records from each group of duplicates,
- a numeric field where the frequencies (number of occurrences of found values) will be saved.



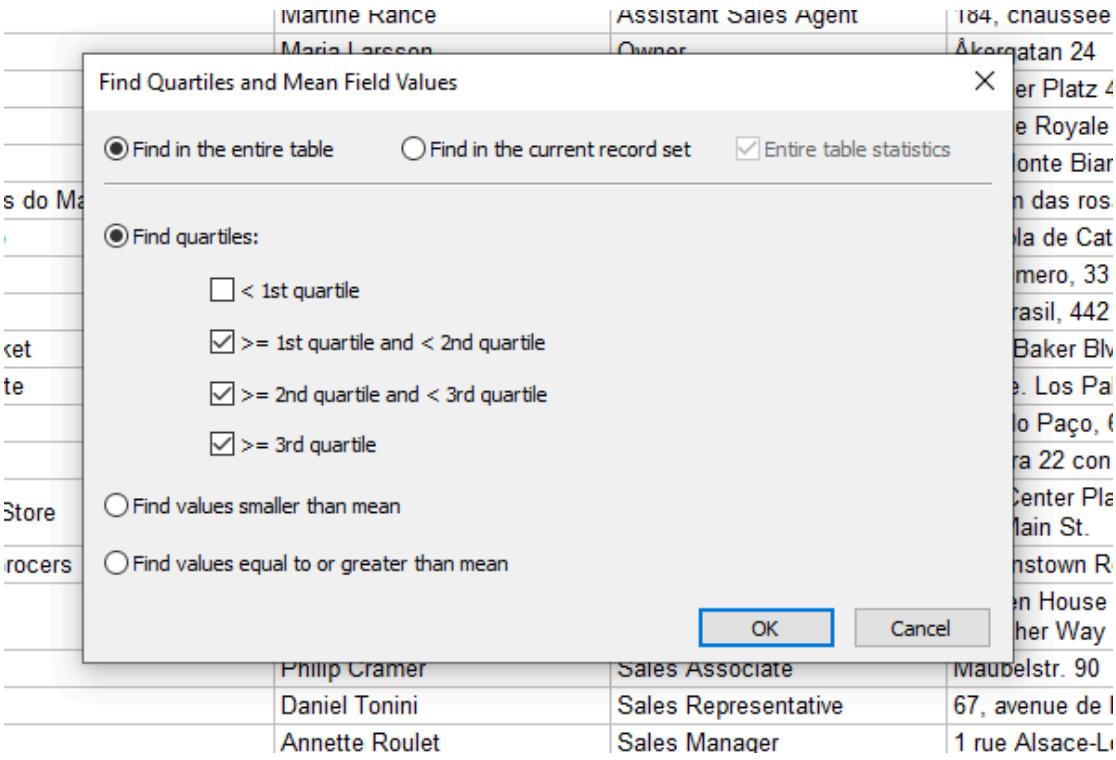
GS-Base Help > *Searching - quartiles, mean values*

To find/list field values from a given quartile or below/above the mean value, use the **Tools > Find Quartiles / Mean** command. In the displayed dialog box you can specify:

- whether to search the entire table or the current record set,
- whether the quartile/mean values used as the search threshold should be calculated from the entire table column, or only from the column values in the current (already filtered) record set.

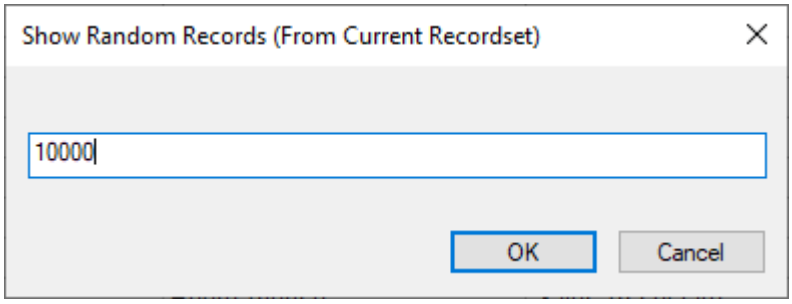
Note: for non-numeric fields the statistics are obtained as follows:

- string lengths for unformatted Text fields,
- numeric date/time values for Text fields formatted as Date/Time,
- the total size in bytes for Long Text and Images/Files fields.



GS-Base Help > *Searching - random records*

Use the **Tools > Show Random Records** command to display a given number of randomly selected records from the current record set.



GS-Base Help > Searching - selected records

Use the **Tools > Show Selected Records** command to display the selected range of records as the current record set.

GS-Base Help > Full text searches and replacing

To perform full-text searching, use the **Edit > Find (F3)** command and, in the **Find** field of the displayed Find & Replace toolbar, enter the data pattern or plain substring to look for.

The **Find Previous/Next** commands simply scroll to the subsequent found table or form cell values. The **Find All** command performs full-text filtering of **Text** and **Numeric** fields (**Long Text** fields are not included).

Depending on which view is currently active (a table, form or a memo/object field), the searching procedure can concern:

- all numeric and text fields of all records contained by the (optionally filtered) table
- all numeric and text fields of the record displayed on the form
- the current memo field

There are three search modes that can be specified using the **Options** button menu:

Regular Expressions

The **Find** string is a data pattern based on standard regular expression syntax. Some examples of regular expressions:

Pattern	*Matches*
`abc`	substrings "abc"
`.bc`	three-character substrings consisting of any character followed by "bc"
`\Aabc`	field contents starting with "abc"
`abc\z`	field contents ending with "abc"
`\Aabc.*123\z`	field contents starting with "abc" and ending with "123", with any number of other characters in between
`abc\d\z`	substrings ending with "abc" and one digit
`^a\d+`	field contents containing a line starting with "a" and at least one digit
`^a\d*`	field contents containing a line starting with "a" and zero or more digits
`[ab]+c`	substrings "abc", "aabc", "abbabc", etc., but not "c"
`[ab]*c`	substrings "abc", "aabc", "abbabc", etc., and "c"
`[^ab]+c`	substrings ending with "c" and not containing "a" or "b"
`\w\d{2,3}`	substrings consisting of one letter followed by 2 or 3 digits
`\Aab\d{2,3}c`	field contents beginning with "ab", two or three digits, and "c"
`abc|xyz`	field contents containing "abc" or "xyz" - logical OR
`(?=.*abc)(?=.*xyz)`	field contents containing "abc" and "xyz" - logical AND

Pattern	*Matches*
`\A\z`	empty fields

For more information, see PCRE regular expression syntax summary.

The **Replace** string can contain:

(1) absolute references (by number) to capturing subpatterns, e.g. \1, \2...
A capturing subpattern (or a "group") is a part of the pattern enclosed in () parenthesis.
(2) \l, \L, \u, \U literals to (binary) switch upper- and lower-case conversion
(3) \r, \n - 'line feed' and 'new line' literals (by default, Alt+Enter and Ctrl+Enter insert the \n character into the text field)
(4) \s - a single space character

Note: to find and replace Unicode/non-ASCII characters, use Unicode regular expressions,
e.g. \p{L} instead of \w, \P{L} instead of \W, etc.

Examples:

Find pattern:	cat
Replace string:	dog
Result:	replaces "cat" with "dog"
Find pattern:	`\\`
Replace string:	`/`
Result:	replaces the "\" characters with "/"
Find pattern:	`\A\z`
Replace string:	NULL
Result:	fills empty fields with the "NULL" string
Find pattern:	`\ANULL\z`
Replace string:	(empty)
Result:	if a field contains only the "NULL" string, deletes its contents

Find pattern: `\b(\w+)(?:\W+\1\b)+`

Replace string: `\1`

Result: removes duplicate words in fields

Find pattern: `(\b\w)(\w*(\W+|\z))`

Replace string: `\u\1\l\2`

Result: converts first letters in words to uppercase, others to lowercase

Find pattern: `(\b\w)(\w*(\W+|\z))`

Replace string: `\1`

Result: creates abbreviations consisting of first letters of words

Find pattern: `(\b\w(\d*))(\w*(\W+|\z))`

Replace string: `\1`

Result: creates abbreviations consisting of first letters of words, leaves full numbers

Find pattern: `\b((\d{1,3}\.){3,3})\d{1,3}\b`

Replace string: `\1\*`

Result: masks an IP address (e.g. 11.12.13.114 to 11.12.13.*)

Find pattern: `(\S*)(\s*)(\S*)(\s*)(\S*)`

Replace string: `\g5\g4\g3\g2\g1`

Result: reverses the order of up to the first 3 words in fields

Find pattern: `\R`

Replace string: `\s`

Result: replaces line breaks with spaces

Find pattern: `ab+`

Database field content: `abcdefaabb`

Replace string:	`x`
Replacement result:	`xcdefax`
Find pattern:	`(.)a+\d{1,3}`
Database field content:	`abc aa0102`
Replace string:	`\1`
Replacement result:	`abc 2`
Find pattern:	`(ab)`
Database field content:	`abcdef ghijk abb123`
Replace string:	`\u\1\l\1xyz\s`
Replacement result:	`ABabxyz cdef ghijk ABabxyz b123`
Find pattern:	`\R`

Database field content:	`abc`
	`def`
	`ghi`

Replace string:	`\s`
Replacement result:	`abc def ghi`

Plain Text - Partial Matching

The **Find** string represents a plain text string that is compared against the beginning of the field contents. The string can contain the wildcard "?" and "*" characters. Any non-wildcard "?" and "*" characters must be prefixed with a tilde (~).

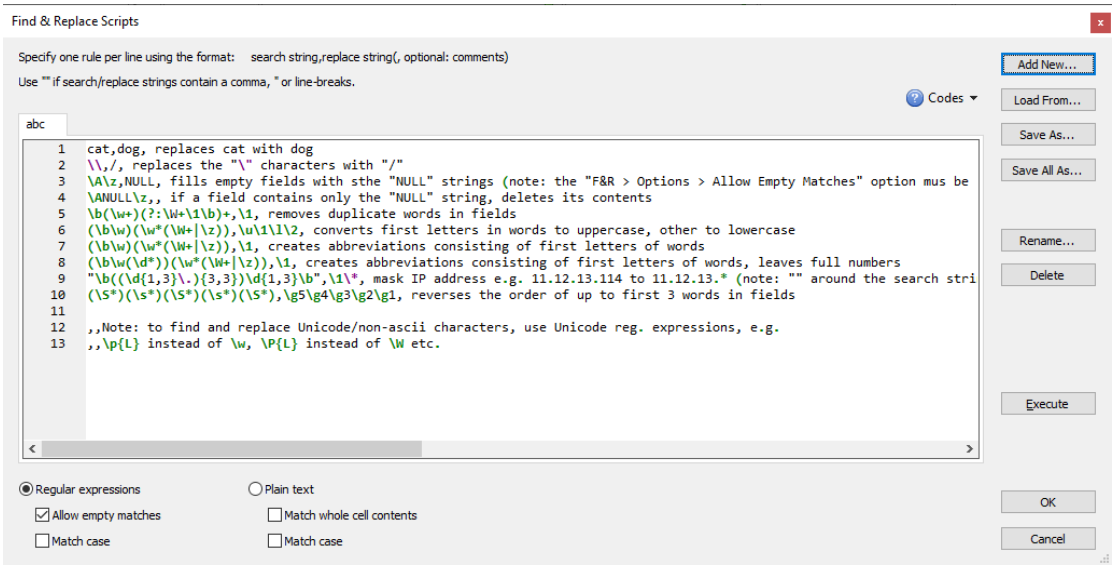
Plain Text - Full Content Matching

The **Find** string represents a plain text string that is compared against the whole field content. The string can contain the wildcard "?" and "*" characters. Any non-wildcard "?" and "*" characters must be prefixed with a tilde (~).

If full-text searching is performed for a single memo field's data, the search mode automatically defaults to the **Plain Text - Partial Matching** search mode.

The **Scripts** button on the **Find & Replace** toolbar can be used to perform quick mass text replacing in a given table.

Note: the "search" and "replace" strings entered in the window below must be formatted like in a CSV text file, so text containing commas, line breaks or quoting symbols must be enclosed in "" and inner quoting symbols must be doubled.



GS-Base Help > Find-As-You-Type and Find-Similar Searching Options

If the **Find As You Type** option is selected, entering/deleting characters in the **Find** edit field results in automatic full-text searching for the whole table.

If the **Find In Current Results** or **Find Similar In Current Results** options are selected, searching is performed within the current record set only, and using the subsequent **Find All** command in these modes performs an **AND** search, progressively narrowing the search results.
The **Find Similar** option matches strings the same way spell checking does, to find suggestions for misspelled words.

By default, the **Find As You Type** mode and the **Find All** command in the other modes enable searching all fields. Making any selection that spans one or more

columns/fields will limit searching to those columns/fields; for example, selecting a range of 1 row x 3 columns will result in searching within those 3 columns/fields only.

GS-Base Help > *Sorting*

To Sort Records

Click the **Sort** toolbar button. If you want to define a multi-field sort key, press and hold down the **Shift** key.

To specify more complex sort keys and to change the default comparison functions, use the **Tools > Sort** command.

When sorting **Long Text**, **Images/Files** and **Code** fields, GS-Base compares the total size of the field contents.
Note: Objects inserted as links (shortcuts) to external files (with the **Insert Shortcut to...** command) will be sorted by size if they were inserted in GS-Base ver. 12.1.3 or later. Unless re-inserted, earlier links will be treated as 1-byte objects.

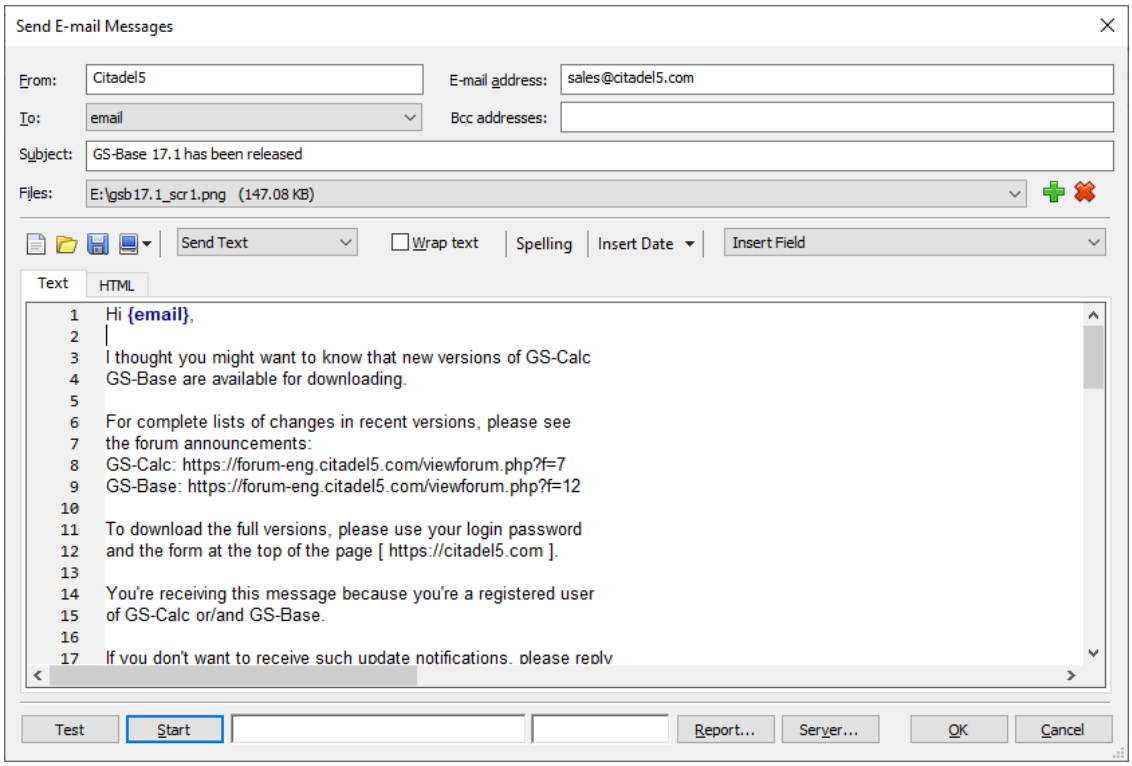
You can use the **Sort > Use the current sort order as the default order** command to make the current sort order the permanent default table order.

GS-Base Help > *Sending email messages*

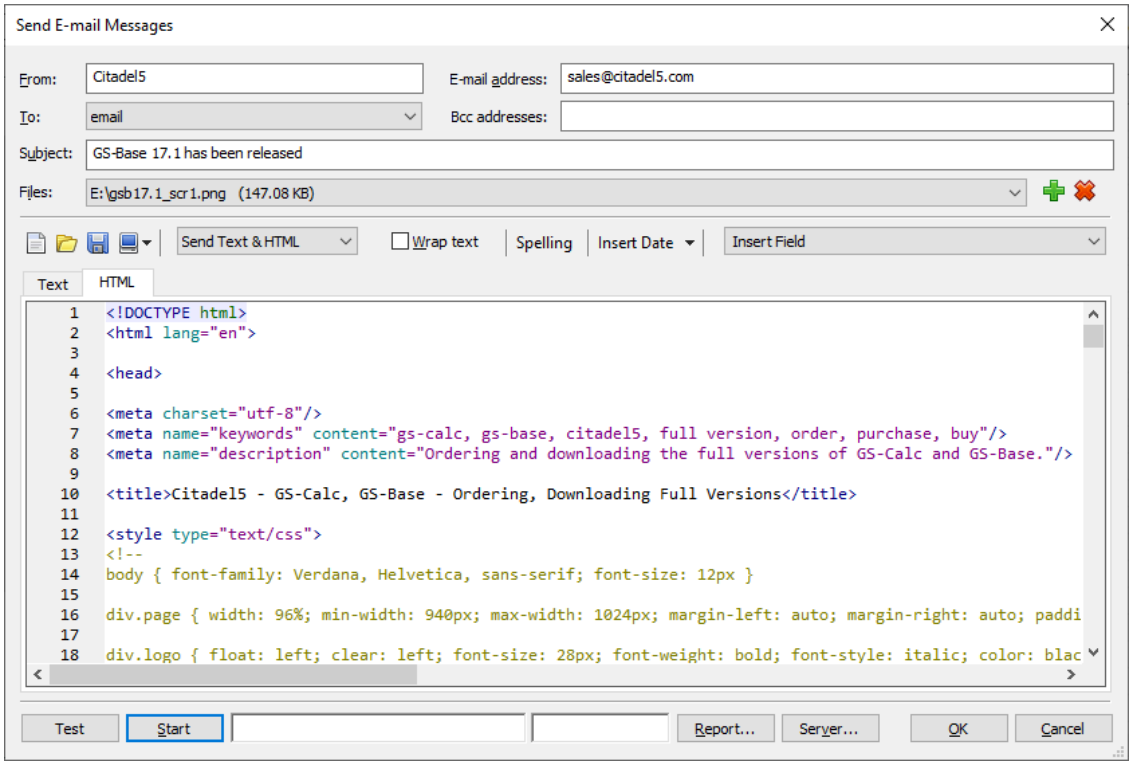
To send messages to addresses entered in a given database field, use the "Tools > Send Mail" command or click the corresponding toolbar button. The messages can be sent either to addresses from all currently filtered records or to addresses included in the currently selected range of records/fields.

The "Send Email Messages" dialog box enables you to specify all the usual mailing parameters:

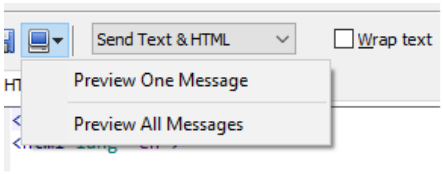
1.

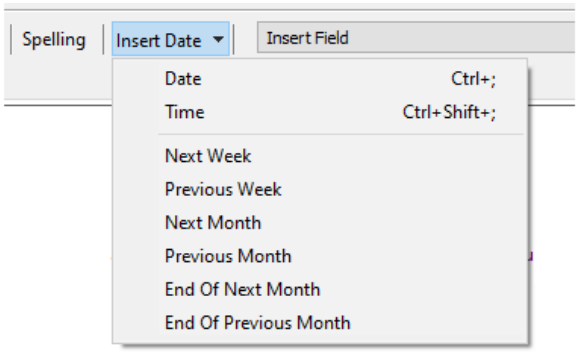

Send E-mail Messages
From: Citadel5
E-mail address: sales@citadel5.com
To: email
Bcc addresses:
Subject: GS-Base 17.1 has been released
Files: E:\gsb17.1_scr1.png (147.08 KB)
Send Text
Wrap text
Spelling
Insert Date
Insert Field
Text
HTML
1 Hi {email},
2
3 I thought you might want to know that new versions of GS-Calc
4 GS-Base are available for downloading.
5
6 For complete lists of changes in recent versions, please see
7 the forum announcements:
8 GS-Calc: https://forum-eng.citadel5.com/viewforum.php?f=7
9 GS-Base: https://forum-eng.citadel5.com/viewforum.php?f=12
10
11 To download the full versions, please use your login password
12 and the form at the top of the page [https://citadel5.com].
13
14 You're receiving this message because you're a registered user
15 of GS-Calc or/and GS-Base.
16
17 If you don't want to receive such update notifications, please reply
Test
Start
Report...
Server...
OK
Cancel

2.


