fiasco

Nils Bandener

COLLABORATORS			
	TITLE : fiasco		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Nils Bandener	June 25, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

fiasco

1.1 Fiasco.guide

```
                Fiasco  Release 2.2
AmigaGuide documentation
Copyright © 1995-1998 Nils Bandener


                Introduction

                  About this Document

                  Features

                  News

                Getting Started

                  Requirements

                  Installation

                  Starting Fiasco

                  Quick-Start

                Basic Elements of a Database

                  Records
                ~
                  Indices

                  Fields

                  Mask

                  List

                  Editing Modes

                    Record Mode
```

1.2 Introduction

Introduction

Fiasco is a Database for the Amiga. I originally wanted to write a simple Program that could test one's English or Latin vocabulary. I later implemented the ability to define more than two fields (answer and question). The program continued to developed and finally became very similar to a database program. I only needed to make minor changes and there it was! Fiasco is now powerful, many featured program.

Basically there is little difference between Fiasco and other database programs. Although Fiasco does not support hierarchical structures (i. e. a kind of a database in a database), it does support relations. Starting with release 2.0, Fiasco supports list view fields, which may contain several entries for a specified set of fields.

Fiasco also has an ARexx interface that can be used to control Fiasco from other Programs or for assigning ARexx scripts to fields within a Fiasco database.

Indices are used for organizing the record order. The implementation of the sort and filter functions makes also use of them.

Fiasco does not have to load the entire database on startup. Record data is only read from disk when required. This makes the use of databases possible, which have a greater size than the available RAM. To reduce access time, Fiasco caches loaded records in RAM. The RAM usage of Fiasco databases may be controlled by the user.

Fiasco's "mask" is not defined by a graphic file -- it is created using internal images and any non-proportional font. Fiasco provides a number of field types. My personal favorite is the datatypes fieldtype which can be used to display graphics, animations, texts etc. directly in the

mask
window.

The

list window
is a second way to display data. The list window is much like the mask fully configurable. However, you cannot use a list window to modify data.

In order not to lose the overview over your data, you can use Fiasco's

search system
. Its features include search by several fields, wildcards, "blurred" search and search by formula.

Furthermore, Fiasco has sort, filter and count functions. Because all functions are related to the search system, all of these functions are easy to use when you know the search system.

1.3 About this Document

About this Document

You do not have to read the whole manual to work with Fiasco. Many functions of Fiasco have been designed to be as easy and intuitive to use as possible. To avoid any misunderstanding, you should read the section

Quick Start
, though. Furthermore, Fiasco supports online-help. Pressing the help key in a requester or over a menu item will display the appropriate AmigaGuide section of this document.

This manual contains descriptions and tutorials for all Fiasco functions and systems.

Several of the advanced functions of Fiasco require some additional knowledge. In these cases, reading the corresponding sections in this manual is highly recommended.

The chapter

Basic Elements of a Database
describes the principles of databases and how Fiasco uses them. The following chapter

Simple Usage of Fiasco
explains the creation of a simple database and how to work with it.

Advanced Usage of Fiasco

documents the features of
Fiasco which are useful but not neccessary.

While the chapters listed above contain information about functions, which directly control the database, the next chapters explain additional functions that are not directly connected to the database structure. To understand these chapters, you do not need any knowledge from the Advanced Usage of Fiasco or from any other chapters in this section.

Searching in a Database
describes Fiasco's search function and all related functions.
Printing a Database
describes Fiasco's print function and explains, how to create print-outs without using the built-in print function. The chapter
Import and Export
explains --- what else --- the Import and Export function of Fiasco which is useful for sharing data with other programs.

The chapters
Fieldtypes
,
Fiasco's Graphic User Interface
and

Fiasco's ARexx Port
are mainly for reference.

1.4 Features

Features

Fiasco has the following capabilities:

- Dynamic record loading. Fiasco reads the record data only when required. Thus, the use of databases bigger than the available RAM is possible.

- Support of several indices for a database. These indices can be automatically sorted or filtered.

- Several projects may be in RAM at the same time. The number of these projects is only limited by the available RAM.

- Masks can be used like any other GUI.

- Masks and lists automatically adapt to non-proportional fonts.

- Requesters are created with gtlayout.library, which provides full font independence.

- Many fieldtypes: String, Integer, Float, Cycle, Boolean, Slider, Date, Time, Extern, Datatypes and Var String.