Send E-mail Messages
From: Citadel5
E-mail address: sales@citadel5.com
To: email
Bcc addresses:
Subject: GS-Base 17.1 has been released
Files: E:\gsb17.1_scr1.png (147.08 KB)
Send Text & HTML
Wrap text
Spelling
Insert Date
Insert Field
Text
HTML
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5
6 <meta charset="utf-8"/>
7 <meta name="keywords" content="gs-calc, gs-base, citadel5, full version, order, purchase, buy"/>
8 <meta name="description" content="Ordering and downloading the full versions of GS-Calc and GS-Base."/>
9
10 <title>Citadel5 - GS-Calc, GS-Base - Ordering, Downloading Full Versions</title>
11
12 <style type="text/css">
13 <!--
14 body { font-family: Verdana, Helvetica, sans-serif; font-size: 12px }
15
16 div.page { width: 96%; min-width: 940px; max-width: 1024px; margin-left: auto; margin-right: auto; paddi
17
18 div.logo { float: left; clear: left; font-size: 28px; font-weight: bold; font-style: italic; color: blac
Test
Start
Report...
Server...
OK
Cancel

3.



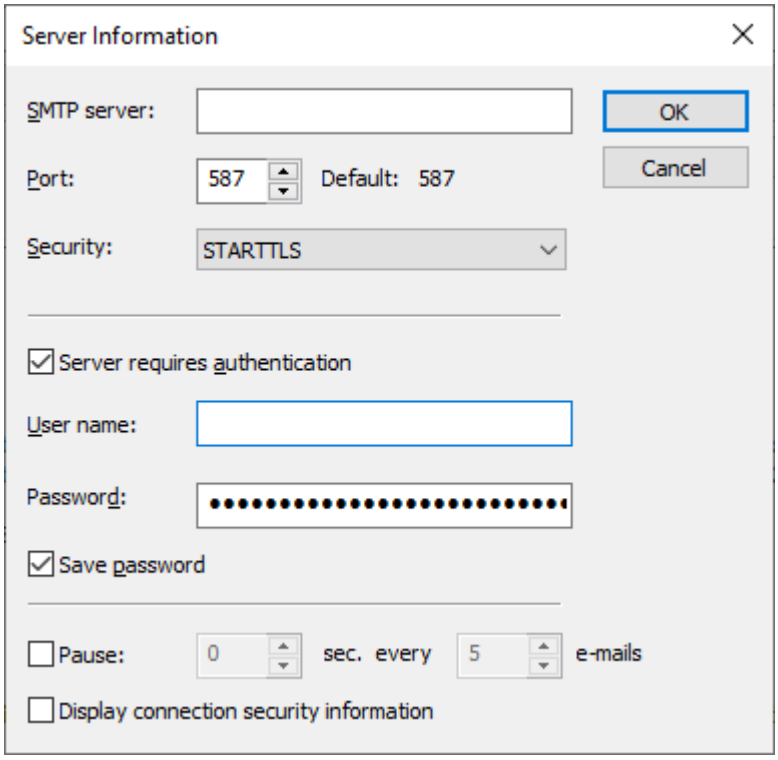


Notes:

- The message body can contain embedded record fields. The corresponding field values from subsequent records are inserted in subsequent sent messages. The embedded fields can be references to fields of each type. For "Long Text" fields, their unformatted contents will be inserted "in-place". For "Images/Files" fields, the objects they contain will be sent as attachments.
- If attachments are the same or mostly the same for each record, use the "Files" static list, which specifies the same attachments for all sent messages. Use the "+" and "x" buttons to add/remove such files.
- You can send messages as plain text, HTML or mixed content. The HTML part must be composed by using the HTML tags directly, and HTML/scripting syntax coloring is provided to make it easier.
- The "Preview" command shows the output as text in the system text editor (e.g. Notepad).
- Clicking the "Test" button results in sending one sample message to the defined "Bcc" address.
- You can pause and continue the process of sending messages at any time.

Setting Up the SMTP Server

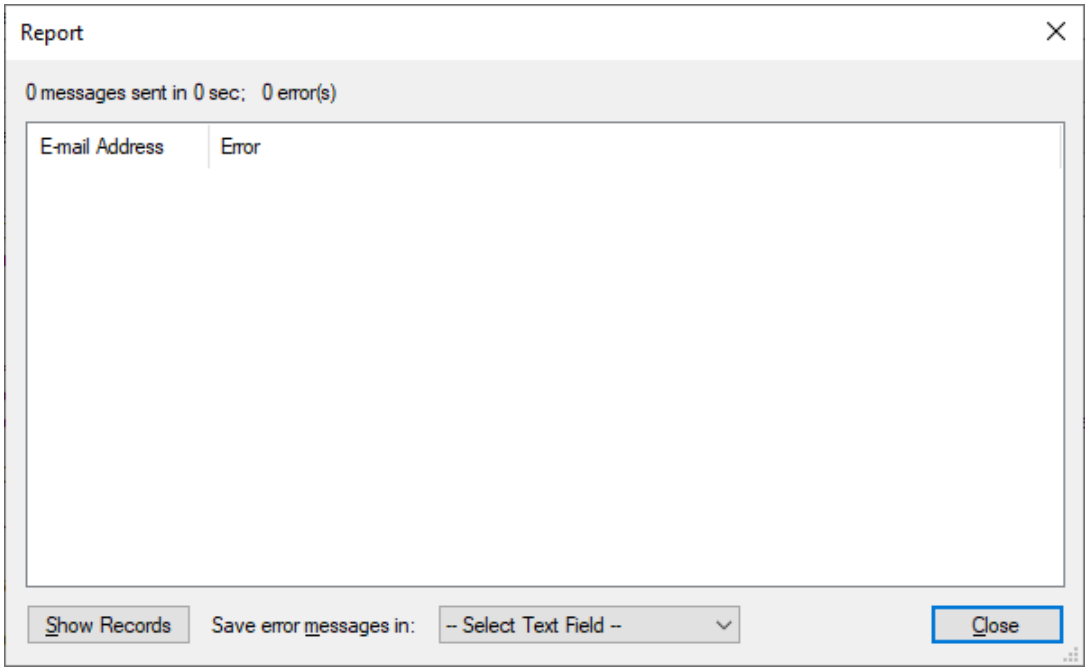
GS-Base sends messages using an SMTP server. All its parameters can be defined in the "Server Information" dialog box:



If "Display connection security information" is on, after clicking the "Start" button GS-Base will show these parameters first, prior to sending anything, and you can choose to terminate the connection.

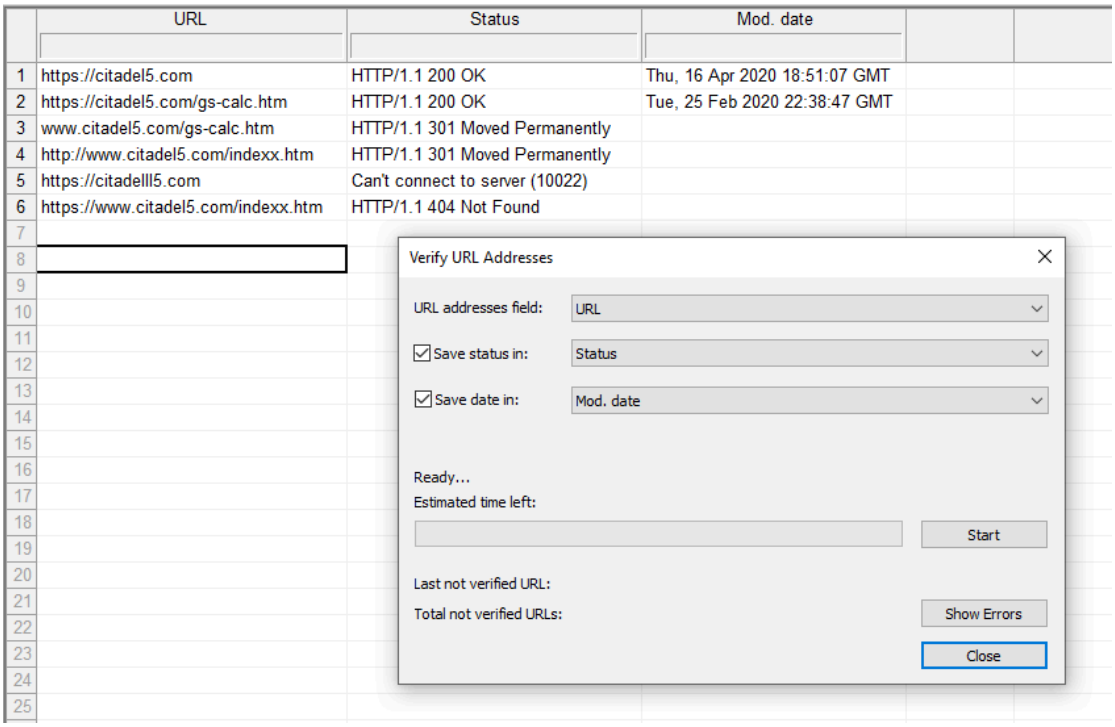
Checking Sending Errors

If sending a message for a given record couldn't be completed by the SMTP server, GS-Base saves the error response codes and shows them after you click the "Report" button.



You can check/verify URLs entered in database fields. GS-Base saves their current status (responses) and the last modification date (if it's available). You can filter the status codes yourself using field filters, or you can let GS-Base verify the returned codes and automatically list all errors/broken URLs.

- You can pause and continue the verification process.
- These can be both http and https URLs, optionally with the ending ":number" port specification.
- Choosing **Cancel** to close that dialog box retains the previous results in the **Status** and **Modification date** fields.



To Print Tables, Forms, Letters or Labels

In the **Print** dialog box, choose the desired print mode by clicking one of the tabs: **Table, Form, Letter, Labels**.

Depending on your selection, you will be able to specify several additional printing options for each print mode.

Table

This is the default print mode. Use it to print selected records in plain tables. **Long Text**, **Image/File** and **Code** fields are not printed in that mode.
To specify which of the text and numeric fields should be printed, click the **Table** tab.

Form

GS-Base will print each record as a fixed-layout form. There are two form variants:

(1) one-page forms, where the first **Long Text**, **Image/File** or **Code** field and the remaining **Text/Number** fields are printed "side by side" on a single page (thus the text can be truncated);
(2) multi-page forms, where all **Long Text**, **Image/File** and **Code** fields are printed on the subsequent pages after all **Text/Number** fields.
If the **Continuous printing** option is selected, GS-Base will continue printing subsequent records/forms up to the bottom of the page. If this option is turned off, each new record/form starts on a new page. This option is disabled if a given table contains **Long Text**, **Images/Files** or **Code** fields.
To specify which field should be printed and to turn on/off printing one-page forms, click the **Form** tab.

Letter

GS-Base will print a user-defined *.rtf form containing any amount of text and graphics with embedded references to text and numeric record fields, and with embedded formulas performing calculations. A field reference is specified by a field name in double-curly braces. A formula must additionally begin with "=".
One copy of such a document is printed for each record of the current record set.
Clicking the **Edit in External Application** command, you can edit the form in the application associated with the *.rtf format. Changes are saved automatically after choosing the **Save** command in that application.

Labels

You must define the required label format. To do this, click the **Labels** tab, then click the **New Labels** button and enter the name of your new label format.
Then specify the number of columns and rows of labels on one page, vertical and horizontal label spacing, and insert the data fields you want to print on each label. Data fields can be simple database field names (references) or formulas (see: Entering formulas).
If the resulting text overflows the field rectangle:

- data entered as formulas is always wrapped, and
- data entered as plain database field references is either wrapped or truncated, depending on the current format of that field.

If you want the formula data to span two or more lines, insert the `char(13)` or `char(10)` (new line/line feed) functions between field values and/or strings in that formula, for example:

```
=Name & char(13) & City
```

When editing labels, you can move or resize the fields. To select a group of fields, press **Shift** and click subsequent fields, or press and hold down the left mouse button and drag the mouse cursor.
To set the default height of the data field rectangle, right-click it.
To rename an existing label format, press **Shift** and click the **New Labels** button.

Select the desired **Page Range** option. You can print all found records, a selected range of records, or particular pages.

To Print Selected Pages

Choose the **Pages** option in the **Print > General** dialog box and specify the list of pages to print.
The list can contain single pages or page ranges separated by commas. For example:

```
1,2,4-8,10
```

To Print More Than One Page on a Single Paper Sheet

Specify the number of printed pages in the **Pages per sheet** fields in the **Print > Layout** dialog box. For example, if you choose two columns and three rows, one sheet will contain (up to) six pages.

To Include a Table Name, Page Number, Date, etc. in the Printed Pages' Headers and Footers

In the **Print > Layout** dialog box, enter the header/footer text, inserting the following special codes:

&p	Prints the current page number
&p+number	Prints the current page number plus 'number'

&p-number	Prints the current page number minus 'number'
&n	Prints the total number of pages
&m	Prints the current paper sheet number (different from &p when printing many pages on one sheet)
&f	Prints the full file path
&a	Prints the name of the printed table
&d	Prints the current date
&t	Prints the current time
&g	Prints the file modification date
&z	Prints the file modification time
&&	Prints a single ampersand

To Format the Header/Footer Text

In the **Print > Layout** dialog box, enter the header/footer text, inserting the following special codes:

&l	Left-aligns the following text
&c	Centers the following text
&r	Right-aligns the following text

To Change the View Scale in the Print-Preview Window

Click the **Zoom In** button, then press and hold the right mouse button and drag to select the rectangle you want to zoom in on.

GS-Base Help > Password protection and encryption

To password-protect and encrypt the data in the database zip file, use the **File > Protect** command and specify a password consisting of at least 6 characters.

To encrypt the data, GS-Base currently uses the Twofish CFB algorithm. All encryption parameters are saved in the **META-INF/manifest.txt** text stream in the database zip file.

If the database zip file is encrypted and you still want to manipulate some of its data manually without GS-Base, using Windows Explorer, you must not alter the **META-INF/manifest.txt** file in any way, or you'll risk losing the data.

Files added by users manually to the zip that are not used/imported by GS-Base remain unencrypted and are moved to the **unused/** zip subfolder.

If a database is encrypted and protected, there is a green shield symbol displayed on the **status bar**.

Regardless of the above file-level protection, you can still use individual field encryption in tables.

To Open a Text File

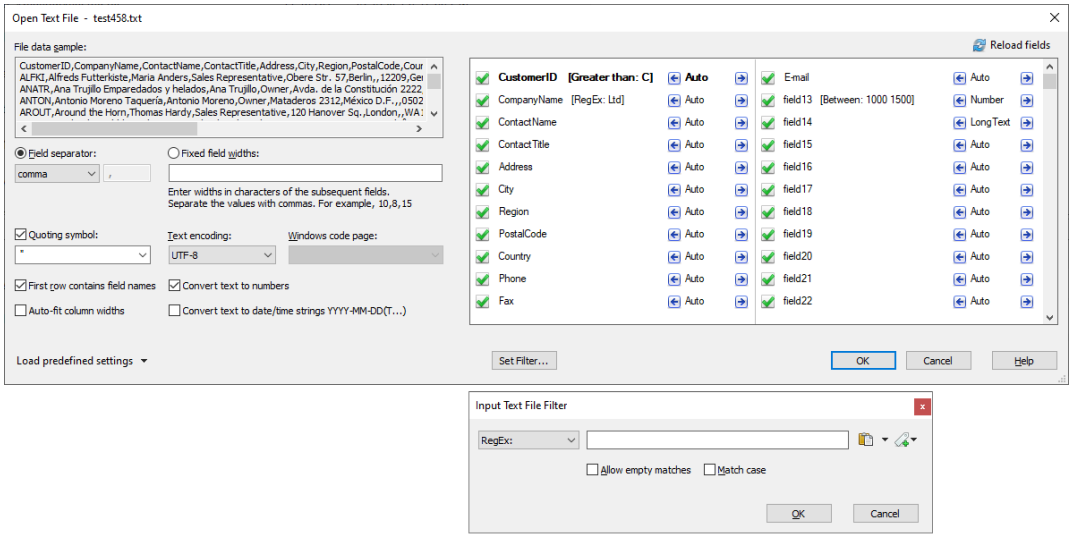
Use the **File > Open** command and choose **Text** from the **File of type** list, or drag and drop a given text file into the main GS-Base window.
Next, specify additional text file options:

Field separator	A single character used to separate fields in one line.

Fixed field widths	A list of fixed field width values. For example: 10,30,15,40 If not all widths are specified, the remaining fields are assumed to contain 1024 characters.
Quoting symbol	A symbol used to delimit fields containing separator characters. **Fields containing quoting symbols, separators or new line characters must be delimited by quoting symbols. The inner quoting symbols must be doubled.** **If you need to open a text file that doesn't conform to this standard, consider specifying a unique quoting symbol that doesn't occur within that file.**
Encoding	Specifies the text encoding for a given text file: UTF-8, UTF-16, ANSI 8-bit, ISO/OEM. The initial value is set based on the file's initial bytes.
First row contains field names	Specifies whether text strings from the first line of the file should be treated as field names. If you clear this option when reading a text file, GS-Base will use simple "field_*nnnn*" names. A field name must begin with a letter.
Convert text to numbers	If you check this option, a column containing **only** text strings that can be converted to numbers (without formatting) will be treated as a numeric database field. Otherwise all such columns will be imported as text

	fields (and will be sorted and searched as plain text fields). You can also change the field type after you import the file using the **Database > Field Properties** command. If automatic field type checking is selected, a given file is scanned twice, with the first pass to make sure there are no conflicts between a given field type and the data.
Convert text to dates	If you check this option, text strings representing dates in one of the local system formats will be converted to generic date/time strings. As selecting this option may significantly slow down opening a text file, it's recommended that you use the **Edit > Convert Field Data > Text To Standard Date String** command instead (for selected fields, after the file is opened). If automatic field type checking is selected, a given file is scanned twice, with the first pass to make sure there are no conflicts between a given field type and the data.
Set Filter	As GS-Base loads files to RAM, to enable processing data from CSV/text files of any size, also for computers with insufficient memory, you can now specify **Text File Input Filters** to limit the size of the loaded data. Only filtered records will be loaded from a given file. Filters can use RegEx patterns, the popular relations, ranges of values and fuzzy filters.

Note: When changing most of the options, you need to click the Reload Fields button to show the number and names of the detected fields.



To Save a Text File

Use the **File > Save Recordset As** command and choose **Text** from the **File of type** list. Next, specify the same additional text file options as above, and the following ones:

Save 'Image/File' objects to a zip file
Save 'Long Text' field contents to a zip file

If you check this option (or these options), the contents of GS-Base binary **Image/File** and **Long Text** fields will be saved to an additional zip file named `[text_file_name].zip` (using the same base name as the saved text table). The corresponding fields in the saved text table will contain file links referring to that zip structure. For example, if some record in the original GS-Base database has a "Memo" **Long Text** field with some long text in it, then in the saved text table — here called `some_text_file.txt` simply as a stand-in for whatever your actual table is named — the "Memo" field

will instead contain a file path such as:

```
some_text_file/r[record_number]\f[field_number]
(e.g.
some_text_file/r00000002f00003\text.txt)
```

If you re-import the `some_text_file.txt` text table, unzip the `some_text_file.zip` file and use the **Format > Hyperlink** command to format that file link text field. Clicking that link will then open an application associated with *.txt files and show that former GS-Base **Long Text** field's contents. The same applies to images and other file types.

Note that, depending on your database size, the above may result in creating a zip file with a very large number of streams/files, and native Windows tools (e.g. File Explorer) will require a lot of time to unzip it.

Leaving this option unchecked results in saving the text file only with labels describing the binary contents.

Note: When saving a text file, GS-Base 14.5.3 and earlier versions save only filtered records from the active table, and GS-Base 14.7 and later versions save the whole table using the default (unsorted) record order. To save only the current filtered and sorted record set, you have to use the **Save Recordset As** command.

To automatically resize the column widths to fit the field contents in the opened file, select the **Options > General > Auto-fit column widths in imported files** option. Keep in mind that for very large files this may slow down the import

process. To optimize the resizing, in the **Options > General** dialog box, increase the declared number of processor cores that GS-Base can use.

Related Topics

Saving PDF files
Opening and saving HTML files
Opening and saving xBase files
Opening and saving Excel workbooks

1. Saving Data as PDF

To save data in PDF format, use one of the following methods:

- **File > Save Table As PDF** and **File > Save All Tables As PDF** — the first saves the current worksheet, the second saves the entire workbook. Page layout and content options can be adjusted via **File > Page Settings**.

2. Font Mapping

Note: When saving a PDF, GS-Base maps all workbook fonts to the standard set of 14 PostScript Type 1 fonts. These fonts are guaranteed to exist in all PDF viewers and operating systems.

3. Advanced PDF Save Options

Use **File > Advanced PDF Save Options** to specify:

- the language/code page used when saving text,
- a custom character encoding list to ensure correct glyph display.