- Support of listview fields for String, Integer, Float, Date and Time field types.
- Datatypes fields can be used to display graphics etc. directly in the mask.
- Flexible search function supporting wildcards, burred search and formulas.
- ARexx interface for external control and scripts for fields.
- Freely configurable "user menu", which can be used to invoke CLI and ARexx Programs.
- Very flexible list, which supports hiding and resizing of entries
- Easy relation handling
- Import and export of databases
- Flexible print function

1.5 News

News

Features added in Fiasco 2.2:

- Enhanced replace function: Now supports several fields to be replaced and formulas for calculating the replacement.
- Implemented "CD ROM mode" for databases.
- Field requester now checks, whether a field ID is suitable for use in formulas and warns if not.
- Fiasco can be configured to open a file requester on startup. when Fiasco is started without arguments.
- New program argument: Iconified starts Fiasco without opening the GUI.
- The user may define startup and shutdown ARexx scripts for individual databases and the whole program.
- New ARexx commands:
  ActivateDBWindow
  for activating a database's window,
  MenuControl
  for switching menus off and
  ReadSettings

and

SaveSettings
for reading or saving, resp., of Fiasco's settings.

· New argument for
SetField
and
GetField
: ExtFormat handles extended,
for example localized, formats.

· New argument for
SetField Stem
: CreateListEntries makes it now
possible to completely replace old listview entries.

· New formula operator: numentries(listview) returns the number of entries in a listview field.

· New formula functions: numrecords() returns the number of records in the database and activerecord() returns the number of the active record in the database.

· Now deals more robust with trashed data in database files.

Bugs fixed in Fiasco 2.2:

· Fixed infamous "Error 500" bug.

· While using relations and both relation databases were open, an "adding existing record" error could occur.

· Relation requester would crash if remote database contained user defined formula functions or constants.

· If an ARexx script started by Fiasco was halted with Ctrl-C, datatypes fields would not work any more.

· GetAttr Fields also returned already deleted fields.

· Project/Save As could sometimes have problems with indices.

· Button field shortcuts did not work when no record was active.

· Arguments attribute for button fields did not work with CLI programs.

· ARexx scripts without arguments were started with an empty pair of quotes.

· Under certain circumstances, the import function could hang.

· Because of an uninitialized variable, the sort function could refuse to work under certain circumstances.

- The sort function did not mark a database changed.
- Automatic sorting indices would sort in a record which should be sorted to the end of an index one position before.
- SetField Stem did not work correctly with automatic sorting indices.
- The formula requester did not display constants.
- ActiveIndex Var and Find Var did not work.

Features added in Fiasco 2.11:

- Print elements may now center the data or justify them right.
- New formula operator current() which returns the number of a listview entry, whose value is just beeing calculated by this formula.
- New ARexx commands
  SetConstant
  ,
  GetConstant
  ,
  DeleteConstant
  ,
  RecompileAllFormulas and
  FlushRecords
  .
- Export function now only filters whole control strings.
- A field with popup gadget now uses Shift + shortcut for opening the popup requester.

Bugs fixed in Fiasco 2.11:

- In databases with virtual fields, internal errors could be shown after usage of the search function (or related functions).
- After deleting many records, record access could sometimes work not correctly.
- After Record/Remove All Records and Save Fiasco could get confused.
- An index created with Compare/Sort could be not overwritten by Compare/Sort again.
- Compare/Filter could cause problems when the database was automatically saved.

- While adding records to a database without records but with activated relations, Enforcer hits were produced.
- After overwriting indices with Compare/Sort or Compare/Filter, internal errors could occur while closing the database.
- After a failed Save, Fiasco could crash.
- Deleting of indices could cause a second Standard.fidx to be created.
- The convert field requester allowed to convert fields into slider listviews (!) instead of date listviews.
- Using var string fields could cause Fiasco to run out of signals.
- The ARexx commands SetField and GetField used locale settings.
- Old ARexx find commands only opened the search requester.
- ARexx command ActiveRecord used with relative record movements (Next, Prev) did not work.
- ARexx command Export had wrong argument template.
- Quitting Fiasco from the ARexx port was likely to result in a crash.
- Checkbox gadgets in import/export requesters were not updated correctly.
- User Interface Settings/Automatically activate first field would also activate a field even if it was disabled.

Features added in Fiasco 2.1:

- Support of formulas for calculated field contents and for complex searches.
- New search function that supports searching by several fields and by formulas.
- Improved patterns for searching.
- Completely revised ARexx port that is now Style Guide compliant. Old scripts will continue to work though.
- Fields may be now assigned predefined values that can be selected using a picker button at the right side of the field. This button may be also used to start ARexx scripts.
- Var String fields can be now displayed in the list window. Var String fields from Fiasco 2.0x files are marked as hidden and can be shown using the field requester or List/Show column.
- Delete Field now checks for uses of the field to be deleted.

- Index history keeps track of the order in which the indices were actived. With Database/Previous active index one can go back through the history.
- Special relation mode Read only, that only reads the relation data when a key is changed, but stores the data in the local file and does not keep the data up-to-date.
- Float, date and time fields use formatting settings from locale.library if available.
- ARexx scripts assigned to buttons may now have arguments and open a console window.
- New attribute for listview fields: Select Only allows selection of, but not editing of, entries.
- Float values are now stored in a 64 bit format and are thus more precise.
- The default path of databases and the location of the Fiasco main program may be specified now.
- New file format for preferences. Old format can be still loaded.

Bugs fixed in Fiasco 2.1:

- Importing into an index that does sorting would not cause the index to be sorted again.
- When no records were loaded (e.g. when a database was loaded in mask mode) and then the database was saved, the active index would be saved empty.
- Deleting records in a database that had never been saved before would cause saving corrupt .frec files.
- Import function could recognize a control character even if it was not there.
- Opening two databases with .fdat files and relations could cause alerts.
- After Reveal Project the service window was not updated correctly.

Features added in Fiasco 2.02:

- The window gadgets of Fiasco’s mask window may be now controlled using the options requester.

Bugs fixed in Fiasco 2.02:

- Deleting an record that was added since the last saving could confuse Fiasco.
- Automatic sorting could change the record data of a newly sorted record.
- Saving a database after request in the index requester could make Fiasco to use wrong index files.
- Save Settings could crash when Service Window/Fixed Position was activated.
- Var String fields did not recognize changes when only the length of the content was changed.
- Several minor bug fixes.

Features added in Fiasco 2.01:

- Editor setting supports %s to specify a place where the file name should be substituted. This is particularly useful for GoldED users, which should add the STICKY argument to avoid nasty problems. Example: c:ged "%s" sticky.
- F_SetFieldCont has been made much faster.

Bugs fixed in Fiasco 2.01:

- Automatic sorting could sort incorrectly and cause enforcer hits.
- Automatic sorting did not work well with Descending.
- If a record was added to a database which did not contain any records and previoulsly a field had been deleted, enforcer hits would be caused.
- Closing a database which is remote database of a relation in a database that is also open could cause enforcer hits or crashes.
- Adding relations with the relation requester did not work on Amigas without memory after $0100 0000.
- If a in the list not displayable field was converted into a displayable field or vice versa, the list window was not correctly

re-layouted.

- Print function cut contents of var string fields after an empty line and could eat single characters.

- Sometimes Reorganize would not delete all unused records.

- Deleting of records, which have been added after the last saving of a database, would cause garbage records to be created in the database file.

- Record argument of F_AddListEntry did not work correctly.

- F_MarkMatch did not work reliable.

- Several minor bug fixes.

1.6 Getting Started

Getting Started

Requirements

Installation

Starting Fiasco

Quick Start

1.7 Requirements

Requirements

The minimum requirements are an Amiga with OS 2.04 (37.175) and 1 MB RAM. Recommended configuration: Amiga with OS 3.x (39.x or higher), 68020 Processor, 2 MB RAM and a hard disk.

Fiasco uses the gtlayout.library by Olaf Barthel for its GUI. Var string fields are implemented using the textfield.gadget by Mark Thomas. The files are included in the archive of Fiasco.

Fiasco requires the ACTION_SET_FILE_SIZE packet, which was introduced with Amiga OS 2.0. Both the 2.0 ROM filesystem and the 2.0 RAM handler support this packet, however, some handlers may not support it. Thus, you cannot save Fiasco projects to these handlers. Reading should be always possible.

Possibly also due to this, Fiasco does not work with Diskexpander and XFH. You are able to read databases, but saving does not work. If you use

Diskexpander, errors will be produced. Take care if you use XFH: Saving a database to a XFH handler may crash the system.

The memory pool functions of Amiga OS 3.0 and Amiga OS 3.1 do not free unused puddles until the pool is deleted. Use SetPatch 40.16 (already included in WB 40.42) to fix this. If you use Amiga OS 2.0 or Amiga OS 2.1 you do not have to worry about that.

Features and required OS-Versions:

Localization: Amiga OS 2.1 (38.x)
Screenmode-Requester: Amiga OS 2.1 (38.x)
Online-Help: Amiga OS 3.0 (39.x) or amigaguide.library v34
Datatypes-Fields: Amiga OS 3.0 (39.x)

1.8 Installation

Installation

If you have the Installer program simply double-click on the install icon of your preferred language in the install drawer. You then will be given step-by-step instructions.

If you don't own the Installer, you may simply drag the Fiasco drawer somewhere you want. You may copy the catalogs to locale:catalogs, but they will work at this place, too. You may delete the unused languages in "Documentation" and drag the remaining files in the parent drawers. The files in "Development" and "Install" are not required for normal operation of Fiasco and may be deleted, too. With this configuration, Fiasco will run. If you have a 68020 processor or better, you should delete the file gtlayout.library in the main directory of Fiasco. Then, you should copy the gtlayout.library from the directory libs/68020 into the main directory. If you want to make the gtlayout.library accessible for all programs, you should copy it into the libs: directory.