4. Example Encoding Lists

iso-8859-1

```
33 /exclam /quotedbl /numbersign /dollar /percent /ampersand
/quotesingle
/parenleft /parenright /asterisk /plus /comma /hyphen /period
```

```
/slash /zero /one
/two /three /four /five /six /seven /eight /nine /colon /semicolon
/less /equal
/greater /question /at /A /B /C /D /E /F /G /H /I /J /K /L /M /N /O
/P /Q /R /S
/T /U /V /W /X /Y /Z /bracketleft /backslash /bracketright
/asciicircum /underscore
/grave /a /b /c /d /e /f /g /h /i /j /k /l /m /n /o /p /q /r /s /t
/u /v /w /x /y /z
/braceleft /bar /braceright /asciitilde /bullet /Euro /bullet
/quotesinglbase /florin
/quotedblbase /ellipsis /dagger /daggerdbl /circumflex /perthousand
/Scaron
/guilsinglleft /OE /bullet /Zcaron /bullet /bullet /quoteleft
/quoteright /quotedblleft
/quotedblright /bullet /endash /emdash /tilde /trademark /scaron
/guilsinglright
/oe /bullet /zcaron /Ydieresis /space /exclamdown /cent /sterling
/currency /yen
/brokenbar /section /dieresis /copyright /ordfeminine
/guillemotleft /logicalnot
/hyphen /registered /uni203E /degree /plusminus /twosuperior
/threesuperior
/acute /mu /paragraph /periodcentered /cedilla /onesuperior
/ordmasculine
/guillemotright /onequarter /onehalf /threequarters /questiondown
/Agrave /Aacute
/Acircumflex /Atilde /Adieresis /Aring /AE /Ccedilla /Egrave
/Eacute /Ecircumflex
/Edieresis /Igrave /Iacute /Icircumflex /Idieresis /Eth /Ntilde
/Ograve /Oacute
/Ocircumflex /Otilde /Odieresis /multiply /Oslash /Ugrave /Uacute
/Ucircumflex
/Udieresis /Yacute /Thorn /germandbls /agrave /aacute /acircumflex
/atilde /adieresis
/aring /ae /ccedilla /egrave /eacute /ecircumflex /edieresis
/igrave /iacute
/icircumflex /idieresis /eth /ntilde /ograve /oacute /ocircumflex
/otilde /odieresis
/divide /oslash /ugrave /uacute /ucircumflex /udieresis /yacute
/thorn /ydieresis
```

Windows-1250

```
32/space 40/parenleft/parenright
44/comma/hyphen/period/slash/zero/one/two/three/four
/five/six/seven/eight/nine/colon 65/A/B/C/D/E/F/G/H/I/J/K/L/M/N/O/P
82/R/S/T/U
87/W/X/Y/Z 97/a 99/c/d/e/f/g/h/i/j/k/l/m/n/o/p 114/r/s/t/u 119/w
121/y/z 140/Sacute
143/Zacute 156/sacute 159/zacute 163/Lslash 165/Aogonek
175/Zdotaccent 179/lslash
185/aogonek 191/zdotaccent 202/Eogonek 198/Cacute 209/Nacute
211/Oacute
230/cacute 234/eogonek 241/nacute 243/oacute
```


To Open a MySQL *.sql File

Use the **File > Open** command and choose **MySQL (*.sql)** from the **File of type** list, or drag and drop a given *.sql file into the main GS-Base window.
Data from *.sql files can also be accessed via the **Add Table** and **Merge Records** commands.

The *.sql dump file must use the **INSERT INTO** command and must include the preceding table definitions.

The preferable method of saving data in binary **BLOB** type fields is hexadecimal (ASCII text).
If data is saved in **BLOB** fields directly as binary streams, GS-Base decodes the following escape sequences:
`\0, \n, \r, \, ', ", \z`

To Save a MySQL *.sql File

If the open database is already in the *.sql format, the **Save** command causes saving it back to the same file using the same, original table (field, keys) definitions. Character sets are converted to **utf8mb4**.

If the current format is not MySQL *.sql, use the **File > Save Database Copy As** or **File > Save Recordset As** commands and choose **MySQL (*.sql)** from the

File of type list. In this case, the newly created MySQL table definitions will follow the rules specified in **Notes** below.

Notes

- When editing existing MySQL *.sql files and saving them in this format, GS-Base preserves MySQL table (field, keys) definitions, except that used characters are converted to **utf8mb4**.
  New fields added to such a file in GS-Base are subject to the rules listed below.

- MySQL numeric fields become **Number** fields in GS-Base.
  The **Number** GS-Base fields are exported to new MySQL *.sql files as **DOUBLE** fields.

- MySQL **CHAR** and **VARCHAR** fields shorter than 8162 bytes become **Text** fields in GS-Base.
  GS-Base **Text** fields are saved as MySQL **VARCHAR(255)**, but no truncation is performed for data within the range 256-8162.

- MySQL **VARCHAR** fields longer than 8162 bytes, and all MySQL **[...]TEXT** fields, are used in GS-Base as **Long Text** fields.
  GS-Base **Long Text** fields are saved as MySQL **LONGTEXT** fields.

- The binary MySQL **BLOB** field contents from *.sql files not created by GS-Base are represented in GS-Base (in **Images/Files** fields) by files with the "sql_blob.bin" name.
  Data from GS-Base **Images/Files** fields containing just one such object will be saved back to the MySQL file as the same, original binary MySQL contents.
  If a GS-Base **Images/Files** field contains more than one inserted object, or the object name is not "sql_blob.bin", the resulting MySQL **BLOB** field will contain binary data representing the list of these objects in the following format:
  1. 36-character GS-Base GUID string: `EBDA2E02-5502-43C4-AF43-44EEF3375275`

  2. a repeating list of the elements:

     - 5 ASCII characters representing the object/file name length,

        - the file name,
        - 12 ASCII characters representing the object/file data length,
        - the file contents

- Files saved by GS-Base require the MySQL system **max_allowed_packet** variable to be at least (approx.) 64MB. This is the default value for MySQL 8.0.
  If some of your saved MySQL single BLOB/Text fields contain more data, this variable must be increased accordingly.

- If a file in the MySQL *.sql format is opened, the Database Explorer tree also displays the SQL field definitions along with the corresponding GS-Base field types.

To automatically resize the column widths to fit the field contents in the opened file, select the **Options > General > Auto-fit column widths in imported files** option. Keep in mind that for very large files this may slow down the import process. To optimize the resizing, in the **Options > General** dialog box, increase the declared number of processor cores that GS-Base can use.

Related Topics

Saving PDF files
Opening and saving text files
Opening and saving HTML files
Opening and saving Excel workbooks

Opening SQLite *.db Files

Use the **File > Open** command and choose **SQLite (*.db)** from the **File of type** list, or drag and drop a given *.db file into the main GS-Base window.
Data from *.db files can also be accessed via the **Add Table** and **Merge Records** commands.

Saving SQLite *.db Files

If the open database is already in the *.db format, the **Save** command causes saving it back to the same file using the same, original table (field, keys) definitions.

If the current format is not SQLite *.db, use the **File > Save Database Copy As** or **File > Save Recordset As** commands and choose **SQLite (*.db)** from the **File of type** list. In this case, the newly created SQLite table definitions will follow the rules specified in **Notes** below.

Notes

- When editing existing SQLite *.db files and saving them in this format, GS-Base preserves SQLite table (field, keys) definitions.
  New fields added to such a file in GS-Base are subject to the rules listed below.

- SQLite numeric fields become **Number** fields in GS-Base.
  Newly added **Number** GS-Base fields are saved to new SQLite *.db files as **DOUBLE** fields.

- SQLite **TEXT** fields shorter than 8169 bytes become **Text** fields in GS-Base, and longer SQLite **TEXT** fields are added as **Long Text** fields in GS-Base.
  Thus, if you export GS-Base **Long Text** contents shorter than 8169 to SQLite and open that SQLite file again in GS-Base, the previous GS-Base **Long Text** contents will be shown as **Text** fields, and to switch again to the **Long Text** pane view you need to change that field type in the **Field Setup** dialog box. Please also note that unless you turn off formatting in **Long Text** fields using **Settings > Options > Editing > Strip off formatting in Long Text fields**, the formatting RTF tags will be preserved and visible in the **Text** fields.

  GS-Base **Text** and **Long Text** fields are saved as SQLite **TEXT** fields.

- The binary SQLite **BLOB** field contents from *.db files not created by GS-Base are represented in GS-Base (in **Images/Files** fields) by files with the "sql_blob.bin" name.
  Data from GS-Base **Images/Files** fields containing just one such object

with that name will be saved back to the SQLite file as the same, original binary SQLite contents.
If a GS-Base **Images/Files** field contains more than one inserted object, or the object name is not "sql_blob.bin", the resulting SQLite **BLOB** field will contain binary data representing the list of these objects in the following format:

1. 36-character GS-Base GUID string: `EBDA2E02-5502-43C4-AF43-44EEF3375275`
2. a repeating list of the elements:
    - 1 byte with the object/file name length,
    - the file name,
    - 8 bytes (representing the binary INT64 number) of the object/file data length,
    - the file contents

- If a file in the SQLite *.db format is opened, the Database Explorer tree also displays the SQL field definitions along with the corresponding GS-Base field types.

To automatically resize the column widths to fit the field contents in the opened file, select the **Options > General > Auto-fit column widths in imported files** option. Keep in mind that for very large files this may slow down the import process. To optimize the resizing, in the **Options > General** dialog box, increase the declared number of processor cores that GS-Base can use.

Related Topics

Saving PDF files
Opening and saving text files
Opening and saving HTML files
Opening and saving Excel workbooks

To Open an HTML File

Use the **File > Open** command and choose **HTML** from the **File of type** list. Similarly to the saving process described below, GS-Base will load the first table that has the "id" attribute set to "gsbase", e.g.:

```
<TABLE id="gsbase">
```

To Save Records to an HTML File

Use the **File > Save Recordset As** command and choose **HTML** from the **File of type** list. Next, specify additional HTML file options:

Option	*Description*
Template	Specifies an optional HTML file that should be used as a template. The file must contain a `TABLE` element with the ID attribute set to "gsbase", e.g. `<TABLE id="gsbase"> ...` GS-Base will empty that table and fill it with the specified records. All attributes of the `TR`, `TH` and `TD` tags are removed; you should specify column formatting using the `COLGROUP` and `COL` tags instead. If you don't provide any template, GS-Base will create and save a basic/minimal HTML document containing a table filled with records. Note that **Image/File** fields are not saved.

Option	*Description*
Saved fields	Specifies which fields should be saved.
Save duplicated field values as	Specifies a text string that should be used to replace the same values occurring in the same field and subsequent records.
Encoding	Specifies text encoding. Currently the only valid value is UTF-8.
First row contains field names	If you check this, the first row of the saved table will contain field names as `TH` elements.

Related Topics

Opening and saving text files
Opening and saving xBase files
Opening and saving Excel workbooks

To Open an xBase File

Use the **File > Open** command and choose **dBaseIII, dBaseIV, FoxPro 2.x or Clipper** from the **File of type** list, or drag and drop a given xBase file into the main GS-Base window.
Next, specify additional xBase file options:

Encoding	Specifies the text encoding for a given xBase file: ANSI 8-bit or ISO/OEM.

dBase/FoxPro/Clipper records marked as "deleted" are displayed with the "Selected" flag (the default "0" flag for each GS-Base database).

To Save an xBase File

Use the **File > Save Recordset As** command and choose **dBaseIII, dBaseIV, FoxPro 2.x or Clipper** from the **File of type** list.

Note: When saving an xBase file, GS-Base 14.5.3 and earlier versions save only filtered records from the active table, and GS-Base 14.7 and later versions save the whole table using the default (unsorted) record order. To save only the current filtered and sorted record set, you have to use the **Save Recordset As** command.

Records marked with the "Selected" flag (the default "0" flag for each GS-Base database) will be saved to dBase/FoxPro/Clipper databases as "deleted" records.

To automatically resize the column widths to fit the field contents in the opened file, select the **Options > General > Auto-fit column widths in imported files** option. Keep in mind that for very large files this may slow down the import process. To optimize the resizing, in the **Options > General** dialog box, increase the declared number of processor cores that GS-Base can use.

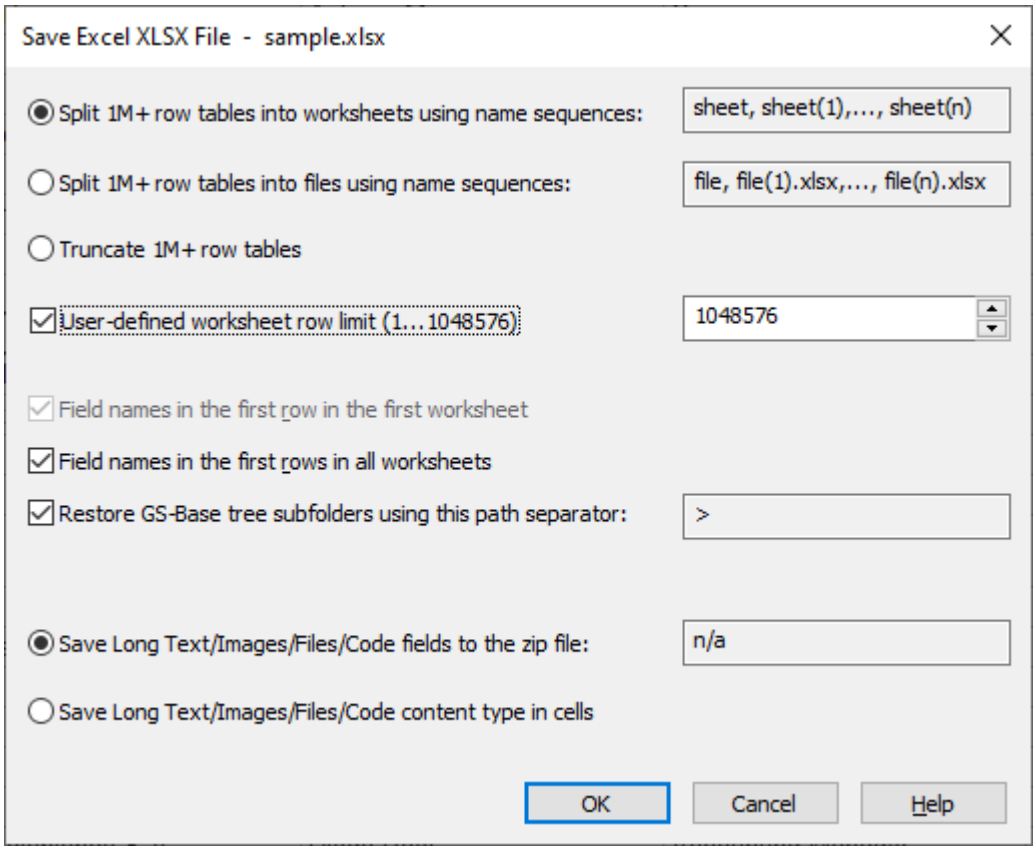
Related Topics

Saving PDF files
Opening and saving text files
Opening and saving HTML files
Opening and saving Excel workbooks

To Save a Database (or Just the Current Filtered Table) as an Excel Workbook

Use the **File > Save Database Copy As** or the **File > Save Record Set As** command and choose the **Excel *.xlsx** format or the **Excel 97-2003 *.xls** file format from the **File of type** list.

In the **Save Excel File** dialog box, specify the following options:



- **Split 64K+ row tables using name sequences**

  or

  Split 1M+ row tables using name sequences

  Tables with records exceeding the Excel row limit will be split into multiple Excel worksheets.
  For example, if a database contains three tables "customers", "orders", "products" and the "orders" table has more than 1M/64K records, then the saved Excel file will contain the following worksheets:

  ```
  customers, orders, orders(1), orders(2), ..., products
  ```

- **Split 1M+ row tables into files using name sequences** (for Excel *.xlsx files)

or

Split 64K+ row tables into files using name sequences (for Excel 97-2003 *.xls files)

If this option is selected, and if the number of records in one or more of the saved tables exceeds the Excel row limit, GS-Base will split such tables and save multiple Excel workbooks:

```
file_name.xlsx (or file_name.xls)
file_name(1).xlsx (or file_name(1).xls)
file_name(2).xlsx (or file_name(2).xls)
...
file_name(n).xlsx (or file_name(n).xls)
```

If there are any previously saved files with larger indices (e.g. file_name(n+1).xls and on), they will be deleted.
If you open the file_name.xls file again in GS-Base, the remaining partial workbooks will be automatically (internally) loaded to form the original GS-Base tables.

- **Save field names in the first row in all first worksheets**

  If this option and the 1st split option are selected, only the first worksheet from each split sequence will contain table field names in the first row.

  If this option and the 2nd split option are selected, the first row in each worksheet in each split Excel workbook will contain the corresponding table field names.

- **Save field names in the first row in subsequent split worksheets**

  In addition to the above, worksheets that were split within one workbook will also contain that top row with table field names.

- **Enable restoring the GS-Base folder structure [...]**

  If this option is selected, and if you created folders in the GS-Base database explorer pane for some tables, the Excel sheet names will be formed as whole such table tree paths, which will enable GS-Base to re-create the original tree folder structure when you open the saved Excel *.xls workbook back in GS-Base.

As the "\" path separator cannot be used in Excel names, it's replaced with ">".

- Binary field options:
  - **Save Long Text/Images/Files/Code fields to the zip file [...]**

    The contents of the above fields will be saved to the

    ```
    file_name.gsb-bin.zip
    ```

    file. You need to keep the saved Excel *.xls workbook(s) and that file together (in the same folder). The corresponding sheet cells in the *.xls workbook will contain links to objects in that zip. To access them, unzip it to the

    ```
    file_name.gsb-bin
    ```

    folder. Clicking a sheet cell formatted as a hyperlink (in GS-Base: **Format > Hyperlink**) will open the field contents (e.g. text, image or a folder with multiple images, etc.). You can edit/update/delete them and distribute your file_name.xls workbook either with the unzipped or zipped folder:

    ```
    file_name.gsb-bin
    ```

  - **Save only Long Text/Images/Files/Code field content description in sheet cells**

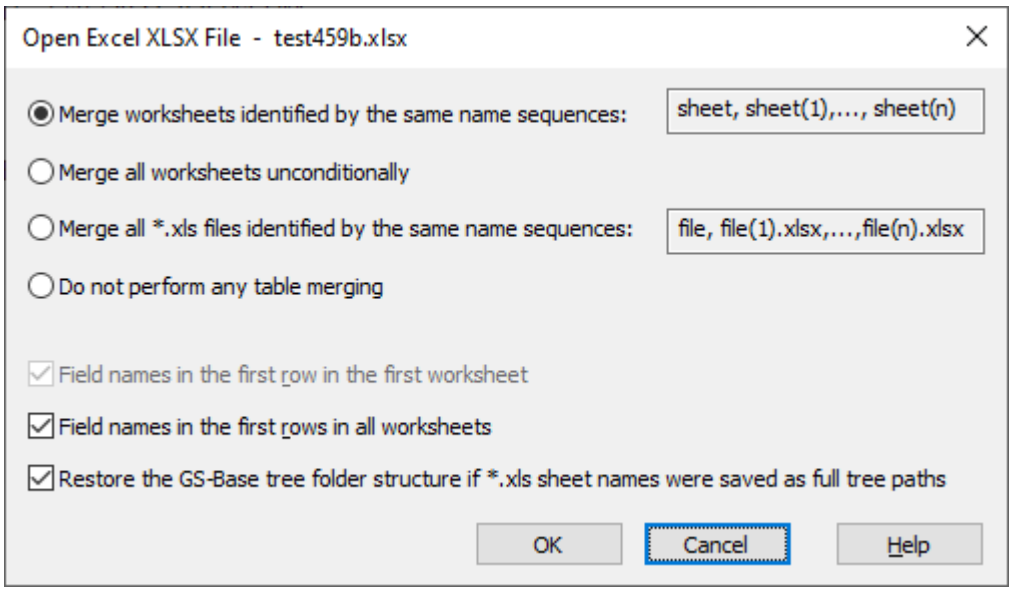
    The description includes the object names, number, total field size and recent modification date.

Note: After saving a document in a different format, you may need to use the **File > Reload** command to refresh the view.

To Open an Excel Workbook

Use the **File > Open** command (or alternatively any of the external table merging commands) and choose the **Excel *.xlsx** format or the **Excel 97-2003 *.xls** format from the **File of type** list.

In the **Open Excel File** dialog box, specify the following options:



- **Merge worksheets identified by name sequences**

  It's the reverse of the 1st split option in the above "Save" dialog box.

- **Merge all worksheets unconditionally**

  If this option is selected, all worksheets are assumed to be a sequence to be merged, regardless of their names.

- **Merge tables from previously split multiple Excel *.xls files [...]**

  See the mirror option described above in the "Save" dialog box.

- **Field names in the first row in all first worksheets**

  If this option is selected, the first row of the first worksheets from each merged sequence has to include the corresponding record field names.

- **Field names in the first row in subsequent split worksheets**

  In addition to the above, all worksheets in each merged worksheet sequence also has to include that top row with field names.

- **Restore the GS-Base tree structure [...]**

  See the mirror option described above in the "Save" dialog box.

Note: if subsequent worksheets in the same merged sequence have more columns, empty columns are added automatically to the preceding worksheets.

Related Topics

GS-Base zip database file format

All GS-Base global settings are stored in the settings.xml file. Its location depends on how you install the application. If you choose the portable installation, the settings file will be saved directly to the installation folder. Otherwise, it's stored in the local application data folder (e.g. C:\Users*username*\AppData\Local\GS-Base).

You can use the **Settings > Save - Load Profile** commands to save or load different settings profile files.