1.9 Starting Fiasco

Starting Fiasco

You can start Fiasco from Workbench or Shell. The simplest way to start Fiasco from Workbench is to double click on the Fiasco icon. From shell, you have to change the directory to Fiasco's directory and then type Fiasco. These startup arguments are supported by Fiasco:

From: The databases to be loaded on startup. You may specify several databases. Only supported from shell. If started from Workbench, there is a different scheme: If Fiasco was started using a project icon of a Fiasco database, Fiasco will load the database of the icon on startup.

You may also select multiple project icons and start Fiasco with the program icon. In that case, Fiasco will load all selected projects.

Config: After this keyword you may specify a Fiasco configuration file to be used by Fiasco. Default: env:fiasco.prefs. Example: CONFIG=env:fiasco_24bit.prefs. Supported both as shell argument and as Workbench tool type.

Iconfied: If you specify ICONIFIED, Fiasco will not open its GUI but start in iconified state. If the Workbench is running, an Fiasco AppIcon will be displayed on it, with which you may open Fiasco's GUI. You may also start Fiasco again or use the
ARexx port
to open the GUI.
Supported both as shell argument and as Workbench tool type. New in Fiasco 2.2.

Before you start Fiasco, you have to ensure, that Fiasco has enough stack. For now, 8192 Bytes is a good value. When you start Fiasco from Workbench, you can set the stack size in the information requester for Fiasco's program icon or the project icons. In a shell, you have to use the Stack command to increase the stack.

If you try to start Fiasco, while it is already running, Fiasco will recognize itself and not start again. If you have specified databases to be opened by Fiasco when you have started it the second time, these will be opened by the already running Fiasco process. If you have not specified any databases, Fiasco will open a new window.

1.10 Quick Start

Quick Start

These are the most important things, which you have to know while working with Fiasco:

· The program may be started over the program or project icon

· There are two working-modes: In the record mode you may edit records, search for them etc. The mask mode allows you to add or modify fields. You may control the modes using the menuitems
Control/Record Mode
and

Control/Mask Mode
· The
service window
makes certain operations easier, especially if you are not familiar with menu shortcuts. You may open it over

Control/Service Window
. Attention: The functions of the gadgets differ in the different modes.

· A
list
, which can be opened with
Control/List Window
, may be changed
by clicking in the titles of the list. Clicking one time on a column title activates the respective column. Using the menu List you can now do several things with this column. If you click at the right border of a title, you can change the size of the column. The other space can be used to drag the column to any other place in the list.

· Certain project options may be changed with the menuitem

Database/Options
· The sort and filter functions use indices. When you sort or ↩ filter a
database, a new index is created and activated. To activate the old index again, use the menu item
Database/Previous active index
. You
have complete control over the indices with the menu item

Database/Indices
.

· Record/Delete Record only removes the record from the index. To get finally rid of it, you have to call
Datenbank/Reorganize
.

· If you have any problems, you may press the help key while browsing through the menu.

1.11 Basic elements of a Database

Basic elements of a Database

Basically, most databases are analogous to a card file. A Fiasco database project consists of two components: First, there is the data which is divided into records. Second, there is the mask which defines the structure of the data.

The following pages describe the basics of databases in general and the basic Fiasco-specific principles.

Records

Indices

Fields

Mask

List

1.12 Records

Records

Records are the file cards of a database. That means a record is a collection of several data items for one main item (e.g. for a person name, address, etc.). In Fiasco the
mask
is only able to display one record at a time. The
list
displays several records as lines.
Indices
give you more control over records. One database may have several indices. Each index determines, which record is actually displayed, and how the records are ordered. Thus, a database may contain records, which are not visible, because they are not in the active index.

1.13 Indices

Indices

Indices are new for Fiasco 2.0. Indices are used to control the order of the records and whether a record is actually displayed or not. There may be several indices per database.

If you imagine the records as file cards, indices are additional lists which say "File card 16 is at position 1, file card 5 at position 2", etc. The so-called record numbers used by Fiasco are actually the positions of the index entries. Thus, with each additional index, the record number of a particular record may change.

The new
sort
and
filter
functions also use indices. These functions simply create a new index which is sorted or filtered as wanted. Furthermore, Fiasco allows you to automatically keep an index sorted or filtered.

Normally, records are only added to the active index. If you want, that another index also gets the new record, you will have to activate the option Automatically add new Records in the
Edit Index requester
.

```
    More information on indices can be found in the section
                Using Indices
                .
```

1.14 Fields

```
                Fields

Fields define what data may be stored. In Fiasco the fields are defined
in the mask. Fields are the basic elements of the mask and the list.
Fiasco supports several types of fields. More information on the field
types and their features are located in the
                field types
                 chapter.
```

1.15 Mask

```
                Mask

The mask is the way to display data, which Fiasco uses most of the time.
A mask, in contrast to a
                list
                , can display only one record. The advantage
of the mask is the clarity of the display. In the card file example, the
mask defines the structure of the file cards.  The mask consists of
fields of which there are several types and images.

    If you use normal Amiga programs, you would call these fields
"gadgets".

    Internally, Fiasco uses gadgets as fields, which optically conform to
the GadTools standard.

    Fiasco masks adjust automatically to any non-proportional font. Topaz
and courier are examples of non-proportional fonts.

    To create a mask in Fiasco you have to be in the mask mode. You may
change the position of existing fields using the mouse or make other
changes with the Field menu.  More on this topic
                here
                .

    See the
                mask window section
                 for more information about the user
interface features of the mask window.
```

1.16 List

List

Opposed to the mask, a list can display several records at once. In a list, the records are represented by lines, while the fields of a record are represented by columns. Thus, there is much less space for one record in a list and the layout of a list is much more restricted than the layout of a mask. On the other hand, the list can be used to get a quick overview of all records in the database or the active index, respectively. See the section
list window
for a detailed explanation
of the specific features of Fiasco's list window.

1.17 Editing Modes in Fiasco

Editing Modes in Fiasco

Fiasco divides it's operation into modes. If you want to make changes in the mask, you have to be in the mask mode. If you want to make changes in the records, you have to be in the record mode.

Record Mode

Mask Mode
The section
Mask Window
contains descriptions of the features of
the mask window in both modes.

1.18 Record Mode

Record Mode

You may add, delete or edit records in this mode. It may be activated with
Control/Record Mode
. When the record mode is active, "tape deck" gadgets are shown in the service windowthe status gadget displays normally the number of the active record and the number of all records (for instance: 78 / 92).

1.19 Mask Mode

Mask Mode

This mode give you the ability to edit the mask, that is, you may create

new fields, delete some or change their position or attributes. Relations may also created and changed here. This mode may be activated with

Control/Mask Mode
. When the mask mode is active a field type cycle gadget in shown in the service window and the status gadget displays normally the coordinates of the cursor in the mask (for instance: X: 10, Y: 5).

1.20 Simple Usage of Fiasco Databases

Simple Usage of Fiasco Databases

And now to actual use: If you want to create a database in Fiasco you will have to create the mask at first and then the records. Fiasco allows you in most aspects to create a database in an intuitive way.

Working in the Mask

Working with Records

1.21 Working in the Mask

Working in the Mask

You have to activate the mask mode before you can create a mask (

Control/Mask Mode
), whereupon a cursor will appear in the mask. You can use the mouse or the cursor keys to choose the location of the next operation in the mask. Before creating a new field you have to choose the type of the new field. You can use either the Field/Type
menu
or the
cycle gadget in the service window to choose the field type.

You then may use
Field/Add Field
to create a new field. This will
open the
field requester
. The gadgets in the requester depend on the supported attributes of the active field type. They are described in the

type documentation
for each field. It is not sufficient to click on Ok without any other action; you must specify certain attributes, such as the ID. Fiasco won't close the requester if it contains any invalid settings. The field will appear in the mask after you close the requester.

You may change all attributes later except the fieldtype (Fiasco

provides another function for that). A field's position may be changed by dragging it with the mouse. You may cancel dragging while you are dragging still it by pressing the right mouse button. You may also select several fields by pressing Shift while selecting the fields. The fields will be deselected when you select a field without Shift.

The field requester may be opened by double clicking on the field or by choosing

Field/Edit active Field
after you have activated the field to edit. The menuitem

Field/Edit named Field
displays a list of all fields.

When you have selected one field, Edit named Field will open the field requester for the selected field. This is useful for editing hidden fields. You should take care if you want to change the field ID. Other Fiasco projects or ARexx scripts which try to access this field won't find it after the change. If you change the value max chars of string, extern or datatypes fields, you will be informed that you could loose data.

Field/Delete Field
allows you to delete Fields. Attention: If Security-Requester in the user interface settings is not active, all data in this field will be freed immediately. Any existing data on disk will be also erased when the project is saved. If the field is used by any other Fiasco system, such as indices or formulas, you will be warned before deleting that field regardless of the Security-Requester setting.

You may specify further parameters for the current project, such as

mask stretching
, name of the author, etc. in the options requester
.

You may also

group
certain fields. If a grouped field is activated, all other fields in the group will be also activated. Furthermore, certain field types, such as the bar and the listview types, may share their visual display when they are grouped.

Field/Edit Relations
works similarly to Edit Field
. With this menuitem you are able to control relations
of this field.

When you have completed the mask you may return to record mode. You are now ready to create records.