The following data can be specified in the **Settings > Options** dialog:

General Settings

Undo/Redo Level

The number of actions that can be undone via the **Edit > Undo-Redo** commands.
A very high undo level causes greater memory usage if some very large copy/paste operations are performed.
The drag-and-drop operation is treated as a two-step action, so the minimum undo level is 2.
Valid range: <2, 100>
Default: 20

Number of CPUs

The number of processor cores that GS-Base is allowed to use when updating calculated fields (**Tools > Update All Calculated Fields**) and when filtering records (excluding full-text searches in memo fields).
Valid range: <1, 100>
Default: 2

Maximum Number of Records

Specifies the maximum number of records a single table can contain. For each running instance, GS-Base initially allocates **"maximum number of records" * 4** bytes.
If your databases don't contain large numbers of records, you can choose some small limit to save memory.
This setting doesn't affect the database file format. After increasing the limit, any existing database can be expanded further up to the new limit. Available values:

- 64 K (65,536)
- 1 M (1,048,576)
- 12 M (12,582,912 **default**)
- 32 M (33,554,432)
- 64 M (67,108,864)
- 128 M (134,217,728)
- 256 M (268,435,456)

If a given database/file contains more records than the current GS-Base limit allows, a warning message is displayed.

Default Text Field Filter

Specifies the default filter type for text fields. This filter type will be initially set in the Search dialog box.
Default: Regular Expression

Default Pivot Function

Specifies a function that will be used by each new or existing pivot table that doesn't have any data field defined.
The default **count** function results in counting all subsequent values of the specified row field(s).
Default: Count

AutoSave

Specifies the period of (user) inactivity after which GS-Base will automatically save the open database (if it's modified).
Default: Never

AutoClose

Specifies the period of (user) inactivity after which GS-Base will automatically close the open database. Modified databases are saved before closing.
Default: Never

After Pressing Enter

Specifies whether (and how) to scroll the current table cell after you finish editing and press **Enter**.
Default: No action

Default File Extension

Specifies the default file extension (either *.zip or *.gsb). GS-Base uses that extension when creating and saving new databases/files. Existing files are handled the same way, regardless of their extensions and the changes of that parameter.

Choosing the ***.gsb** extension enables you to register it in the Windows registry and open databases by double-clicking them. The only disadvantage is that you can't view the zip contents using File Manager until you rename it to a *.zip file. You can register the *.gsb file type in Windows and associate it with GS-Base using the **Settings > Register GS-Base *.gsb File Type** command. The corresponding **Settings > Unregister...** command removes all registry entries created during the registration.
Note: To use those commands, you must start GS-Base as an administrator (e.g. right-click the GS-Base desktop shortcut and choose the "Run As Administrator" command).

Choosing the ***.zip** extension enables you to use File Manager to view/edit/browse text tables contained in a given database. In general, it might be helpful only if you're planning to perform some additional text (pre-)processing of these tables with external software, or if you're planning to drag-and-drop text files/tables, e.g. to quickly create some initial collections of tables manually, without GS-Base.
The disadvantage is that if a database contains a large number of inserted objects/images/memos, Windows will considerably slow down when indexing such zip files. Also, you can't open *.zip files in GS-Base by double-clicking such files.

Default: *.zip

Start Folder

Specifies the folder displayed in the **Open File** dialog box when you open it for the first time (after launching GS-Base).
Default: Empty

Relative Shortcut Path

If you insert some link *.lnk files into **Images/Files** fields and later change the location of the target objects that the links are connected to, you can "overwrite" such broken links globally by specifying that path.
Default: Empty

No-Deflate File Types

Specifies the list of file types (stored in **Images/Files** fields) that will be saved in the database zip file without compression. The list should contain file types that already contain compressed data. This can improve loading/saving times without worsening the overall compression factor.
Enter the extensions in the ***.ext1;*.ext2;...*** form.
Default: *.png;*.jpg;*.gif;*.pdf;*.zip;*.ods

Show Lists of Files Attached to Databases

If you manually drag-and-drop any files to a given database zip file, GS-Base will detect such files and show the list of them.
If the option is off, GS-Base will silently load and save back such unused files.
Default: On

Show Progress When Opening/Saving Files

If this option is checked, GS-Base will display a progress dialog box when opening or saving all files. The progress dialog box contains information about the opening/saving process and enables you to cancel it at any time.
Default: On

Auto-Fit Column Widths in Imported Files

If this option is checked, after opening a text, xBase or HTML file, GS-Base will resize (used) columns to fit the widths to the field/column contents.

Selecting this option will slow down opening very large files. To speed up this resizing procedure, increase the **Number of CPUs** value.
Default: Off

Auto-Fit Row Heights in Imported Files

If this option is checked, after opening a text, xBase or HTML file, GS-Base will resize (used) rows to fit the heights to the multiline field/column contents. Checking this option is useful only if some field data may contain new-line characters (CR/LF). If **Auto-fit column widths** is not selected, this option is disabled.
Selecting this option will slow down opening very large files. To speed up this resizing procedure, increase the **Number of CPUs** value.
Default: Off

Windows Opacity

Specifies the opacity of all dialog boxes displayed by GS-Base. Decreasing this value means making the windows more transparent.
Valid range: <10, 100>
Default: 100% (no transparency)

Default Font

Specifies the default font name. The font specified here will be used by database tables, forms, binary edit windows and pivot tables.
Default: Arial

Table/Form Editing Settings

Use Date/Time Picker When Editing Dates

Check this option to use the date-time picker control instead of the plain edit control.
This option applies to text fields that are formatted using one of the standard **Date** styles (not custom style patterns) and that don't use the drop-down list functionality.
Default: On

Edit LongText/Files Fields in New Windows

If this option is **unchecked**, trying to edit (e.g. by pressing Enter, double-clicking or pressing any character/number key) a **Long Text** or **Images/Files** field will open the LongText or Images/Files view pane to enable users to view/edit the contents.

If this option is **checked**, trying to edit a **Long Text** or **Images/Files** field will open an application associated with a given file/contents type. Choosing the "Save" command in that application will automatically update the data in GS-Base.
Depending on how that edit action is initiated, the following additional functionality is supported:

- pressing a digit key opens either the text or the n-th (1...9) object for editing
- pressing a letter key prints either the text or the n-th (a...z) object

Default: Off

Display LongText & Files Field Contents

If this option is unchecked, the table/form fields related to the **Long Text** and **Images/Files** fields will display text containing the number of files, the total size and the last modification date/time. (If the **Images/Files** field contains some shortcuts to external files, only the number of files is displayed as the label.)

If this option is checked, the table/form fields related to the **Long Text** and **Images/Files** fields will display the contained images or file icons with optional labels containing (similarly optional) file names, sizes and the last modification dates/times.

Default: Off

Display Description of Images and Files

Specifies whether images and files (icons) in tables/forms should be displayed along with their general file information, which includes the file name, size and the last modification date/time.
Default: On

Display Images as Thumbnails

Specifies whether images and files (icons) in tables/forms should be displayed as thumbnails of the specified size.

Default: Off

LongText/Files Editing Settings

Strip Off Formatting in Edited LongText Fields

Text entered in **Long Text** fields is saved in the plain UTF-8 *.txt text format rather than in the rich text *.rtf format.
Using plain text may noticeably reduce memory usage if there is a very large number of **Long Text** entries in the database zip file.
Changing this option affects newly entered or edited data only. Older entries remain in the previous format until they are edited.
Default: Off

Wrap Text in Pane

Text displayed in **Long Text** fields is wrapped according to the width of the view pane.
Default: On

Automatically Scroll to Bottom in LongText Fields

By default, after loading LongText fields, the beginning of the text is displayed. If you check this option, the text cursor will be automatically positioned at the bottom of the text.
Default: Off

Display Description of Images and Files

Specifies whether images and files (icons) in **Images/Files** fields should be displayed along with their general file information, which includes the file name, size and the last modification date.
Default: On

Display Images as Thumbnails

Specifies whether images and files (icons) in **Images/Files** fields should be displayed as thumbnails of the specified size.
Default: Off

Spelling Settings

Please see Spell-checking and adding Hunspell dictionaries

GS-Base Help > *Managing the list of recently-used files*

After closing the database window, GS-Base displays the list of recently used databases.

To open a given file, click the corresponding link or use the cursor keys and press Space. Alternatively, drag and drop a given file (or files) into that window.

To change the icon associated with a given file, right-click the current link icon.

To delete a file from the RU list, right-click it and use the **Remove** command from the context menu.

If you want to preserve the displaying order of the RU file list, right-click the list and uncheck the **Cycle RU List Ring** option. Otherwise the recently opened database will be moved to the top.


Recently-used files list context menu

GS-Base Help > *GS-Base scripting: samples*

Sample JScripts scripts

Creating and saving a new database

Adding and removing records, updating calculated fields

Adding, removing and renaming tables and folders

Importing tables from GS-Base databases, text files and Excel workbooks

Merging records from all text and Excel files in a folder

Formatting fields

List of all properties and functions

Window methods

Obtaining the last error details

To make scripting accessible, you need to perform one-time GS-Base scripting registration using the "Settings > Register GS-Base Scripting" command in the admin mode. (Right-click the GS-Base shortcut or file and choose "Run As Administrator", register, close GS-Base). This registers GS-Base in Windows registry as a program that can be scripted not only in GS-Base, but also using external programming tools. (The "Unregister(...)" command removes the added entries.)

You can create your scripts either as global scripts saved in the program settings and available to all databases or you can create scripts stored in a given GS-Base database and available only after you open that database.
To create these scripts use the "File > Application Scripts" and "File > Database Scripts" commands.

The "(...) Scripts" dialog box enables you to organize your scripts in subfolders, test them to locate errors, copy/import/export them etc.

Sample "Scripts" screen.

Notes:

- If a global script is executed when there is no open database, it should start with one of the database Open(...)/New(..) functions
- If a database is already loaded, an executed script, either global or local, automatically refers to that database and using the Open(...) functions with the same database path has no effect.

Creating and saving a new database

```
// a new database with the "table1" table
GSBase.NewDatabase("table1");

// insert another "table1" in the root ("") folder at the bottom;
// as "table1" already exists, the uniqueName will contain a
unique
```

```
// name table1(1)...table1(n) that GS-Base will create and use
instead;
var uniqueName = GSBase.InsertDatabaseItem("", 1, "table1");

var field = GSBase.CreateFieldParams();

// add one Text field to the currently selected table;
//
// adding and importing a table make it 'selected' automatically;
// deleting tables may result in a new selection;
// SetActiveTable() and GetActiveTable() set and retrieve the
selection
//
field.name = "field1";
// T - text field
// N - numeric field
// M - long text / Memo
// O - objects (images, files etc.)
// C - code field (long text with specific syntax highlighting)
field.type = 'T';
GSBase.AppendField(field);

//add one Number field with range validation
//
field.Reset();
field.name = "field2";
field.type = 'N';
field.formula = "=(field2 > 1) * (field2 < 10)";
// 1 - calculation formula/calculated field
// 2 - validation formula
// 3 - conversion formula
// 4 - default value
// 5 - incremented maximum
field.formulaType = 2;
GSBase.AppendField(field);

//add one calculated Number field
//
field.Reset();
field.name = "field3";
field.type = 'N';
field.formula = "=field2 * 10";
field.formulaType = 1;
GSBase.AppendField(field);

//insert one more Text field at the beginning of the record
field.Reset();
field.name = "field4";
field.type = 'T';
GSBase.InsertField(1, field);

//add one Code field that uses the "cpp" syntax highlighting
```

```
field.Reset();
field.name = "field5";
field.type = 'C';
//subtypes same as in the "Field Setup": "cpp", "assembler",
"php"...
field.subtype = "cpp";
GSBase.AppendField(field);

//choose to use the standard zip (zip32) file format for the new
database;
the default value for new files is "true" ("use zip64");
for existing database files the default value is the one that was
used previously;

//GSBase.zip64 = true;
GSBase.zip64 = false;

//save a new database (with one table and no records so far);
GSBase.SaveDatabaseAs("e:\\test_dbase.gsb");
GSBase.Close();
```

Adding and removing records, updating calculated fields

```
var password = "fhE4!lko"
GSBase.OpenDatabase("e:\\test_dbase.gsb", password);

//all editing actions always refer to the currently selected
table;
//once a table is selected (and the database is saved), it remains
selected till you change this;
//
if (GSBase.GetActiveTable() != "table1")
        GSBase.SetActiveTable("table1");

//as the table contains the calculated "field3", for performance
reasons turn off
//recalculation of each row which would otherwise occur after each
of the 10,000 modifications below;
//to restore the default automatic updating use "automatic" or re-
open the database;
GSBase.updateMode = "manual";

//NOTE: InsertText() and InsertNumber() do exactly what plain
entering data does
//which means they set up Undo information and refreshes the
```

```
screen which makes them slow. If you need
//to fill millions of fields instantly, use the
Insert(...)Series() functions family instead.

//NOTE: for best performance, when inserting a large series of
data, always fill the table fields
//in the "top to bottom" (and "left to right") order. The "bottom
to top" order may
//be considerably slower.

//insert the following date string in the 1st record field in 100
records;
var today = "2021-01-15";
for (i = 1; i <= 100; ++i)
{
        GSBase.InsertText(i, 1, today);
}

//insert some numbers in the 3rd record field in the first 100
records
for (i = 1; i <= 100; ++i)
{
        GSBase.InsertNumber(i, 3, 2.0 + i%8);
}

//update all calculated fields in this table using 4 processor
cores
GSBase.UpdateTable(4);

//get the sum of the 4th field for the entire current record set
var counter = GSBase.GetRecordSetCount();
var sum = 0;
for (i = 1; i <= counter; ++i)
        sum = sum + GSBase.GetNumber(i, 4);

//clear the 3rd field in records 11 to 21
GSBase.ClearRange(11, 3, 21, 3);

//update all "field3" calculated fields as the "manual" update
mode was set earlier
GSBase.UpdateTable(4);

//remove record 1
GSBase.RemoveRecords(1, 1);
//remove records 2 to 3
GSBase.RemoveRecords(2, 3);
//remove record 11
GSBase.RemoveRecords(11, 11);

// save using the current file format and file format options
GSBase.Save();
```

```
GSBase.Close();
```

Adding, removing and renaming tables and folders

```
GSBase.OpenDatabase("e:\\test_dbase.gsb", "");

//notes:
//insert a new folder "folder1\" in the root folder at the bottom;
//if "folder1" already exists at that level, the uniqueName will
contain a unique
//name folder1(1)...folder1(n) that GS-Base will create and use
instead;
var uniqueName = GSBase.InsertDatabaseItem("", 1, "folder1\\");

//insert folder2 at the top
GSBase.InsertDatabaseItem("", 0, "folder2\\");

GSBase.InsertDatabaseItem("\\folder1\\", 1, "folder1\\");
GSBase.InsertDatabaseItem("folder1", 1, "folder1\\");

//rename the "text_abc(2)" table in the "\folder2" folder to
"test_abc_b";
GSBase.RenameDatabaseItem("\\folder2\\test_abc(2)", "test_abc_b");
//delete the entire "folder2" folder
GSBase.DeleteDatabaseItem("\\folder2\\");

//delete the "\folder1\test_abc_b" table
GSBase.DeleteDatabaseItem("\\folder1\\test_abc_b");

GSBase.Save();
GSBase.Close();
```

Importing tables from GS-Base databases, text files and Excel workbooks

```
//import the "product" table from the sample.zip database and
insert it in the root folder
GSBase.ImportTable("e:\\sample.zip", "products", "", "\\");
```

```
//import to the "folder1" subfolder
GSBase.ImportTable("c:\\sample2.zip", "products2",
"pass4563word!", "folder1");

//import the first table from file_a.xlsx and insert it in the
root folder
//first row in file_a.xlsx contains field names
GSBase.ImportExcelTable("e:\\excel_data\\file_a.xlsx", "", 1, "");
//import the first table from file_c.xlsx and insert it in the
root folder
//first row in file_a.xlsx doesn't contain field names
GSBase.ImportExcelTable("e:\\excel_data\\file_c.xlsx", "", 0, "");
//import the first table from file_a.xlsx and insert it in the
root folder
//first row in file_a.xlsx doesn't contain field names
GSBase.ImportExcelTable("e:\\excel_data\\file_i.xlsx", "", 0, "");

//create text file parameters
var textParams = GSBase.CreateTextParams();
//use "|" as the field separator ("," is the default one)
textParams.separator = "|";

//import a new table from the "text_abc.txt" text file and place
it in the "folder2" folder
GSBase.ImportTextTable("c:\\test_abc.txt", "", textParams,
"\\folder2\\");
//import it again (the name of the imported table will be modified
to "text_abc(1)")
GSBase.ImportTextTable("c:\\test_abc.txt", "", textParams,
"\\folder2\\");
//import it again (the name of the imported table will be modified
to "text_abc(2)")
GSBase.ImportTextTable("c:\\test_abc.txt", "some_name",
textParams, "\\folder2");

GSBase.Save();
GSBase.Close();
```

Merging records from all text and Excel files in a folder

```


GSBase.OpenDatabase("e:\\test_dbase.gsb", "");

var mergeParams = GSBase.CreateMergeParams();
// merge record e.g. from report2000.txt, report2001.txt, ...,
```

```
report2023.txt
mergeParams.path = "e:\\gsb19_test\\report20??.txt";
// don't perform field names matching, add fields "as is"
// note: field types must match, text fields can't be copied to
numeric fields
mergeParams.matchFieldNames = false;

var textParams = GSBase.CreateTextParams();
textParams.separator = "|";
textParams.encoding = "windows";

GSBase.MergeRecordsFromTextFile(mergeParams, textParams);

// if there are calculated fields, update all records using 4
processor cores
GSBase.UpdateTable(4);

// you can also only perform more complex merging using the
"mergeType" property:
// mergeParams.mergeType=0 - merge records unconditionally
(default value)
// mergeParams.mergeType=1 - merge records only with new values of
the mergeParams.slaveIndex field from merged files
// mergeParams.mergeType=2 - update records in the main table
where the specified fields in the main/master table and in the
merge tables are the same, mergeParams.masterIndex =
mergeParams.slaveIndex
// mergeParams.mergeType=3 - delete records from the main table
where the specified fields in the main/master table and in the
merge tables are the same, mergeParams.masterIndex =
mergeParams.slaveIndex

// For Excel and other file format use:
// MergeRecords(mergeParams)
// MergeRecordsFromTextFile(mergeParams, textParams)
// MergeRecordsFromExcelFile(mergeParams, namesInFirstRow)
// MergeRecordsFromXBaseFile(mergeParams, xBaseParams)
// MergeRecordsFromMySQLFile(mergeParams)

Record merging functions

GSBase.Save();
GSBase.Close();
```

Formatting fields

```
GSBase.OpenDatabase("e:\\test_dbase.gsb", 0);

//select the "table1" table in the root folder
GSBase.SetActiveTable("\\table1");

//create formatting settings
var format = GSBase.CreateFormatParams();

//set the currency format: variable/automatic number of decimals
and the exponent value
//parameters:
//1. decimals: 0 - 14 | "auto"
//2. currency position: "$1.1" | "$ 1.1" | "1.1$" | "1.1 $"
//3. currency symbol: "$", "GBP" etc.
//4. true - use curly braces for negative values
//5. true - use red color for negative values
//
format.SetCurrencyFormat("2", "$1.1", "gbp", true, true);

// other style examples:
//
//set the scientific style: 5 decimal digits and the fixed "07"
exponent
//
//----- format.SetScientificFormat("5", "07");
//
//set the scientific style: variable/automatic number of decimals
and the exponent value
//
//----- format.SetScientificFormat("auto", "auto");
//
//
//set the accounting style
//parameters:
//1. decimals: 0 - 14 | "auto"
//2. currency symbol: "$", "GBP" etc.
//
//----- format.SetAccountingFormat("2", "gbp");
//
//
//set the fractional format
//parameters:
//1. if the 2nd parameter is "false", (1) is a denominator value
2, 3, ..., n;
//   if the 2nd parameter is "true", (1) is a fixed number of
denominator digits 1...14
//2. ...
//
//----- format.SetFractionFormat(2, true);
//
```

```
//
//set the general number format
//parameters:
//1. decimals: 0 - 14 | "auto"
//2. leading zeroes: 0 - 14
//3. true - use curly braces for negative values
//4. true - use red color for negative values
//5. true - use the thousand separator
//
//----- format.SetGeneralNumberFormat("auto", 5, false, false,
true);
//

GSBase.SetFieldFormat(3, format);

//set the date format
//parameters:
//1. a date pattern: same as in the "Format > Style" window
//2. 1 - always switch to the current Windows system day/month
order ("m/d..." | "d/m...")
format.SetDateFormat("m/d/yyyy", 1);
GSBase.SetFieldFormat(1, format);

//clear the previously set information
format.Reset();

format.fontSize = 14;
format.boldFont = true;
GSBase.SetFieldFormat(3, format);

GSBase.Save();
GSBase.Close();
```