1.22 Working with Records

```
                Working with Records

You may create records for data storage in any mask containing fields.
The simplest way to create a record is to select
                Record/Add Record
                 or its
equivalent Add in the service window. This creates, as the name implies,
a record and activates it. The fields in the record will contain the
values that have been assigned in the mask mode. You may now activate a
field using the mouse and edit its contents. Record/Duplicate Record
provides another way of creating records. This function creates a record,
which is an exact clone of the record, which was previously active. All
init cont-attributes will be ignored.

    You may also cut or copy a record and paste it at another position.
Please note that cutting and pasting is just like deleting and creating a
new record, not just moving. Thus, if other
                indices
                 use the record, which
you have cut, the contents of the record will remain constant, even if
you paste the record somewhere else and edit it.

    If you no longer need a record you may delete it using

                Record/Delete Record
                 or Delete in the service window. If you have
selected Security-Requesters in the user interface settings, you will be
asked for confirmation before the record is deleted.

    Please note that only the entry in the active index is deleted. The
data of the records will remain in the database. To get rid of it
permanently, you will have to use Database/Reorganize.

    You may use the menu, the service window, the cursor keys or a list
window to view the records you have created.  I believe that the use of
GUI is intuitive, therefore, I will only explain the cursor keys. The
up-key activates the previous record. The down-key activates the next
record. The order corresponds to the concept of a list window. The cursor
keys combined with the Ctrl key activate the first or the last record
respectively.
```

1.23 Advanced Usage of Fiasco Databases

```
                Advanced Usage of Fiasco Databases

The information described in this chapter is not necessary to work with
Fiasco. However, it often may be helpful.


                Converting Fields
```

Groups

Using Indices

Using Marks

Relations

Virtual Fields

1.24 Converting Fields

Converting Fields

As your project develops you may decide that you want to change one or more of the field types. For instance, the contents of a field may have developed in a direction other than the one you originally intended. In that case, the convert function will be useful. This function is also helpful if you have imported a file. After a file is imported all fields are string fields. You must be in mask mode to open the convert requester. Activate the field you want to convert and select Field/Convert Field. The convert requester displays the ID of the field, the current field type and the field types to which this field may be converted. If you select Alternative format, the convert function may convert the data to another, often more abstract format. Not all field types support this option. If you select the the new type and proceed with Ok, the field will be converted. Note that Fiasco will not warn about the possible loss of data. If the new field type requires additional attributes (e.g., the extern fieldtype needs a program), the field requester will open. Other attributes will use default values. If you convert a field and then convert it back to its original type it won't retain the original attributes.

If a field is used by relations, formulas or indices (for sorting or filtering), these uses will be removed. You will be warned before that is done.

Information about the results of a field type conversion can be found in the
field documentation
. Text, button and bar fields cannot be converted. In other cases, converting from one field type to another does not make much sense (e.g., boolean to datatypes).

If you convert a listview field to a non-listview field or vice-versa, the character '|' will be used to separate the entries of the listview.

1.25 Groups

Groups

Generally, a Fiasco group is a group of fields, which have been selected by the user. This only sounds less than useful. The advantages of groups are the additional characteristics grouped fields have. First: In mask mode grouped fields stick together. That means, if you select one grouped field, the whole group will be selected. Thus, if you want to drag a grouped field, the whole group will be dragged. All other functions which affect all selected fields will also affect the whole group. The second advantage of groups is the joined display (and function) of grouped fields. Some fieldtypes share their images with other fields next to them in the group. This is supported by these fieldtypes:

.
Bar fields
.
Listview fields
How these fields share their images is described in the

field type documentation
.

To create a group, select all fields to be in the group and call

Field/Create Group
. To resolve a group, select it and call

Field/Resolve Group
.

If you try to group an already existent group with other fields or other groups, you will get a new big group of all selected fields. Resolving that group will resolve all groups, which have been grouped in the group. Thus, all fields in the group will get free.

1.26 Using Indices

Using Indices

The concept and simple use of indices is described in the sections
Indices
and
Filters
. However, the index system is much more powerful. It allows you to automatically sort, filter and update indices. The GUI of the index system is the
index requester
, which can be opened using

Project/Indices
. You can activate an index here by selecting it and clicking on Ok. You can also create new indices and edit or delete

existing ones.

There may be no indices before the database has been saved or if the database is in Fiasco 1.x format. A standard index will be created when the database is saved the first time.

Creating a new Index

When you create a new index using the New button the
new index requester
will open.

The topmost gadget takes a name for the index.

The checkbox Automatically add new Records controls whether you want Fiasco to automatically add new records to the index. If the checkbox is not active, records that have been added to the database while another index was active will not be added to this index. If the checkbox is active, Fiasco will automatically add the records to the index, which have been added to the database, while another index was active. If the sort option (see below) is active, the new record will be automatically sorted in the index. If the filter option (also see below) is active, it will be tested using the filter and only added if it matches the filter.