Setting column/field widths and row heights

```


GSBase.OpenDatabase("e:\\test_dbase.gsb", 0);

if (GSBase.GetActiveTable() != "table1")
        GSBase.SetActiveTable("table1");

//get the 1st column/field width in screen pixels
var width = GSBase.GetColumnWidth(1);

//set a new width
width += 50;
```

```
GSBase.SetColumnWidth(1, width);

//fit the 3rd column/field width to the data in that column/field
GSBase.FitColumnWidth(3, 3);

//get the 9th row/record height in screen pixels: typically it
should be 25px
var height = GSBase.GetRowHeight(9);

//set a new height
height += 20;
GSBase.SetRowHeight(9, height);

//check the "auto-height" state of the 9th row
var autoHeight = GSBase.GetAutoRowHeight(9);

//restore on the "auto-height" state for the 9th row
GSBase.SetAutoRowHeight(9, 9, true);

//should be true now
autoHeight = GSBase.GetAutoRowHeight(9);

//it should be 25px again
height = GSBase.GetRowHeight(9);

GSBase.Save();
GSBase.Close();
```

Browsing the folders/tables tree

```
GSBase.OpenDatabase("e:\\test_dbase.gsb", "");

//1st method

//get the total number of tables and folders in the main/root
folder (including nested folders)
var counter = GSBase.GetDatabaseItemCount();

//iterate through all table/folders
for (i = 1; i <= counter; ++i)
{
        var path = GSBase.GetDatabaseItem(i);
        if (path.length && path.charAt(path.length - 1) == '\\')
        {
                //folder
```

```
                var ret = GSBase.MessageBox("folder - " + path,
"ok", 1, "information");
        }
        else
        {
                //table
                var ret = GSBase.MessageBox("table - " + path,
"ok", 1, "information");
        }

        var ret = GSBase.MessageBox(path, "ok", 1, "information");
}

//2nd method

//get the first "child" element in the specified folder, e.g. the
main/"root" folder
//var path = GSBase.GetFirstDatabaseItem("\\folder2(1)\\");
var path = GSBase.GetFirstDatabaseItem("\\");

//iterate through direct "child" elements of the specified folder
(not expanding nested folders)
while (path.length)
{
        if (path.charAt(path.length - 1) == '\\')
        {
                //folder
                var ret = GSBase.MessageBox("folder - " + path,
"ok", 1, "information");
        }
        else
        {
                //table
                var ret = GSBase.MessageBox("table - " + path,
"ok", 1, "information");
        }
        path = GSBase.GetNextDatabaseItem(path);
}

GSBase.Close();
```

Searching and sorting

```
GSBase.OpenDatabase("e:\\test_dbase.gsb", "");
```

```
//select the "table1" table in the root folder
GSBase.SetActiveTable("\\products");

var field = GSBase.CreateFieldParams();

//find the "ProductName" and "UnitPrice" fields;
//filter "ProductName" and sort "UnitPrice"

//reset the previous sorting indices - resetting should be used
before calling "put_sortIndex()"
GSBase.ResetSorting();

var iname = 0;
var iprice = 0;

for (i = 1; i <= GSBase.GetFieldCount() && (!iname || !iprice);
++i)
{
        GSBase.GetField(i, field);
        if (!iname && field.name == "ProductName")
        {
                //set the "\Ai" RegEx filter for "ProductName"
(=search for names starting with "I");
                //searching is performed automatically if a call
to the "SetField" changes the "filter" value;
                //note: in this version the filter type is always
"RegEx"
                field.filter = "\\Ai";

                GSBase.SetField(iname = i, field);
        }
        if (!iprice && field.name == "UnitPrice")
        {
                //set the 1 as the sorting index for "UnitPrice";
                //sorting is performed automatically after a call
to "SetField" if the "sorting" index is modified;
                //to create a compound sorting index, use
subsequent numbers (2, 3...) for further fields;
                //note: using an index not in that strictly
incremented manner causes an error;

                field.sortIndex = 1;

                // 'A' - ascending order, 'D' - descending order
                field.sortOrder = 'A';

                GSBase.SetField(iprice = i, field);
        }
}

GSBase.Save();
```

```
GSBase.Close();
```

Predefined searching

```
GSBase.OpenDatabase("e:\\test_dbase.gsb", "");

//select the "table1" table in the root folder
GSBase.SetActiveTable("\\products");

//find duplicates in the 5th field
GSBase.FindDuplicates(5, 5, 1, false, false);

//check the results
var counter1 = GSBase.GetRecordTotalCount();
var counter2 = GSBase.GetRecordSetCount();

//find records with the flag "1"
GSBase.FindFlagged(1);

// ...

//find the records not included in the current record set
GSBase.FindComplement();

// ...

//clear all filters and display all records
GSBase.FindAll();

// ...

GSBase.Close();
```

File password protection

```
//set a password
GSBase.OpenDatabase("e:\\test_dbase.gsb", "");

GSBase.SetFilePassword(true, "blowfish", "", "rocc4545");
```

```
GSBase.Save();
GSBase.Close();

//open a password-protected database
GSBase.OpenDatabase("e:\\test_dbase.gsb", "rocc4545");
//...
//GSBase.Save();
GSBase.Close();

//remove password protection
GSBase.OpenDatabase("e:\\test_dbase.gsb", "rocc4545");

GSBase.SetFilePassword(false, "blowfish", "rocc4545", "");

GSBase.Save();
GSBase.Close();
```

Enabling sharing databases and text, Excel and other files

```
var fpath = GSBase.ReleaseFile()
GSBase.MessageBox("File closed: " + fpath, "ok", 1,
"information");

//You can keep on viewing/working with this file in GS-Base as
usual (as it's in RAM).
//It can be edited in other programs.


//At any moment you can execute the following script, which will
check for updates:

if (GSBase.AttachFile(fpath) == 1)
{
    // AttachFile returned 1, so the file was modified
    // after ReleaseFile(), you can ask whether reload
    // it now or later
    GSBase.Reload();
}
else
{
    // no changes detected, so leave the file accessible
    // for any other programs.
    GSBase.ReleaseFile();
```

```
}
```

Opening and saving text files

```
// 1. Exporting the current record set to a text file
// ------------------------------------------------

GSBase.OpenDatabase("e:\\test_dbase.gsb", "");

//select the "table1" table in the root folder
GSBase.SetActiveTable("\\products");

//create text file parameters
var textParams = GSBase.CreateTextParams();

//use ";" as the field separator; the default value is ","
textParams.separator = ";";

//use of separators can be turned off; the default value is true
//textParams.useSeparator = false;

//use "'" as the quoting symbol; the default value is """
textParams.quotingSymbol = "'";

//use of quoting symbols can be turned off; the default value is
true
//textParams.useQuoting = false;

//save field names in the first row; the default value is true
textParams.fieldNames = true;

//change the text encoding: "utf8" | "windows" | "dos"; the
default value is "utf8"
textParams.encoding = "utf8";

//export the current table to a text file;
//"dbase" remains the originally opened database and can edited
further as usual
GSBase.SaveRecordSetAsText("e:\\test_b.txt", textParams);


// 2. Opening and saving a text file
// ---------------------------------

//re-use the above "txt" settings and add new ones
```

```
//if a column contains textual representations of numbers, try
converting them to a number field
textParams.parseNumbers = true;

//if a column contains textual representations of dates in various
formats, convert these strings
//to the generic "DT" W3 text representation of dates in GS-Base
(please see the "data types" help topic for details).
textParams.parseDates = true;

GSBase.OpenTextFile("e:\\test_b.txt", textParams);

var fcounter = GSBase.GetFieldCount();

var field = GSBase.CreateFieldParams();

GSBase.GetField(1, field);
//
// ...perform any editing, field changes etc.
//

//save the edited text file
GSBase.Save();
GSBase.Close();

// 3. Opening a text file and saving it as a database
// ------------------------------------------------

//re-use the above "txt"
textParams.parseNumbers = false;

GSBase.OpenTextFile("e:\\test_b.txt", txt);

//SaveDatabaseAs changes "textFile" to a database with the path
given below; the text file is closed
GSBase.SaveDatabaseAs("e:\\sample_b.gsb");

//
// ...perform any editing, field actions etc. with
"e:\\sample_b.gsb"
//

GSBase.Close();
```

Error handling

```
GSBase.OpenDatabase("e:\\test_dbase.gsb", "");

//
// ...
//

// if a COM function returns an error other than E_OUTOFMEMORY,
// additional error information can be obtained via the
"lastError" property;
var code = GSBase.lastError;

// 21   // Out of memory while creating/editing worksheet data
// 22   // A database must contain at least one table
// 37   // Invalid password.
// 53   // Not allowed field type change - e.g. conversion form
"Files/Images" to "Number"
// 61   // Can't open the specified file
// 62   // Can't open or create the specified file
// 63   // Error while closing the specified file
// 64   // Error while repositioning a file pointer
// 65   // Error while reading from a file
// 66   // Error while writing to a file
// 67   // Error while deleting a file
// 68   // Error while checking the file size/info
// 69   // Error while allocating a file read/write buffer
// 70   // Can't open the specified file. File in use.
// 71   // Error while decrypting a file.
// 72   // Error while encrypting a file.
// 73   // Error while reading binary fields."
// 79   // Can't find the manifest file or its content is
incorrect
// 80   // The file format requires a newer GS-Base version.
// 119  // Too many zip streams in a standard zip (zip32) file
// 120  // Inconsistent zip stream state
// 121  // Corrupted zip stream data
// 122  // Out of memory while processing a zip stream
// 123  // Unexpected end of zip stream
// 125  // Unknown zlib error
// 131  // Some data in the file can't be converted to numeric
field values.

GSBase.Close();
```

File/database opening functions

Method / Property	*Description*
`NewDatabase(tableName)`	
`OpenDatabase(path, password)`	If the database is not encrypted, password should be ""/null.
`OpenTextFile(path, showDialogBox, textParams)`	showDialogBox = 1 to show the standard "Open Text File" dialog box. textParams - an object created with GSBase.CreateTextParams().
`OpenExcelFile(path, showDialogBox, mergeType, options)`	showDialogBox = 1 to show the standard "Open Excel File" dialog box. mergeType - represents the 0...3 "merge" radio buttons from the "Open Excel File" dialog box. options - a combination of 1, 2, 4 (1 - field names in the 1st worksheet, 2 - field names in every worksheet, 4 - restore GS-Base table/folder tree paths).
`OpenHtmlFile(path, fnames)`	fnames = 1 if there are field names in the 1st row of the html table (note: the table must have the "id" attribute set to "gsbase").
`OpenXBaseFile(path, showDialogBox, xBaseParams)`	showDialogBox = 1 to show the standard "Open xBase File" dialog box. xBaseParams - an object created with GSBase.CreateXBaseParams().
`OpenMySQLFile(path)`	
`Close()`	Closing is also automatic when subsequent "open" methods are used.
`Reload()`	Reloads the opened file using its originally used file format settings.

params = GSBase.CreateTextParams() - settings corresponding to the "Open Text File" dialog box options

Method / Property	*Description*
`params.useSeparator`	0, 1
`params.separator`	any character
`params.fixedFieldWidths`	for example, "10,20,30,15"
`params.encoding`	"utf8", "utf16", "windows"
`params.useQuoting`	0, 1
`params.quotingSymbol`	any character
`params.fieldNames`	0, 1 (field names in the first row)
`params.parseNumbers`	0, 1
`params.parseDates`	0, 1
`params.saveLongTextAsZip`	0, 1
`params.saveObjectsAsZip`	0, 1
`params.autoFitColumns`	0, 1

params = GSBase.CreateXBaseParams() - settings corresponding to the "Open xBase File" dialog box options

Method / Property	*Description*
`params.format`	"dbaseIII", "dbaseIV", "foxpro", "clipper"
`params.encoding`	"windows", "dos"
`params.GetFieldCount()`	
`params.SetField(index, name, type, length, decimals)`	type: "C", "N", "F", "L", "D", "M"

Method / Property	*Description*
`params.AddField(name, type, length, decimals)`	
`params.params.InsertField(index, name, type, length, decimals)`	
`params.GetFieldName(index)`	index: 1...number of fields
`params.GetFieldType(index)`	
`params.GetFieldLength(index)`	
`params.GetFieldDecimals(index)`	0, 1
`params.DeleteField(index)`	

Saving recordsets

Method / Property	*Description*
`SaveRecordSetAs(path, password)`	
`SaveRecordSetAsText(path, textParams)`	If the file is not to be encrypted, password should be ""/null.
`SaveRecordSetAsExcel(path, split, fnames, saveZip)`	
`SaveRecordSetAsXBase(path, format, xBaseParams)`	
`SaveRecordSetAsMySQL(path)`	
`SaveRecordSetAsPDF(path)`	

Method / Property	*Description*
`zip64`	0, 1 - specify whether ZIP64 or the older standard Windows ZIP32 should be used for *.gsb and *.xlsx files.

Note: Unlike "SaveDatabaseAs" methods, recordset saving methods do not change the current database path or state.

Saving databases

Method / Property	*Description*
`Save()`	Saves the current file using the current file format and its current file format settings
`SaveTextFile(textParams)`	Enables you to modify settings when saving a previously opened text file.
`SaveExcelFile(split, fnames, saveZip)`	
`SaveXBaseFile(xBaseParams)`	
`SaveMySQLFile()`	

Saving databases as new files

Method / Property	*Description*
`SaveDatabaseAs(path)`	
`SaveDatabaseAsExcel(path, split, fnames, saveZip)`	
`SaveDatabaseAsMySQL(path)`	

Method / Property	*Description*
`zip64`	0, 1 - specify whether ZIP64 or the older standard Windows ZIP32 should be used for *.gsb and *.xlsx files.

Note: The above methods change the current database path, saving a new copy of a given database.

Importing tables from databases, text and Excel files

Method / Property	*Description*
`ImportTable(filePath, tablePath, password, folder)`	If tablePath="", the first found table from "filePath" is imported The target "folder" must exist or the parameter must be empty. If folder="", the imported table is inserted in the main/root folder
`ImportTextTable(textFilePath, tableName, textParams, folder)`	If tableName="", the inserted table name will be the same as the text file name The target "folder" must exist or the parameter must be empty. If folder="", the imported table is inserted in the main/root folder
`ImportExcelTable(excelFilePath, tableName, namesInFirstRow, folder)`	

Performing JOIN operations for tables in the same database file

Method / Property	*Description*
`MergeTable(masterField, tablePath, slaveField, allowDuplicates)`	masterField - 1-based index of the field in the current table tablePath - full path of the joined table slaveField - 1-based index of the related field in the joined table allowDuplicates - 0, 1: specifies whether duplicated values in the joined table should result in adding multiple joined records in the result table

Merging/adding records from other files

Method / Property	*Description*
`MergeRecords(mergeParams)`	Merge records from another *.gsb database mergeParams - an object created with GSBase.CreateMergeParams().
`MergeRecordsFromTextFile(mergeParams, textParams)`	textParams - an object created with GSBase.CreateTextParams()
`MergeRecordsFromExcelFile(mergeParams, namesInFirstRow)`	
`MergeRecordsFromXBaseFile(mergeParams, xBaseParams)`	xBaseParams - an object created with GSBase.CreateXBaseParams().
`MergeRecordsFromMySQLFile(mergeParams)`	

params = GSBase.CreateMergeParams() - settings corresponding to the record merging options

Method / Property	*Description*
`params.path`	full file path; the file name contain wildcard characters to enable mass merge of files from a given folder, for example, "e:\\folder*.csv", "f:\\folder*20??.gsb", "c:\\report*.xlsx"
`params.table`	full table path for *.gsb, *.xlsx and *.sql files; if it's empty the first table in the specified file is used.
`params.masterField`	1-based index of the field in the current table; used only if the mergeType <> 0
`params.slaveField`	1-based index of the related field in the merged table; used only if the mergeType <> 0
`params.mergeType`	0 - add all records, 1 - add records where the "slaveField" values don't exist in the main table 2 - update records in the main table where the "slaveField" values exist in the main table 3 - delete records from the main table where the "slaveField" values exist in the main table
`params.matchFieldNames`	0, 1; used only if the mergeType=0 and causes adding records unconditionally, without matching field names in both tables
`params.ignoreEmpty`	0, 1; used only if the mergeType=2 and causes not using empty fields
`params.enableUndo`	0, 1; specifies whether the UI Undo command should be enabled for

Method / Property	Description
	merging; it's 0 by default as the undo information can multiply the memory usage for these actions

Switching the active table

Method / Property	Description
`SetActiveTable(path)`	path - full table path.
`GetActiveTable()`	

Record counters

Method / Property	Description
`GetRecordTotalCount()`	
`GetRecordSetCount()`	
`GetFieldCount()`	

Searching

Method / Property	Description
`FindDuplicates(fieldFrom, fieldTo, searchAll, type, sortResults)`	fieldFrom and fieldTo are 1-based field indices. Fields in this range are treated as the search key. searchAll: 0, 1 - if it's 1, GS-Base will search for duplicates in the current filtered record set; otherwise the entire table is searched.

Method / Property	*Description*
	type=0 - show all duplicates, type=1 - show first records from each group of duplicates, type=2 - show the last record from each group of duplicates, type=3 - all duplicates except the first from each group, type=4 - all duplicates except the last one from each group sortResults: 0,1 - sort the obtained records using the above key.
`FindUnique(fieldFrom, fieldTo, searchAll, type, sortResults, useCounters, counterIndex)`	fieldFrom and fieldTo are 1-based field indices. Fields in this range are treated as the search key. searchAll: 0, 1 - if it's 1, GS-Base will search for duplicates in the current filtered record set; otherwise the entire table is searched. type=0 - if there are duplicates, show first records from each group of duplicates, type=1 - if there are duplicates, show the last record from each group of duplicates, useCounters: 0, 1 - use 1 to specify a field where the number of occurrences will be saved, counterIndex: 1-based index of a numeric field, where the number of occurrences will be saved, sortResults: 0,1 - sort the obtained records using the above key.
`FindQuartile(field, searchAll, statAll, type)`	The "type" parameter can be a mix of the following values: 1: 1st, 2: 2nd,

Method / Property	*Description*
	4: 3rd, 8: 4th, 16: below mean, 32: above mean statAll: 0, 1 - if it's 1, the statistics will be calculated for the entire table, otherwise for the current recordset.
`FindRandom(n)`	n randomly selected records from the current recordset
`FindFlagged(flag)`	flag: a flag index from 1 to the number of created flags
`FindComplement()`	Shows all records not included in the current recordset.
`FindAll()`	Shows all records.

Note: regular field filter are set using the SetField() function.

Adding, modifying and removing fields

Method / Property	*Description*
`AppendField(fieldParams)`	fieldParams - an object created with GSBase.CreateFieldParams()
`InsertField(field, fieldParams)`	
`AppendField(fieldParams)`	
`RemoveField(field)`	
`GetField(field, fieldParams)`	

Method / Property	*Description*
`SetField(field, fieldParams)`	SetField() triggers filtering and/or sorting if the respective data was set in the fieldParams
`ConvertField(field, type)`	
`ResetSorting()`	Clears any currently defined single or multiple sort key

params = GSBase.CreateFieldParams() - settings corresponding to the record field definition

Method / Property	*Description*
`params.name`	
`params.type`	"T" - textual "N" - numeric "M" - LongText "C" - Code "O" - Images/Files
`params.subType`	used for the "C" field type: "cpp", "cpp/rc", "cpp/idl(odl)", "c#", "assembler", "java", "jscript", "vbscript/vb.net", "html", "xml", "flash", "php", "python", "perl", "powershell",

Method / Property	*Description*
	"sql". Choosing the subtype causes applying the correct language syntax and keyword coloring.
`params.hlink`	0, 1
`params.sortIndex`	1 for single key sorting or 1-based index for multi-key sorting
`params.sortOrder`	"A" or "D"
`params.formula`	e.g. "=some_numeric_field_name + 10"
`params.formulaType`	1 - calculation formula/calculated field 2 - validation formula 3 - conversion formula 4 - default value 5 - incremented maximum
`params.filter`	sets the current filter expression; if it changes, the SetField() method will trigger searching
`params.filterType`	1 - regex 2 - plain text pattern 3 - equal 4 - not equal 5 - greater than 6 - less than 7 - between 8 - and 9 - or 10 - is empty 11 - is similar 12 - flag index
`params.matchCase`	0, 1

Method / Property	*Description*
`params.matchWords`	0, 1; match whole words for full text plain searching
`params.Reset()`	Clear all set options and flags

Browsing the database folder and tables tree structure

Method / Property	*Description*
`GetDatabaseItemCount()`	The total number of tables and folders in a database file.
`GetDatabaseItem(index)`	Returns the full table or folder path.
`GetFirstDatabaseItem(folder)`	
`GetNextDatabaseItem(prevName)`	
`RenameDatabaseItem(path, name)`	
`DeleteDatabaseItem(path)`	

Editing records

NOTE: InsertText() and InsertNumber() do exactly what plain entering data does which means they set up Undo information and refreshes the screen which makes them slow. If you need to fill millions of fields instantly, use the Insert(...)Series() functions family instead.

Method / Property	*Description*
`RemoveRecords(recordFrom, recordTo)`	

Method / Property	*Description*
`InsertRecords(recordFrom, recordTo)`	
`InsertText(record, field, text)`	
`GetText(record, field)`	
`InsertNumber(record, field, number)`	
`GetNumber(record, field)`	
`Clear(record, field)`	
`ClearRange(record1, field1, record2, field2)`	

Note: all record numbers are relative to the current, filtered and sorted (or not) record set. They are equal to physical cardinal record numbers in a table only if they are not active searches or sort keys.