Click on the appropriate buttons to edit a filter for the index or the sorting. The sorting and filtering functions will be activated only if you have selected the checkboxes beneath the buttons. Sorting will be always automatically done when you change or add a record. The filter will be used only during the initial creation of the index and when a record has been added in another index and Automatically add new Records is active. The filter will not be used when a record is added while the index is active or when a record is changed.

The listview gadget allows you to select an existing index as the model for the new index. Only records that are in the selected index will be examined for the new index. If you do not use sorting, the order of the records will be carried over. If you use neither sorting nor filtering, a copy of the selected index will be created. By selecting an index, which already uses a filter, you can create a filter, which uses several conditions logically combined with "and". The special entry «No Index» corresponds to all records in the database, even those not used in any index.

The new index will be created after you have exited both the new index requester and the index requester with the Ok gadgets.

Editing an Index

When you Edit an index, the
edit index requester
will appear. This is the
new index requester without the index name and listview gadgets. You may change filtering, sorting or the Automatically add new Records option. The changes will be executed after you have exited both requesters with Ok.

Deleting an Index

The Delete gadget removes the selected index. However, you cannot remove the last index; Fiasco requires at least one index.

1.27 The Index History

The Index History

The index history is a system that remembers the order in which indices are activated by the user. For example, if you create a new index with the filter function, this new index will be activated. However, Fiasco keeps track of the index that was active before this one. Thus, you can simply switch off the filter by going a step back in the index history. This can be done with the menuitem Database/Previous active index. To activate the filter again, you may use Database/Next active index to go that step forward again.

1.28 Using Marks

Using Marks

Marks can be useful in advanced database use. A mark is simply a record's flag that may be toggled on or off, that is, a record is either marked or unmarked. Marks could be simulated using boolean or other fields, but the marking feature of Fiasco provides some additional advantages over that approach. First of all, a marked record can be easily discovered in the list because it is displayed in a highlighted state. If a marked record is active an "M" will be displayed to the right of the service window's status gadget, therefore, marked records can only recognized in a mask if the service window is open. Marks can be set using
Record/Mark Record
and
cleared using
Record/Unmark Record
. Use
Record/Unmark all Records
to
clear all marks in a project. To set all marks, use

Record/Mark all Records
. Use
Record/Toggle all Marks
to unmark all marked
records and to mark all unmarked records.

Compare/Mark
opens a search requester that can be used to mark all

records that match a given pattern.

Marks are saved with the record data and thus, they are independent from indices.

1.29 Relations

Relations

Relations are fields that store their contents in another database rather than in the database of the relations. An additional field is required that contains a key used to identify the record from which the data should be taken.

This mechanism prevents the situation, that in many different projects the same data are stored; it therefore saves disk space. Furthermore, you only have to change the contents of one field in one of the databases -- all other corresponding fields will also recognize that change.

Using Fiasco, you have define relations in the database, which will actually read the data from another database. The database, in which you have defined the relations will be referenced with the word "local" in this document. The database from which the data are read, will be called "remote".

Relation Types

Creating Relations

Relation checklist

1.30 Relation Types

Relation Types

Fiasco supports these relation types:

1:N

This is the simplest relation type. A 1:N relation reads the data of any fieldtype in the remote database using a key, which has to be of a simple type, i.e. it must not be a listview type. In the remote database, the key has to be unique. The read data are directly copied into the appropriate field in the local database.

N:Sum

This is not really a relation, because the data cannot be written back. A N:Sum relation reads the data of simple fieldtypes and tries to calculate the sum of all fields in the records with matching keys. Thus, the key needs not to be unique in the remote database.

N:L

A N:L relation reads the the data of simple fieldtypes in the remote database. For each record in the remote database, in which a matching key is found, Fiasco will add an entry in the local listview field, and copy the data into it. Thus, the key needs not to be unique in the remote database.

1:L

A 1:L relation is the only relation type, which requires the local key in a listview field. The remote key has to be in field of simple type, though. This type is very similar to the 1:N type, however, you may specify several local keys using the capabilities of the

listview field type
. Fiasco will search the real data in the remote database for each key in the listview and create entries in the local real data listview with the found data. Because there is no thing like a listview in a listview, the remote real data has to be simple.

Only Read

Only Read is a modifier, that can be applied to the relation types 1:N and 1:L. If Only Read is active, the relation system is only used to enhance editing possibilities. Thus, you can copy data from a remote database using an unique key. However, the data are stored in the file of the local database and are not kept any longer up-to-date according to the remote database.

1.31 Creating Relations

Creating Relations

This section describes creating of relations in Fiasco databases. It was originally written to describe the creation of relations of the type 1:N. However, you can also create the other relation types according to these instructions when you also keep the conditions for the relation types described in the previous section in mind. To use relations in Fiasco you have to create a project, which will be the data source for another project. You have to create at least two fields in the remote project, one for the data and one for the key. The field for the key should be an integer field. This is the fastest method. However, it is possible to use any other field type as a keys.