Inserting series

Method / Property	*Description*
`InsertSeries(field, recordFrom, recordTo, copy)`	copy=0 - numeric or date/time sequence copy=1 - copying the top item within the selected range See: Inserting series
`InsertRecurrenceSeries(field, recordFrom, recordTo, formula)`	See: Inserting series

Method / Property	*Description*
`InsertCustomSeries(field, recordFrom, recordTo, series)`	See: Inserting series
`InsertRandomSeries(field, recordFrom, recordTo, distribution, param1, param2)`	See: Inserting series

Note: all record numbers are relative to the current, filtered and sorted (or not) record set. They are equal to physical cardinal record numbers in a table only if they are not active searches or sort keys.

Changing column widths and row heights

Method / Property	*Description*
`GetColumnWidth(field)`	
`SetColumnWidth(field, width)`	
`FitColumnWidth(fieldFrom, fieldTo)`	
`GetRowHeight(record)`	
`SetRowHeight(record, height)`	
`GetNumber(record, field)`	
`GetAutoRowHeight(record)`	
`SetAutoRowHeight(recordFrom, recordTo, val)`	val: 0, 1

Note: all record numbers are relative to the current, filtered and sorted (or not) record set. They are equal to physical cardinal record numbers in a table only if they are not active searches or sort keys.

Field format settings

Method / Property	*Description*
`CreateFormatParams()`	
`GetFieldFormat(field, formatParams)`	field - 1-based field index, formatParams - an object created with GSBase.CreateFormatParams()
`SetFieldFormat(field, formatParams)`	

params = GSBase.CreateFormatParams() - format/style settings

Method / Property	*Description*
`params.SetGeneralNumberFormat(decimals, zeroes, brackets, inRed, separators, scaling)`	1. decimals: 0 - 14 | "auto" 2. leading zeroes: 0 - 14 3. true - use curly braces for negative values 4. true - use red color for negative values 5. true - use the thousand separator For example: params.SetGeneralNumberFormat("auto", 5, false, false, true);
`params.SetCurrencyFormat(decimals, position, symbol, brackets, inRed, scaling)`	1. decimals: 0 - 14 | "auto" 2. currency position: "$1.1" \| "$ 1.1" | "1.1$" | "1.1 $" 3. currency symbol: "$", "GBP" etc. 4. true - use curly braces for negative values 5. true - use red color for negative

Method / Property	*Description*
	values For example: params.SetCurrencyFormat("2", "$1.1", "gbp", true, true);
`params.SetAccountingFormat (decimals, symbol, scaling)`	1. decimals: 0 - 14 | "auto" 2. currency symbol: "$", "GBP" etc. For example: params.SetAccountingFormat("2", "gbp");
`params.SetDateFormat(patter n, systemOrder)`	1. a date pattern: same as in the "Format > Style" window 2. 1 - always switch to the current Windows system day/month order ("m/d..." | "d/m...") For example: params.SetDateFormat("m/d/yyyy", 1)
`params.SetTimeFormat(patter n)`	
`params.SetDateTimeFormat(da tePattern, timePattern, syst emOrder, timeFirst)`	
`params.SetPercentFormat(dec imals, scaling)`	decimals - the number of decimal places; this can be "auto" or any number from 0 to 14. Default value: "auto" scaling - the display factor as a power of 1000; this can be any number from 0 to 5. Default value: 0
`params.SetFractionFormat(de nominator, digits)`	1. if the 2nd parameter is "false", (1) is a denominator value 2, 3, ..., n if the 2nd parameter is "true", (1) is a fixed number of denominator digits 1...14

Method / Property	*Description*
	2. ... For example: params.SetFractionFormat(2, true);
`params.SetScientificFormat (decimals, exponent)`	For example: 5 decimal digits and the fixed "07" exponent params.SetScientificFormat("5", "07") variable/automatic number of decimals and the exponent value params.SetScientificFormat("auto", "auto");
`params.fontName`	
`params.fontSize`	0, 1
`params.boldFont`	
`params.italicFont`	
`params.underlineFont`	
`params.strikeoutFont`	
`params.fontColor`	
`params.horzAlignment`	
`params.vertAlignment`	
`params.wrapText`	
`params.shrinkText`	
`params.Reset`	

Setting record flags

Method / Property	*Description*
`GetRecordFlag(record)`	
`SetRecordFlag(recordFrom, recordTo, flag)`	flag index - 1 to the number of defined flags

Setting database passwords for GS-Base *.gsb/*.zip files

Method / Property	*Description*
`SetFilePassword(enable, cryptMethod, oldPassword, newPassword)`	enable: 0, 1 cryptMethod: "blowfish" oldPassword: empty if setting a new password newPassword: empty if removing password protection

Note: The industry-level encryption includes the entire database contents and all fields (textual, numeric, LongText, Code, Images/Objects). Information included in JScript and VBscript scripts stored either as GS-Base global settings or scripts added to files using the "File > Database Scripts" command is not encrypted.

File information and access

Method / Property	*Description*
`GetFileInfo(path, type)`	type=1 - returns the file size, type=2 - return the last modification date in the format YYYY-MM-DD
`ReleaseFile()`	Closes, detaches the current database file and return its file path. This enables updating records by another processes/users. Using "Save()" before attaching the file back

Method / Property	*Description*
	causes displaying the "File Save As" dialog box.
`AttachFile(path)`	Attach a given file to the current database loaded in RAM and returns 0 if the file has been modified after the most recent use of the Release() function.

Updating all calculation formulas

Method / Property	*Description*
`UpdateTable(procCores)`	Update calculation formulas in all records. By default, record editing actions cause updating the modified records only.
`updateMode`	Can be: "default", "automatic", "manual". Changing this value from "automatic" to "manual" can be helpful if there are many "block" editing actions (field editing, conversion, clearing ranges) and only one final UpdateTable() will be sufficient instead of multiple time-consuming automatic updates.

Input/output methods

Method / Property	*Description*
`MessageBox(message, buttons, button, icon)`	The "button" parameter must be one of the following strings:

Method / Property	*Description*
	• ok • ok-cancel • retry-cancel • yes-no • yes-no-cancel • abort-retry-ignore The "button" parameter is an index 1-3 of the default message box button. The "icon" parameter must be one of the following strings: • exclamation • warning • information • question • error Returns a text string representing the name of the clicked/pressed button: • abort • cancel • continue • ignore • no • ok • retry • tryagain • yes
`InputBox(title, isPassword, initValue)`	

Method / Property	Description
`Sleep(unsigned long time)`	

Window methods

Method / Property	Description
`MaximizeAppWindow()`	
`MinimizeAppWindow()`	
`RestoreAppWindow()`	

Obtaining the last error details

Method / Property	Description
`lastError`	

Related Topics

Interfaces

Searching and sorting
Predefined searching
File password protection
Opening and saving text files
Error handling

The following examples were created in MS Visual Studio C++. (The "community" version can be downloaded from the MS website at no cost.) The examples use "smart pointers" and function wrappers with declarations and definitions automatically generated as header and source files by MS VS C++ when you use the #import directive. The "main()" definitions are skipped. These function wrappers use exceptions as a method of handling errors. If you don't want to use exceptions, please see those headers files for how to use the generic COM system function calls instead.

In higher level languages supporting COM Automation including scripting languages like JScript, VBScript the rules for creating the "GS-Base.Application" object and the other available objects and their interfaces remain in general the same except that you can access object "properties" (see the **GS-Base COM interfaces** help topic) directly, without the "get_..." and "put_..." functions.

Before the interfaces will be accessible to other programs in the Windows system, they must be registered by GS-Base with the "Register GS-Base COM Interfaces" command (on the GS-Base "Settings" menu).

Creating and saving a new database

```
#include <stdio.h>
#include <stdio.h>
#include <comdef.h>

#import "E:\gsbase\gsbase.exe"

using namespace GSBASELib;

struct StartOle
{
        StartOle() { ::CoInitialize(NULL); }
        ~StartOle() { ::CoUninitialize(); }
} _startOle;
```

```
try
{
        IApplicationPtr app;
        app.CreateInstance(L"GSBase.Application");

        IDatabasePtr dbase = app->NewDatabase();

        // insert a new table "table1" in the root ("") folder at
the bottom;
        // if "table1" already exists, the uniqueName will
contain a unique
        // name table1(1)...table1(n) that GS-Base will create
and use instead;
        _bstr_t uniqueName = dbase-
>InsertDatabaseItem(_bstr_t(""), 1, _bstr_t("table1"));

        IFieldParamsPtr field = dbase->CreateFieldParams();

        // add one Text field to the currently selected table;
        //
        // adding and importing a table make it 'selected'
automatically;
        // deleting tables may result in a new selection;
        // SetActiveTable() and GetActiveTable() set and retrieve
the selection
        //
        field->put_name(_bstr_t("field1"));
        // T - text field
        // N - numeric field
        // M - long text / Memo
        // O - objects (images, files etc.)
        // C - code field (long text with specific syntax
highlighting)
        field->put_type('T');
        dbase->AppendField(field);

        //add one Number field with range validation
        //
        field->Reset();
        field->put_name(_bstr_t("field2"));
        field->put_type('N');
        field->put_formula(_bstr_t("=(field2 > 1) * (field2 <
10)"));
        // 1 - calculation formula/calculated field
        // 2 - validation formula
        // 3 - conversion formula
        // 4 - default value
        // 5 - incremented maximum
        field->put_formulaType(2);
```

```
        dbase->AppendField(field);

        //add one calculated Number field
        //
        field->Reset();
        field->put_name(_bstr_t("field3"));
        field->put_type('N');
        field->put_formula(_bstr_t("=field2 * 10"));
        field->put_formulaType(1);
        dbase->AppendField(field);

        //insert one more Text field at the beginning of the
record
        field->Reset();
        field->put_name(_bstr_t("field4"));
        field->put_type('T');
        dbase->InsertField(1, field);

        //add one Code field that uses the "cpp" syntax
highlighting
        field->Reset();
        field->put_name(_bstr_t("field5"));
        field->put_type('C');
        //subtypes same as in the "Field Setup": "cpp",
"assembler", "php"...
        field->put_subtype(_bstr_t("cpp"));
        dbase->AppendField(field);

        //choose to use the standard zip (zip32) file format for
the new database;
        the default value for new files is "true" ("use zip64");
        for existing database files their the default value is
the one that was used previously;

        //dbase->put_zip64(true);
        dbase->put_zip64(false);

        //save a new database (with one table and no records so
far);
        //in c++ the special '\' character in a string must be
doubled
        dbase->SaveDatabaseAs(_bstr_t("e:\\test_dbase.gsb"));
        dbase->Close();
}
catch(_com_error error)
{
        // ...
}
```

Editing an existing database

```
#include <stdio.h>
#include <stdio.h>
#include <comdef.h>

#import "E:\gsbase\gsbase.exe"

using namespace GSBASELib;

struct StartOle
{
        StartOle() { ::CoInitialize(NULL); }
        ~StartOle() { ::CoUninitialize(); }
} _startOle;

try
{
        IApplicationPtr app;
        app.CreateInstance(L"GSBase.Application");

        BSTR password = NULL;
        IDatabasePtr dbase = app-
>OpenDatabase(_bstr_t("e:\\test_dbase.gsb"), password);

        //all editing actions always refer to the currently
selected table;
        //once a table is selected (and the database is saved),
it remains selected till you change this;
        //
        if (dbase->GetActiveTable() != _bstr_t("table1"))
                dbase->SetActiveTable("table1");

        //as the table contains the calculated "field3", for
performance reasons turn off
        //recalculation of each row which would otherwise occur
after each of the 10,000 modifications below;
        //to restore the default automatic updating use
"automatic" or re-open the database;
        dbase->put_updateMode(_bstr_t("manual"));

        //Note: for best performance, when inserting a large
series of data, always fill the table fields
        //in the "top to bottom" (and "left to right") order. The
"bottom to top" order may
        //be considerably slower.
```

```
        //insert the following date string in the 1st record
field in 10,000 records;
        _bstr_t today = "2021-01-15";
        for (__int64 i = 1; i <= 100000; ++i)
                dbase->InsertText(i, 1, today);

        //insert some numbers in the 3rd record field in the
first 10,000 records
        for (__int64 i = 1; i <= 100000; ++i)
                dbase->InsertNumber(i, 3, 2.0 + i%8);

        //update all calculated fields in this table using 4
processor cores
        dbase->UpdateTable(4);

        //get the sum of the 4th field for the entire current
record set
        __int64 counter = dbase->GetRecordSetCount();
        double sum = 0;
        for (__int64 i = 1; i <= counter; ++i)
                sum += dbase->GetNumber(i, 4);

        //clear the 3rd field in records 11 to 21
        dbase->ClearRange(11, 3, 21, 3);

        //update all "field3" calculated fields as the "manual"
update mode was set earlier
        dbase->UpdateTable(4);

        //remove record 1
        dbase->RemoveRecords(1, 1);
        //remove records 2 to 3
        dbase->RemoveRecords(2, 3);
        //remove record 11
        dbase->RemoveRecords(11, 11);

        dbase->SaveDatabase();
        dbase->Close();
}
catch(_com_error error)
{
        // ...
}
```

Copying tables and folders

```
#include <stdio.h>
#include <stdio.h>
#include <comdef.h>

#import "E:\gsbase\gsbase.exe"

using namespace GSBASELib;

struct StartOle
{
        StartOle() { ::CoInitialize(NULL); }
        ~StartOle() { ::CoUninitialize(); }
} _startOle;


try
{
        IApplicationPtr app;
        app.CreateInstance(L"GSBase.Application");

        BSTR password = NULL;
        IDatabasePtr dbase = app-
>OpenDatabase(_bstr_t("e:\\test_dbase.gsb"), password);

        //notes:
        //the special "\" characters in string must be doubled in
this c++ code;
        //the trailing "\" determines whether the
InsertDatabaseItem method is to create a table or a folder;
        //
        //insert a new folder "folder1\" in the root folder at
the bottom;
        //if "folder1" already exists at that level, the
uniqueName will contain a unique
        //name folder1(1)...folder1(n) that GS-Base will create
and use instead;
        _bstr_t uniqueName = dbase-
>InsertDatabaseItem(_bstr_t(""), 1, _bstr_t("folder1\\"));

        //insert folder2 at the top
        uniqueName = dbase->InsertDatabaseItem(_bstr_t(""), 0,
_bstr_t("folder2\\"));

        uniqueName = dbase-
>InsertDatabaseItem(_bstr_t("\\folder1\\"), 1,
_bstr_t("folder1\\"));
```

```
        uniqueName = dbase-
>InsertDatabaseItem(_bstr_t("folder1"), 1, _bstr_t("folder1\\"));

        //import the "product" table form the sample.zip database
and insert it in the root folder
        uniqueName = dbase-
>ImportTable(_bstr_t("e:\\sample.zip"), _bstr_t("products"),
password, _bstr_t("\\"));

        //import it again and insert it in the "folder1" folder
        uniqueName = dbase-
>ImportTable(_bstr_t("c:\\sample.zip"), _bstr_t("products"),
password, _bstr_t("folder1"));

        //create text file parameters
        ITextParamsPtr txt = app->CreateTextParams();
        //use "," as the field separator
        txt->put_separator(_bstr_t("*"));

        //import a new table from the "text_abc.txt" text file
and place it in the "folder2" folder
        uniqueName = dbase-
>ImportTextTable(_bstr_t("c:\\test_abc.txt"), txt,
_bstr_t("\\folder2\\"));
        //import it again (the name of the imported table will be
modified to "text_abc(1)")
        uniqueName = dbase-
>ImportTextTable(_bstr_t("c:\\test_abc.txt"), txt,
_bstr_t("\\folder2\\"));
        //import it again (the name of the imported table will be
modified to "text_abc(2)")
        uniqueName = dbase-
>ImportTextTable(_bstr_t("c:\\test_abc.txt"), txt,
_bstr_t("\\folder2"));

        //rename the "text_abc(2)" table in the "\folder2" folder
to "test_abc_b";
        uniqueName = dbase-
>RenameDatabaseItem(_bstr_t("\\folder2\\test_abc(2)"),
_bstr_t("test_abc_b"));

        //copy the "\folder2\test_abc_b" table to "\folder1" and
place it at the top, before "products" table
        uniqueName = dbase-
>CopyDatabaseItem(_bstr_t("\\folder2\\test_abc_b"),
_bstr_t("\\folder1\\products"));

        //copy the entire "\\folder2" - it'll be duplicated as
"folder2(1)"
```

```
        uniqueName = dbase-
>CopyDatabaseItem(_bstr_t("\\folder2\\"), _bstr_t("\\"));

        //delete the entire "folder2" folder
        dbase->DeleteDatabaseItem(_bstr_t("\\folder2\\"));

        //delete the "\folder1\test_abc_b" table
        dbase-
>DeleteDatabaseItem(_bstr_t("\\folder1\\test_abc_b"));

        dbase->SaveDatabase();
        dbase->Close();
}
catch(_com_error error)
{
        // ...
}
```

Formatting fields

```


#include <stdio.h>
#include <stdio.h>
#include <comdef.h>

#import "E:\gsbase\gsbase.exe"

using namespace GSBASELib;

struct StartOle
{
        StartOle() { ::CoInitialize(NULL); }
        ~StartOle() { ::CoUninitialize(); }
} _startOle;


try
{
        IApplicationPtr app;
        app.CreateInstance(L"GSBase.Application");

        BSTR password = NULL;
        IDatabasePtr dbase = app-
>OpenDatabase(_bstr_t("e:\\test_dbase.gsb"), password);
```

```
        //select the "table1" table in the root folder
        dbase->SetActiveTable(_bstr_t("\\table1"));

        //create formatting settings
        IFormatParamsPtr format = dbase->CreateFormatParams();

        //set the currency format: variable/automatic number of
decimals and the exponent value
        //parameters:
        //1. decimals: 0 - 14 | "auto"
        //2. currency position: "$1.1" | "$ 1.1" | "1.1$" | "1.1
$"
        //3. currency symbol: "$", "GBP" etc.
        //4. true - use curly braces for negative values
        //5. true - use red color for negative values
        //
        format->SetCurrencyFormat(_bstr_t("2"), _bstr_t("$1.1"),
_bstr_t("gbp"), true, true);

        // other style examples:
        //
        //set the scientific style: 5 decimal digits and the
fixed "07" exponent
        //
        //----- format->SetScientificFormat(_bstr_t("5"),
_bstr_t("07"));
        //
        //set the scientific style: variable/automatic number of
decimals and the exponent value
        //
        //----- format->SetScientificFormat(_bstr_t("auto"),
_bstr_t("auto"));
        //
        //
        //set the accounting style
        //parameters:
        //1. decimals: 0 - 14 | "auto"
        //2. currency symbol: "$", "GBP" etc.
        //
        //----- format->SetAccountingFormat(_bstr_t("2"),
_bstr_t("gbp"));
        //
        //
        //set the fractional format
        //parameters:
        //1. if the 2nd parameter is "false", (1) is a
denominator value 2, 3, ..., n;
        //   if the 2nd parameter is "true", (1) is a fixed
number of denominator digits 1...14
```

```
        //2. ...
        //
        //----- format->SetFractionFormat(2, true);
        //
        //
        //set the general number format
        //parameters:
        //1. decimals: 0 - 14 | "auto"
        //2. leading zeroes: 0 - 14
        //3. true - use curly braces for negative values
        //4. true - use red color for negative values
        //5. true - use the thousand separator
        //
        //----- format->SetGeneralNumberFormat(_bstr_t("auto"),
5, false, false, true);
        //

        dbase->SetFieldFormat(3, format);

        //set the date format
        //parameters:
        //1. a date pattern: same as in the "Format > Style"
window
        //2. 1 - always switch to the current Windows system
day/month order ("m/d..." | "d/m...")
        format->SetDateFormat(_bstr_t("m/d/yyyy"), 1);
        dbase->SetFieldFormat(1, format);

        //clear the previously set information
        format->Reset();

        format->put_fontSize(14);
        format->put_boldFont(true);
        dbase->SetFieldFormat(3, format);

        dbase->SaveDatabase();
        dbase->Close();
}
catch(_com_error error)
{
        // ...
}
```

Setting column/field widths and row heights

```
#include <stdio.h>
#include <stdio.h>
#include <comdef.h>

#import "E:\gsbase\gsbase.exe"

using namespace GSBASELib;

struct StartOle
{
        StartOle() { ::CoInitialize(NULL); }
        ~StartOle() { ::CoUninitialize(); }
} _startOle;


try
{
        IApplicationPtr app;
        app.CreateInstance(L"GSBase.Application");

        BSTR password = NULL;
        IDatabasePtr dbase = app-
>OpenDatabase(_bstr_t("e:\\test_dbase.gsb"), password);

        if (dbase->GetActiveTable() != _bstr_t("table1"))
                dbase->SetActiveTable("table1");

        //get the 1st column/field width in screen pixels
        unsigned short width = dbase->GetColumnWidth(1);

        //set a new width
        width += 50;
        dbase->SetColumnWidth(1, width);

        //fit the 3rd column/field width to the data in that
column/field
        dbase->FitColumnWidth(3, 3);

        //get the 9th row/record height in screen pixels:
typically it should be 25px
        unsigned short height = dbase->GetRowHeight(9);

        //set a new height
        height += 20;
        dbase->SetRowHeight(9, height);

        //check the "auto-height" state of the 9th row
        BOOL autoHeight = dbase->GetAutoRowHeight(9);
```

```
        //after the previous SetRowHeight() it should be FALSE
        assert(!autoHeight);

        //restore on the "auto-height" state for the 9th row
        dbase->SetAutoRowHeight(9, 9, TRUE);

        //should be TRUE now
        autoHeight = dbase->GetAutoRowHeight(9);
        assert(autoHeight);

        //it should be 25px again
        height = dbase->GetRowHeight(9);

        dbase->SaveDatabase();
        dbase->Close();
}
catch(_com_error error)
{
        // ...
}
```

Browsing the folders/tables tree

```
#include <stdio.h>
#include <stdio.h>
#include <comdef.h>

#import "E:\gsbase\gsbase.exe"

using namespace GSBASELib;

struct StartOle
{
        StartOle() { ::CoInitialize(NULL); }
        ~StartOle() { ::CoUninitialize(); }
} _startOle;


try
{
        IApplicationPtr app;
        app.CreateInstance(L"GSBase.Application");
```

```
        BSTR password = NULL;
        IDatabasePtr dbase = app-
>OpenDatabase(_bstr_t("e:\\test_dbase.gsb"), password);

        //1st method

        //get the total number of tables and folders in the
main/root folder (including nested folders)
        __int64 counter = dbase->GetDatabaseItemCount();

        //iterate through all table/folders
        for (int i = 1; i <= counter; ++i)
        {
                char path[MAX_PATH] = { 0 };
                ::strcpy(path, dbase->GetDatabaseItem(i));
                size_t length = ::strlen(path);
                if (length && path[length - 1] == '\\')
                {
                        //folder
                }
                else
                {
                        //table
                }
        }

        //2nd method

        //get the first "child" element in the specified folder
        _bstr_t bpath = dbase-
>GetFirstDatabaseItem(_bstr_t("\\folder2(1)\\"));

        //iterate through direct "child" elements of the
specified folder (not expanding nested folders)
        while (bpath.length())
        {
                char path[MAX_PATH] = { 0 };
                ::strcpy(path, bpath);
                size_t length = ::strlen(path);
                if (length && path[length - 1] == '\\')
                {
                        //folder
                }
                else
                {
                        //table
                }
                bpath = dbase->GetNextDatabaseItem(bpath);
        }
```

```
        dbase->Close();
}
catch(_com_error error)
{
        // ...
}
```

Searching and sorting

```
#include <stdio.h>
#include <stdio.h>
#include <comdef.h>

#import "E:\gsbase\gsbase.exe"

using namespace GSBASELib;

struct StartOle
{
        StartOle() { ::CoInitialize(NULL); }
        ~StartOle() { ::CoUninitialize(); }
} _startOle;


try
{
        IApplicationPtr app;
        app.CreateInstance(L"GSBase.Application");

        BSTR password = NULL;
        IDatabasePtr dbase = app-
>OpenDatabase(_bstr_t("e:\\test_dbase.gsb"), password);

        //select the "table1" table in the root folder
        dbase->SetActiveTable(_bstr_t("\\products"));

        IFieldParamsPtr fparams = dbase->CreateFieldParams();

        //find the "ProductName" and "UnitPrice" fields;
        //filter "ProductName" and sort "UnitPrice"

        //reset the previous sorting indices - resetting should
be used before calling "put_sortIndex()"
```

```
        dbase->ResetSorting();

        unsigned short int iname = 0, iprice = 0;

        for (unsigned short int i = 1; i <= dbase-
>GetFieldCount() && (!iname || !iprice); ++i)
        {
                dbase->GetField(i, fparams);
                BSTR fname = NULL;
                fparams->get_name(&fname);
                if (!iname && _bstr_t(fname) ==
_bstr_t(L"ProductName"))
                {
                        //set the "\Ai" RegEx filter for
"ProductName" (=search for names starting with "I");
                        //searching is performed automatically if
a call to the "SetField" changes the "filter" value;
                        //note: in this version the filter type
is always "RegEx"
                        fparams->put_filter(_bstr_t(L"\\Ai"));

                        dbase->SetField(iname = i, fparams);
                }
                if (!iprice && _bstr_t(fname) ==
_bstr_t(L"UnitPrice"))
                {
                        //set the 1 as the sorting index for
"UnitPrice";
                        //sorting is performed automatically
after a call to "SetField" if the "sorting" index is modified;
                        //to create a compound sorting index, use
subsequent numbers (2, 3...) for further fields;
                        //note: using an index not in that
strictly incremented manner causes an error;

                        fparams->put_sortIndex(1);

                        // 'A' - ascending order, 'D' -
descending order
                        fparams->put_sortOrder('A');

                        dbase->SetField(iprice = i, fparams);
                }
                if (fname)
                        ::SysFreeString(fname);
        }

        dbase->SaveDatabase();
        dbase->Close();
```

```
}
catch(_com_error error)
{
        // ...
}
```

Predefined searching

```
#include <stdio.h>
#include <stdio.h>
#include <comdef.h>

#import "E:\gsbase\gsbase.exe"

using namespace GSBASELib;

struct StartOle
{
        StartOle() { ::CoInitialize(NULL); }
        ~StartOle() { ::CoUninitialize(); }
} _startOle;


try
{
        IApplicationPtr app;
        app.CreateInstance(L"GSBase.Application");

        BSTR password = NULL;
        IDatabasePtr dbase = app-
>OpenDatabase(_bstr_t("e:\\test_dbase.gsb"), password);

        //select the "table1" table in the root folder
        dbase->SetActiveTable(_bstr_t("\\products"));

        //find duplicates in the 5th field
        dbase->FindDuplicates(5, 5);

        //check the results
        __int64 counter1 = dbase->GetRecordTotalCount();
        __int64 counter2 = dbase->GetRecordSetCount();

        //find filtered duplicates in the 5th field (the 2nd and
```

```
subsequent occurrences of a given duplicated value)
        dbase->FindFilteredDuplicates(5, 5);

        //check the results
        counter1 = dbase->GetRecordTotalCount();
        counter2 = dbase->GetRecordSetCount();

        //find records with the flag "1"
        dbase->FindFlagged(1);

        // ...

        //find the records not included in the current record set
        dbase->FindCompliment();

        // ...

        //clear all filters and display all records
        dbase->FindAll();

        // ...

        dbase->Close();
}
catch(_com_error error)
{
        // ...
}
```

File password protection

```
#include <stdio.h>
#include <stdio.h>
#include <comdef.h>

#import "E:\gsbase\gsbase.exe"

using namespace GSBASELib;

struct StartOle
{
        StartOle() { ::CoInitialize(NULL); }
        ~StartOle() { ::CoUninitialize(); }
```

```
} _startOle;


try
{
        IApplicationPtr app;
        app.CreateInstance(L"GSBase.Application");


        //set a password
        BSTR password = NULL;
        IDatabasePtr dbase = app-
>OpenDatabase(_bstr_t("e:\\test_dbase.gsb"), password);

        dbase->SetFilePassword(true, _bstr_t("Twofish"),
_bstr_t(""), _bstr_t("rocc4545"));

        dbase->SaveDatabase();
        dbase->Close();

        //open a password-protected database
        dbase = app->OpenDatabase(_bstr_t("e:\\test_dbase.gsb"),
_bstr_t("rocc4545"));
        //...
        //dbase->SaveDatabase();
        dbase->Close();


        //remove password protection
        dbase = app->OpenDatabase(_bstr_t("e:\\test_dbase.gsb"),
_bstr_t("rocc4545"));

        dbase->SetFilePassword(false, _bstr_t("blowfish"),
_bstr_t("rocc4545"), _bstr_t((char*)NULL));

        dbase->SaveDatabase();
        dbase->Close();
}
catch(_com_error error)
{
        // ...
}
```

Opening and saving text files

```
#include <stdio.h>
#include <stdio.h>
#include <comdef.h>

#import "E:\gsbase\gsbase.exe"

using namespace GSBASELib;

struct StartOle
{
        StartOle() { ::CoInitialize(NULL); }
        ~StartOle() { ::CoUninitialize(); }
} _startOle;


try
{
        IApplicationPtr app;
        app.CreateInstance(L"GSBase.Application");

        // 1. Exporting the current record set to a text file
        // ------------------------------------------------

        BSTR password = NULL;
        IDatabasePtr dbase = app-
>OpenDatabase(_bstr_t("e:\\test_dbase.gsb"), password);

        //select the "table1" table in the root folder
        dbase->SetActiveTable(_bstr_t("\\products"));

        //create text file parameters
        ITextParamsPtr txt = app->CreateTextParams();

        //use ";" as the field separator; the default value is
","
        txt->put_separator(_bstr_t(";"));

        //use of separators can be turned off; the default value
is true
        //txt->put_useSeparator(false);

        //use "'" as the quoting symbol; the default value is """
        txt->put_quotingSymbol(_bstr_t("'"));

        //use of quoting symbols can be turned off; the default
value is true
        //txt->put_useQuoting(false);
```

```
        //save field names in the first row; the default value is
true
        txt->put_fieldNames(true);

        //change the text encoding: "utf8" | "windows" | "dos";
the default value is "utf8"
        txt->put_encoding(_bstr_t("utf8"));

        //export the current table to a text file;
        //"dbase" remains the originally opened database and can
edited further as usual
        dbase->SaveTextFileAs("e:\\test_b.txt", txt);


        // 2. Opening and saving a text file
        // --------------------------------

        //re-use the above "txt" settings and add new ones

        //if a column contains textual representations of
numbers, try converting them to a number field
        txt->put_loadNumbers(true);

        //if a column contains textual representations of dates
in various formats, convert these strings
        //to the generic "DT" W3 text representation of dates in
GS-Base (please see the "data types" help topic for details).
        txt->put_loadDates(true);

        IDatabasePtr textFile = app-
>OpenTextFile("e:\\test_b.txt", txt);

        unsigned short int fcounter = textFile->GetFieldCount();

        IFieldParamsPtr fparams = textFile->CreateFieldParams();

        textFile->GetField(1, fparams);
        //
        // ...perform any editing, field changes etc.
        //

        //save the edited text file
        textFile->SaveTextFile(txt);

        textFile->Close();

        dbase->Close();
```

```
        // 3. Opening a text file and saving it as a database
        // ------------------------------------------------

        //re-use the above "txt"
        txt->put_loadNumbers(false);

        textFile = app->OpenTextFile("e:\\test_b.txt", txt);

        //SaveDatabaseAs changes "textFile" to a database with
the path given below; the text file is closed
        textFile->SaveDatabaseAs(_bstr_t("e:\\sample_b.gsb"));

        //
        // ...perform any editing, field actions etc. with
"e:\\sample_b.gsb"
        //

        textFile->Close();

        dbase->Close();
}
catch(_com_error error)
{
        // ...
}
```

Error handling

```
#include <stdio.h>
#include <stdio.h>
#include <comdef.h>

#import "E:\gsbase\gsbase.exe"

using namespace GSBASELib;

struct StartOle
{
        StartOle() { ::CoInitialize(NULL); }
        ~StartOle() { ::CoUninitialize(); }
} _startOle;


try
```

```
{
        IApplicationPtr app;
        app.CreateInstance(L"GSBase.Application");

        BSTR password = NULL;
        IDatabasePtr dbase = app-
>OpenDatabase(_bstr_t("e:\\test_dbase.gsb"), password);

        //
        // ...
        //

        // if a COM function returns an error other than
E_OUTOFMEMORY,
        // additional error information can obtained via the
"lastError" property;
        BYTE code = 0;
        dbase->get_lastError(&code);

        // 21   // Out of memory while creating/editing worksheet
data
        // 22   // A database must contain at least one table
        // 37   // Invalid password.
        // 53   // Not allowed field type change - e.g.
conversion from "Files/Images" to "Number"
        // 61   // Can't open the specified file
        // 62   // Can't open or create the specified file
        // 63   // Error while closing the specified file
        // 64   // Error while repositioning a file pointer
        // 65   // Error while reading from a file
        // 66   // Error while writing to a file
        // 67   // Error while deleting a file
        // 68   // Error while checking the file size/info
        // 69   // Error while allocating a file read/write
buffer
        // 70   // Can't open the specified file. File in use.
        // 71   // Error while decrypting a file.
        // 72   // Error while encrypting a file.
        // 73   // Error while reading binary fields.
        // 79   // Can't find the manifest file or its content is
incorrect
        // 80   // The file format requires a newer GS-Base
version.
        // 119  // Too many zip streams in a standard zip (zip32)
file
        // 120  // Inconsistent zip stream state
        // 121  // Corrupted zip stream data
        // 122  // Out of memory while processing a zip stream
        // 123  // Unexpected end of zip stream
```

```
        // 125  // Unknown zlib error
        // 131  // Some data in the file can't be converted to
numeric field values.

        dbase->Close();
}
catch(_com_error error)
{
        // ...
}
```

Related Topics

Interfaces


```
        [
                dual,
                helpstring("IXBaseParams Interface"),
                pointer_default(unique)
        ]
        interface IXBaseParams : IDispatch
        {
                [propget, id(1), helpstring("property format")]
HRESULT format([out, retval] BSTR *pFormat);
                [propput, id(1), helpstring("property format")]
HRESULT format([in] BSTR format);

                [propget, id(2), helpstring("property encoding")]
HRESULT encoding([out, retval] BSTR *pEncoding);
                [propput, id(2), helpstring("property encoding")]
HRESULT encoding([in] BSTR encoding);

                [id(3), helpstring("method GetFieldCount")]
HRESULT GetFieldCount([out, retval] int *pCounter);

                [id(4), helpstring("method SetField")] HRESULT
SetField([in] int index, [in] BSTR name, [in] BSTR type, [in] int
length, [in] int decimals);
                [id(5), helpstring("method AddField")] HRESULT
```

```
AddField([in] BSTR name, [in] BSTR type, [in] int length, [in]
int decimals);
                [id(6), helpstring("method InsertField")] HRESULT
InsertField([in] int index, [in] BSTR name, [in] BSTR type, [in]
int length, [in] int decimals);

                [id(7), helpstring("method GetFieldName")]
HRESULT GetFieldName([in] int index, [out, retval] BSTR *pName);
                [id(8), helpstring("method GetFieldType")]
HRESULT GetFieldType([in] int index, [out, retval] BSTR *pType);
                [id(9), helpstring("method GetFieldLength")]
HRESULT GetFieldLength([in] int index, [out, retval] int
*pLength);
                [id(10), helpstring("method GetFieldDecimals")]
HRESULT GetFieldDecimals([in] int index, [out, retval] int
*pDecimals);

                [id(11), helpstring("method DeleteField")]
HRESULT DeleteField([in] int index);
        };

        [
                dual,
                helpstring("ITextParams Interface"),
                pointer_default(unique)
        ]
        interface ITextParams : IDispatch
        {
                [propget, id(1), helpstring("property
separator")] HRESULT separator([out, retval] BSTR *pSeparator);
                [propput, id(1), helpstring("property
separator")] HRESULT separator([in] BSTR separator);

                [propget, id(2), helpstring("property
useSeparator")] HRESULT useSeparator([out, retval] BOOL*
pUseSeparator);
                [propput, id(2), helpstring("property
useSeparator")] HRESULT useSeparator([in] BOOL useSeparator);

                [propget, id(3), helpstring("property
quotingSymbol")] HRESULT quotingSymbol([out, retval] BSTR
*pSymbol);
                [propput, id(3), helpstring("property
quotingSymbol")] HRESULT quotingSymbol([in] BSTR symbol);

                [propget, id(4), helpstring("property
useQuoting")] HRESULT useQuoting([out, retval] BOOL* pUseSymbol);
                [propput, id(4), helpstring("property
useQuoting")] HRESULT useQuoting([in] BOOL useSymbol);
```

```
                [propget, id(5), helpstring("property encoding")]
HRESULT encoding([out, retval] BSTR* pEncoding);
                [propput, id(5), helpstring("property encoding")]
HRESULT encoding([in] BSTR encoding);

                [propget, id(6), helpstring("property
fieldNames")] HRESULT fieldNames([out, retval] BOOL* pValue);
                [propput, id(6), helpstring("property
fieldNames")] HRESULT fieldNames([in] BOOL value);

                [propget, id(7), helpstring("property
parseNumbers")] HRESULT parseNumbers([out, retval] BOOL *pValue);
                [propput, id(7), helpstring("property
parseNumbers")] HRESULT parseNumbers([in] BOOL value);

                [propget, id(8), helpstring("property
parseDates")] HRESULT parseDates([out, retval] BOOL *pValue);
                [propput, id(8), helpstring("property
parseDates")] HRESULT parseDates([in] BOOL value);

                [propget, id(9), helpstring("property
fixedFieldWidths")] HRESULT fixedFieldWidths([out, retval] BSTR
*pColumnWidths);
                [propput, id(9), helpstring("property
fixedFieldWidths")] HRESULT fixedFieldWidths([in] BSTR
columnWidths);

                [propget, id(10), helpstring("property
saveObjectsAsZip")] HRESULT saveObjectsAsZip([out, retval] BOOL
*pValue);
                [propput, id(10), helpstring("property
saveObjectsAsZip")] HRESULT saveObjectsAsZip([in] BOOL value);

                [propget, id(11), helpstring("property
saveLongTextAsZip")] HRESULT saveLongTextAsZip([out, retval] BOOL
*pValue);
                [propput, id(11), helpstring("property
saveLongTextAsZip")] HRESULT saveLongTextAsZip([in] BOOL value);

                [propget, id(12), helpstring("property
autoFitColumns")] HRESULT autoFitColumns([out, retval] BOOL*
pValue);
                [propput, id(12), helpstring("property
autoFitColumns")] HRESULT autoFitColumns([in] BOOL value);
        };

        [
                dual,
```

```
                helpstring("IFormatParams Interface"),
                pointer_default(unique)
        ]
        interface IFormatParams : IDispatch
        {
                [id(1), helpstring("method
SetGeneralNumberFormat")] HRESULT SetGeneralNumberFormat([in]
BSTR decimals, [in] BYTE zeroes, [in] BOOL brackets, [in] BOOL
inRed, [in] BOOL separators);
                [id(2), helpstring("method SetCurrencyFormat")]
HRESULT SetCurrencyFormat([in] BSTR decimals, [in] BSTR position,
[in] BSTR symbol, [in] BOOL brackets, [in] BOOL inRed);
                [id(3), helpstring("method SetAccountingFormat")]
HRESULT SetAccountingFormat([in] BSTR decimals, [in] BSTR
symbol);
                [id(4), helpstring("method SetDateFormat")]
HRESULT SetDateFormat([in] BSTR pattern, [in] BOOL systemOrder);
                [id(5), helpstring("method SetTimeFormat")]
HRESULT SetTimeFormat([in] BSTR pattern);
                [id(6), helpstring("method SetDateTimeFormat")]
HRESULT SetDateTimeFormat([in] BSTR datePattern, [in] BSTR
timePattern, [in] BOOL systemOrder, [in] BOOL timeFirst);
                [id(7), helpstring("method SetPercentFormat")]
HRESULT SetPercentFormat([in] BSTR decimals, [in] int scaling);
                [id(8), helpstring("method SetFractionFormat")]
HRESULT SetFractionFormat([in] long denominator, [in] BOOL
digits);
                [id(9), helpstring("method SetScientificFormat")]
HRESULT SetScientificFormat([in] BSTR decimals, [in] BSTR
exponent);

                [propget, id(10), helpstring("property
fontName")] HRESULT fontName([out, retval] BSTR *pVal);
                [propput, id(10), helpstring("property
fontName")] HRESULT fontName([in] BSTR newVal);
                [propget, id(11), helpstring("property
fontSize")] HRESULT fontSize([out, retval] int *pVal);
                [propput, id(11), helpstring("property
fontSize")] HRESULT fontSize([in] int newVal);

                [propget, id(12), helpstring("property
language")] HRESULT language([out, retval] BSTR *pLanguage);
                [propput, id(12), helpstring("property
language")] HRESULT language([in] BSTR language);

                [propget, id(13), helpstring("property
boldFont")] HRESULT boldFont([out, retval] BOOL *pVal);
                [propput, id(13), helpstring("property
boldFont")] HRESULT boldFont([in] BOOL newVal);
```

```
                [propget, id(14), helpstring("property
italicFont")] HRESULT italicFont([out, retval] BOOL *pVal);
                [propput, id(14), helpstring("property
italicFont")] HRESULT italicFont([in] BOOL newVal);
                [propget, id(15), helpstring("property
underlineFont")] HRESULT underlineFont([out, retval] BOOL *pVal);
                [propput, id(15), helpstring("property
underlineFont")] HRESULT underlineFont([in] BOOL newVal);
                [propget, id(16), helpstring("property
strikeoutFont")] HRESULT strikeoutFont([out, retval] BOOL *pVal);
                [propput, id(16), helpstring("property
strikeoutFont")] HRESULT strikeoutFont([in] BOOL newVal);
                [propget, id(17), helpstring("property
fontColor")] HRESULT fontColor([out, retval] BSTR *pColor);
                [propput, id(17), helpstring("property
fontColor")] HRESULT fontColor([in] BSTR color);

                [propget, id(18), helpstring("property
horzAlignment")] HRESULT horzAlignment([out, retval] BSTR
*pHorzAlign);
                [propput, id(18), helpstring("property
horzAlignment")] HRESULT horzAlignment([in] BSTR horzAlign);
                [propget, id(19), helpstring("property
vertAlignment")] HRESULT vertAlignment([out, retval] BSTR
*pVertAlign);
                [propput, id(19), helpstring("property
vertAlignment")] HRESULT vertAlignment([in] BSTR vertAlign);

                [propget, id(20), helpstring("property
wrapText")] HRESULT wrapText([out, retval] BOOL *pValue);
                [propput, id(20), helpstring("property
wrapText")] HRESULT wrapText([in] BOOL value);

                [propget, id(21), helpstring("property
shrinkText")] HRESULT shrinkText([out, retval] BOOL *pValue);
                [propput, id(21), helpstring("property
shrinkText")] HRESULT shrinkText([in] BOOL value);

                [id(22), helpstring("method Reset")] HRESULT
Reset();
        };

        [
                dual,
                helpstring("IFieldParams Interface"),
                pointer_default(unique)
        ]
        interface IFieldParams : IDispatch
        {
```

```
                [propput, id(1), helpstring("property field
name")] HRESULT name([in] BSTR pVal);
                [propget, id(1), helpstring("property field
name")] HRESULT name([out, retval] BSTR* pVal);

                [propput, id(2), helpstring("property field
type")] HRESULT type([in] BSTR newVal);
                [propget, id(2), helpstring("property field
type")] HRESULT type([out, retval] BSTR *newVal);

                [propput, id(3), helpstring("property field
subtype")] HRESULT subtype([in] BSTR stype);
                [propget, id(3), helpstring("property field
subtype")] HRESULT subtype([out, retval] BSTR* pStype);

                [propput, id(4), helpstring("property
hyperlink")] HRESULT hlink([in] BOOL pVal);
                [propget, id(4), helpstring("property
hyperlink")] HRESULT hlink([out, retval] BOOL* pVal);

                [propput, id(5), helpstring("property
sortIndex")] HRESULT sortIndex([in] unsigned short newVal);
                [propget, id(5), helpstring("property
sortIndex")] HRESULT sortIndex([out, retval] unsigned short*
newVal);

                [propput, id(6), helpstring("property
sortOrder")] HRESULT sortOrder([in] BSTR order);
                [propget, id(6), helpstring("property
sortOrder")] HRESULT sortOrder([out, retval] BSTR* pOrder);

                [propput, id(7), helpstring("property formula")]
HRESULT formula([in] BSTR newVal);
                [propget, id(7), helpstring("property formula")]
HRESULT formula([out, retval] BSTR* newVal);

                [propput, id(8), helpstring("property
formulaType")] HRESULT formulaType([in] unsigned short type);
                [propget, id(8), helpstring("property
formulaType")] HRESULT formulaType([out, retval] unsigned short*
pType);

                [propput, id(9), helpstring("property filter")]
HRESULT filter([in] BSTR newVal);
                [propget, id(9), helpstring("property filter")]
HRESULT filter([out, retval] BSTR* newVal);

                [propput, id(10), helpstring("property
filterType")] HRESULT filterType([in] int newVal);
```

```
                [propget, id(10), helpstring("property
filterType")] HRESULT filterType([out, retval] int* newVal);

                [propput, id(11), helpstring("property
matchWords")] HRESULT matchWords([in] BOOL pVal);
                [propget, id(11), helpstring("property
matchWords")] HRESULT matchWords([out, retval] BOOL* pVal);

                [propput, id(12), helpstring("property
matchCase")] HRESULT matchCase([in] BOOL pVal);
                [propget, id(12), helpstring("property
matchCase")] HRESULT matchCase([out, retval] BOOL* pVal);

                [id(13), helpstring("method Reset")] HRESULT
Reset();
        };

        [
                dual,
                helpstring("IMergeParams Interface"),
                pointer_default(unique)
        ]
        interface IMergeParams : IDispatch
        {
                [propput, id(1), helpstring("property path
pattern")] HRESULT path([in] BSTR val);
                [propget, id(1), helpstring("property path
pattern")] HRESULT path([out, retval] BSTR* val);

                [propput, id(2), helpstring("property table
name")] HRESULT table([in] BSTR val);
                [propget, id(2), helpstring("property table
name")] HRESULT table([out, retval] BSTR* val);

                [propput, id(3), helpstring("property master
field index")] HRESULT masterField([in] int field);
                [propget, id(3), helpstring("property master
field index")] HRESULT masterField([out, retval] int* field);

                [propput, id(4), helpstring("property slave field
index")] HRESULT slaveField([in] int field);
                [propget, id(4), helpstring("property slave field
index")] HRESULT slaveField([out, retval] int* field);

                [propput, id(5), helpstring("property merge
type")] HRESULT mergeType([in] int val);
                [propget, id(5), helpstring("property merge
type")] HRESULT mergeType([out, retval] int* val);
```

```
                [propput, id(6), helpstring("property
matchFieldNames")] HRESULT matchFieldNames([in] int val);
                [propget, id(6), helpstring("property
matchFieldNames")] HRESULT matchFieldNames([out, retval] int*
val);

                [propput, id(7), helpstring("property ignore
empty fields")] HRESULT ignoreEmpty([in] BOOL val);
                [propget, id(7), helpstring("property ignore
empty fields")] HRESULT ignoreEmpty([out, retval] BOOL* val);

                [propput, id(8), helpstring("property allow
undo")] HRESULT enableUndo([in] BOOL val);
                [propget, id(8), helpstring("property allow
undo")] HRESULT enableUndo([out, retval] BOOL* val);
        };

        [
                dual,
                helpstring("IApplication Interface"),
                pointer_default(unique)
        ]
        interface IApplication : IDispatch
        {
                [id(1), helpstring("method CreateTextParams")]
HRESULT CreateTextParams([out, retval] ITextParams**
ppTextParams);
                [id(2), helpstring("method CreateXBaseParams")]
HRESULT CreateXBaseParams([out, retval] IXBaseParams**
ppTextParams);

                [id(3), helpstring("method NewDatabase")] HRESULT
NewDatabase([in] BSTR table);
                [id(4), helpstring("method OpenDatabase")]
HRESULT OpenDatabase([in] BSTR path, [in] BSTR password);
                [id(5), helpstring("method OpenTextFile")]
HRESULT OpenTextFile([in] BSTR path, [in] BOOL showDialogBox,
[in] ITextParams* pTextParams);
                [id(6), helpstring("method OpenExcelFile")]
HRESULT OpenExcelFile([in] BSTR path, [in] BOOL showDialogBox,
[in] int merge, [in] int fnames);
                [id(7), helpstring("method OpenHtmlFile")]
HRESULT OpenHtmlFile([in] BSTR path, [in] int fnames);
                [id(8), helpstring("method OpenXBaseFile")]
HRESULT OpenXBaseFile([in] BSTR path, [in] BOOL showDialogBox,
[in] IXBaseParams* pXBaseParams);
                [id(9), helpstring("method OpenMySQLFile")]
HRESULT OpenMySQLFile([in] BSTR path);
                [id(102), helpstring("method OpenSQLiteFile")]
```

```
HRESULT OpenSQLiteFile([in] BSTR path);

                [id(10), helpstring("method Reload")] HRESULT
Reload();

                [id(11), helpstring("method SaveRecordSetAs")]
HRESULT SaveRecordSetAs([in] BSTR path, [in] BSTR password);
                [id(12), helpstring("method
SaveRecordSetAsText")] HRESULT SaveRecordSetAsText([in] BSTR
path, [in] ITextParams* params);
                [id(13), helpstring("method
SaveRecordSetAsExcel")] HRESULT SaveRecordSetAsExcel([in] BSTR
path, [in] int merge, [in] int fnames, [in] int saveZip);
                [id(14), helpstring("method
SaveRecordSetAsXBase")] HRESULT SaveRecordSetAsXBase([in] BSTR
path, [in] BYTE format, [in] IXBaseParams* params);
                [id(15), helpstring("method
SaveRecordSetAsMySQL")] HRESULT SaveRecordSetAsMySQL([in] BSTR
path);
                [id(103), helpstring("method
SaveRecordSetAsSQLite")] HRESULT SaveRecordSetAsSQLite([in] BSTR
path);
                [id(16), helpstring("method SaveRecordSetAsPDF")]
HRESULT SaveRecordSetAsPDF([in] BSTR path);

                [id(17), helpstring("method Save")] HRESULT
Save();
                [id(18), helpstring("method SaveTextFile")]
HRESULT SaveTextFile([in] ITextParams* params);
                [id(19), helpstring("method SaveExcelFile")]
HRESULT SaveExcelFile([in] int merge, [in] int fnames, [in] int
saveZip);
                [id(20), helpstring("method SaveXBaseFile")]
HRESULT SaveXBaseFile([in] IXBaseParams* params);
                [id(21), helpstring("method SaveMySQLFile")]
HRESULT SaveMySQLFile();
                [id(104), helpstring("method SaveSQLiteFile")]
HRESULT SaveSQLiteFile();

                [id(22), helpstring("method SaveDatabaseAs")]
HRESULT SaveDatabaseAs([in] BSTR path);
                [id(23), helpstring("method
SaveDatabaseAsExcel")] HRESULT SaveDatabaseAsExcel([in] BSTR
path, [in] int merge, [in] int fnames, [in] int saveZip);
                [id(24), helpstring("method
SaveDatabaseAsMySQL")] HRESULT SaveDatabaseAsMySQL([in] BSTR
path);
                [id(105), helpstring("method
SaveDatabaseSQLite")] HRESULT SaveDatabaseAsSQLite([in] BSTR
```

```
path);

                [id(25), helpstring("method Close")] HRESULT
Close();

                [id(26), helpstring("method ImportTable")]
HRESULT ImportTable([in] BSTR filePath, [in] BSTR tablePath, [in]
BSTR password, [in] BSTR folder, [out, retval] BSTR* uniqueName);
                [id(27), helpstring("method ImportTextTable")]
HRESULT ImportTextTable([in] BSTR textFilePath, [in] BSTR
tableName, [in] ITextParams* pTextParams, [in] BSTR folder);
                [id(100), helpstring("method ImportExcelTable")]
HRESULT ImportExcelTable([in] BSTR textFilePath, [in] BSTR
tableName, [in] int fnames, [in] BSTR folder);

                [id(91), helpstring("method CreateMergeParams")]
HRESULT CreateMergeParams([out, retval] IMergeParams**
ppMergeParams);

                [id(28), helpstring("method MergeTable")] HRESULT
MergeTable([in] int masterField, [in] BSTR tablePath, [in] int
slaveField, [in] BOOL allowDuplicates);
                [id(29), helpstring("method MergeRecords")]
HRESULT MergeRecords([in] IMergeParams* mergeParams);
                [id(92), helpstring("method MergeRecords")]
HRESULT MergeRecordsFromTextFile([in] IMergeParams* mergeParams,
[in] ITextParams* params);
                [id(93), helpstring("method MergeRecords")]
HRESULT MergeRecordsFromExcelFile([in] IMergeParams* mergeParams,
[in] int fnames);
                [id(94), helpstring("method MergeRecords")]
HRESULT MergeRecordsFromXBaseFile([in] IMergeParams* mergeParams,
[in] IXBaseParams* params);
                [id(95), helpstring("method MergeRecords")]
HRESULT MergeRecordsFromMySQLFile([in] IMergeParams*
mergeParams);
                [id(106), helpstring("method MergeRecords")]
HRESULT MergeRecordsFromSQLiteFile([in] IMergeParams*
mergeParams);

                [id(30), helpstring("method SetActiveTable")]
HRESULT SetActiveTable([in] BSTR path);
                [id(31), helpstring("method GetActiveTable")]
HRESULT GetActiveTable([out, retval] BSTR* path);

                [id(32), helpstring("method GetRecordCount")]
HRESULT GetRecordTotalCount([out, retval] __int64* counter);
                [id(33), helpstring("method GetRecordSetCount")]
HRESULT GetRecordSetCount([out, retval] __int64* counter);
```

```
                [id(34), helpstring("method GetRecordCount")]
HRESULT GetFieldCount([out, retval] unsigned short int* counter);

                [id(35), helpstring("method FindDuplicates")]
HRESULT FindDuplicates([in] unsigned short int fieldFrom, [in]
unsigned short int fieldTo, [in] BOOL searchAll, [in] BYTE type,
[in] BOOL sortResults);
                [id(36), helpstring("method FindUnique")] HRESULT
FindUnique([in] unsigned short int fieldFrom, [in] unsigned short
int fieldTo, [in] BOOL searchAll, [in] BYTE type, [in] BOOL
sortResults, [in] BOOL useCounters, [in] unsigned short int
counterIndex);
                [id(37), helpstring("method FindQuartile")]
HRESULT FindQuartile([in] unsigned short int field, [in] BOOL
searchAll, [in] BOOL statAll, [in] BYTE type);
                [id(38), helpstring("method FindRandom")] HRESULT
FindRandom([in] __int64 counter);
                [id(39), helpstring("method FindFlagged")]
HRESULT FindFlagged([in] unsigned short int flag);
                [id(40), helpstring("method FindComplement")]
HRESULT FindComplement();
                [id(41), helpstring("method FindAll")] HRESULT
FindAll();

                [id(42), helpstring("method CreateFieldParams")]
HRESULT CreateFieldParams([out, retval] IFieldParams**
fieldParams);
                [id(43), helpstring("method GetField")] HRESULT
GetField([in] unsigned short field, [in/*out, retval*/]
IFieldParams* fieldParams);
                [id(44), helpstring("method SetField")] HRESULT
SetField([in] unsigned short field, [in] IFieldParams*
fieldParams);
                [id(45), helpstring("method ConvertField")]
HRESULT ConvertField([in] unsigned short field, [in] BYTE type);
                [id(46), helpstring("method ResetSorting")]
HRESULT ResetSorting();

                [id(47), helpstring("method AppendField")]
HRESULT AppendField([in] IFieldParams* fieldParams);
                [id(48), helpstring("method InsertField")]
HRESULT InsertField([in] unsigned short field, [in] IFieldParams*
fieldParams);
                [id(49), helpstring("method RemoveField")]
HRESULT RemoveField([in] unsigned short field);

                [id(50), helpstring("method
GetDatabaseItemCount")] HRESULT GetDatabaseItemCount([out,
retval] __int64* pCounter);
```

```
                [id(51), helpstring("method GetDatabaseItem")]
HRESULT GetDatabaseItem([in] __int64 index, [out, retval] BSTR*
path);
                [id(52), helpstring("method
GetFirstDatabaseItem")] HRESULT GetFirstDatabaseItem([in] BSTR
folder, [out, retval] BSTR* name);
                [id(53), helpstring("method
GetNextDatabaseItem")] HRESULT GetNextDatabaseItem([in] BSTR
prevName, [out, retval] BSTR* nextName);
                [id(54), helpstring("method RenameDatabaseItem")]
HRESULT RenameDatabaseItem([in] BSTR path, [in] BSTR name, [out,
retval] BSTR* uniqueName);
                [id(55), helpstring("method InsertDatabaseItem")]
HRESULT InsertDatabaseItem([in] BSTR folder, [in] BOOL bottom,
[in] BSTR name, [out, retval] BSTR* uname);
                [id(56), helpstring("method DeleteDatabaseItem")]
HRESULT DeleteDatabaseItem([in] BSTR path);


                [id(57), helpstring("method RemoveRecords")]
HRESULT RemoveRecords([in] __int64 recordFrom, [in] __int64
recordTo);
                [id(58), helpstring("method InsertRecords")]
HRESULT InsertRecords([in] __int64 recordFrom, [in] __int64
recordTo);
                [id(59), helpstring("method InsertText")] HRESULT
InsertText([in] __int64 record, [in] unsigned short field, [in]
BSTR text);
                [id(60), helpstring("method GetText")] HRESULT
GetText([in] __int64 record, [in] unsigned short field, [out,
retval] BSTR* text);
                [id(61), helpstring("method InsertNumber")]
HRESULT InsertNumber([in] __int64 record, [in] unsigned short
field, [in] double number);
                [id(62), helpstring("method GetNumber")] HRESULT
GetNumber([in] __int64 record, [in] unsigned short field, [out,
retval] double* number);
                [id(63), helpstring("method ClearField")] HRESULT
Clear([in] __int64 record, [in] unsigned short field);
                [id(64), helpstring("method ClearRange")] HRESULT
ClearRange([in] __int64 record1, [in] unsigned short field1, [in]
__int64 record2, [in] unsigned short field2);

                [id(96), helpstring("method InsertSeries")]
HRESULT InsertSeries([in] unsigned short field, [in]__int64
record1, [in] __int64 record2, [in] BOOL copy);
                [id(97), helpstring("method
InsertRecurrenceSeries")] HRESULT InsertRecurrenceSeries([in]
unsigned short field, [in]__int64 record1, [in] __int64 record2,
```

```
BSTR formula);
                [id(98), helpstring("method InsertCustomSeries")]
HRESULT InsertCustomSeries([in] unsigned short field, [in]__int64
record1, [in] __int64 record2, [in] __int64 series);
                [id(99), helpstring("method InsertRandomSeries")]
HRESULT InsertRandomSeries([in] unsigned short field, [in]__int64
record1, [in] __int64 record2, [in] unsigned short distribution,
[in] double param1, [in] double param2);

                [id(65), helpstring("method GetColumnWidth")]
HRESULT GetColumnWidth([in] unsigned short field, [out, retval]
unsigned short* pWidth);
                [id(66), helpstring("method SetColumnWidth")]
HRESULT SetColumnWidth([in] unsigned short field, [in] unsigned
short width);

                [id(67), helpstring("method FitColumnWidth")]
HRESULT FitColumnWidth([in] unsigned short fieldFrom, [in]
unsigned short fieldTo);

                [id(68), helpstring("method GetRowHeight")]
HRESULT GetRowHeight([in] __int64 record, [out, retval] unsigned
short* pHeight);
                [id(69), helpstring("method SetRowHeight")]
HRESULT SetRowHeight([in] __int64 record, [in] unsigned short
height);

                [id(70), helpstring("method GetAutoRowHeight")]
HRESULT GetAutoRowHeight([in] __int64 record, [out, retval] BOOL*
pVal);
                [id(71), helpstring("method SetAutoRowHeight")]
HRESULT SetAutoRowHeight([in] __int64 recordFrom, [in] __int64
recordTo, [in] BOOL val);

                [id(72), helpstring("method CreateFormatParams")]
HRESULT CreateFormatParams([out, retval] IFormatParams**
ppFormatParams);
                [id(73), helpstring("method GetFieldFormat")]
HRESULT GetFieldFormat([in] unsigned short field, [in]
IFormatParams* pFormatParams);
                [id(74), helpstring("method SetFieldFormat")]
HRESULT SetFieldFormat([in] unsigned short field, [in]
IFormatParams* pFormatParams);

                [id(75), helpstring("method GetRecordFlag")]
HRESULT GetRecordFlag([in] __int64 record, [out, retval] int*
flag);
                [id(76), helpstring("method SetRecordFlag")]
HRESULT SetRecordFlag([in] __int64 recordFrom, [in] __int64
```

```
recordTo, [in] int flag);

                [id(77), helpstring("method SetFilePassword")]
HRESULT SetFilePassword([in] BOOL enable, [in] BSTR cryptMethod,
[in] BSTR oldPassword, [in] BSTR newPassword);

                [id(78), helpstring("method GetFileInfo")]
HRESULT GetFileInfo([in] BSTR path, [in] BYTE type, [out, retval]
BSTR* value); // 1 - size, 2 - mod. date
                [id(79), helpstring("method ReleaseFile")]
HRESULT ReleaseFile([out, retval] BSTR* value);
                [id(80), helpstring("method AttachFile")] HRESULT
AttachFile([in] BSTR path, [out, retval] int* modified);

                [id(81), helpstring("method UpdateTable")]
HRESULT UpdateTable([in] BYTE procCores);
                [propget, id(82), helpstring("property
updateMode")] HRESULT updateMode([out, retval] BSTR* pMode);
                [propput, id(82), helpstring("property
updateMode")] HRESULT updateMode([in] BSTR mode);

                [id(83), helpstring("method MessageBox")] HRESULT
MessageBox([in] BSTR message, [in] BSTR buttons, [in] int button,
[in] BSTR icon, [out, retval] BSTR* retValue);
                [id(84), helpstring("method InputBox")] HRESULT
InputBox([in] BSTR title, [in] BOOL password, [in] BSTR
initValue, [out, retval] BSTR* retValue);
                [id(85), helpstring("method Sleep")] HRESULT
Sleep([in] unsigned long time);

                [id(86), helpstring("method MaximizeAppWindow")]
HRESULT MaximizeAppWindow();
                [id(87), helpstring("method MinimizeAppWindow")]
HRESULT MinimizeAppWindow();
                [id(88), helpstring("method RestoreAppWindow")]
HRESULT RestoreAppWindow();

                [propget, id(89), helpstring("property
lastError")] HRESULT lastError([out, retval] BYTE* pCode);
                [propput, id(89), helpstring("property
lastError")] HRESULT lastError([in] BYTE code);

                [propget, id(90), helpstring("property zip64")]
HRESULT zip64([out, retval] BYTE* pCode);
                [propput, id(90), helpstring("property zip64")]
HRESULT zip64([in] BYTE code);
```

Related Topics

Contacting Technical Support

If you have any questions, comments or suggestions, please visit the support forum or email Citadel5 directly.