You may use the special field attribute Unique Key, if you want to automatically get an unique key whenever you create a new record. Note that the key is only created when you create a new record. If you activate this attribute later the already existing entries will keep their old value. If you change the contents of such a field the change will occur without any checking.

It is up to you to choose the type of the second field. If you create fields, which store strings of a designated length (string, extern and datatypes), you should remember the Max Chars value because you also have to use this same value in the second project.

If you want to see any consequences of activating the relation, you should create a few records with some content at first.

Now it is time to save the project and create a new one.

The second project also must contain two fields that have to match in typeMax CharsThe key field
should not use unique key, because you should freely decide which key you want to use.

Before you activate the relation the project should be saved in the directory in which the other project has been saved in order to be able to use relative rather than absolute paths.

Now you can open the
relation requester
for the field that is not supposed to contain the key, but the real data (
Field/Edit Relations
).
Set the relation type in the topmost cycle gadget. To start, you should select the local key in the listview in the upper left edge of the window. After that you should select the other project with the file requester gadget at the bottom of the requester. Now you can select the remote key and real fields. Proceed with Ok. If everything works correctly the requester will be closed and the relations will be loaded. Otherwise, a requester will inform you of any failure.

A
relation checklist
, which contains the information in a compressed form, is also available.

1.32 Virtual Fields

Virtual Fields

The data of virtual fields are not saved on disk; their data are calculated while loading the project. If you want to make a field virtual you should activate the Virtual option in the field requester. Fiasco uses the formula of a field for calculating these data. Whenever a record is read, the contents of the virtual fields in it are calculated using

the formula. More about formulas can be found
here
.

It is also possible to use ARexx scripts for virtual fields. However, for speed and for technical reasons, the use of ARexx scripts for this purpose is discouraged. Therefore, this use is no longer documented. If you want to know more about ARexx and virtual fields, refer to the release 2.0x versions of this document.

1.33 Searching in a Database

Searching in a Database

Fiasco's search function allows you to search for specific data in a database. There are two ways to search for data: You may specify search patterns for one or more fields which must all match the field contents. Or you may specify a formula that can do complex investigations on the contents of a record.

The Search Requester

The main interface to the Fiasco search function is the search requester. It may be opened with the menu item
Compare/Find
. Furthermore, the
filter, count, replace and mark functions or menu items also use this requester.

To select, whether you want to search with specifications of field contents or with formulas, use the Mode cycle gadget in the upper part of the requester.

If you use the field mode, the requester displays two listview gadgets: Fields displays all the fields in the active database. Clicking on one of the fields causes it to be displayed in the listview with the title Search for. Below it you can enter the search pattern for that field. To remove the field from that list, use the Delete gadget.

If you search by Formula, the window displays only one string gadget for the formula. The picker gadget of the right side can be used to open the
formula requester
.

The gadgets at the bottom of the requester start the search. The record will be displayed if a matching record is found. You can use the menu items
Compare/Find next
and
Compare/Find previous
to continue your
search.

```
    More about the search requester can be found
                here
                .


                Searching by Fields

                Patterns

                Wildcards

                Wildcards for Numbers

                Blurred Search

                Searching by Formulas

                Search Informations

                Searching with ARexx

                Count function

                Replace function

                Filter function
```

1.34 Searching by Fields

```
                Searching by Fields

If you use the search mode By Fields, you must specify one or more fields
to be examined. For each specified field, you have to specify a

                search pattern
                . Fiasco considers a record to be a match only if the field
contents of all specified fields match the specified patterns.
```

1.35 Patterns

```
                Patterns

Search patterns are the data that are compared to the field contents. For
the individual field types there are different rules for pattern
specification:

String, Var String, Extern, Datatypes: You have to specify the string
   itself. Furthermore, you may use
                wildcards
```

```
                 and
                blurred search
                .

Integer, Slider, Float: You have to specify the number itself.
   Furthermore, you may use
                wildcards for numbers
                .

Boolean: true or 1 specify a checked field, false or 0 specify an
   unchecked field.

Cycle: You have to specify the label name to search for, or the number of
   the label (counting from zero).

Date, Time: You have to specify the date or the time.

The search requester has an additional feature that makes the
specification of patterns for fields easier. With the picker gadget at
the right side of the pattern string gadget, you may open a requester
that displays a list of possible patterns for the field. If you chose one
and click on Ok it will be copied to the pattern gadget. The list of the
possible values again depends on the field type:


String, Integer, Float, Extern, Datat., Date, Time, Var String: The
   predefined values.

Slider: The maximum and minimum values for the slider.

Boolean: true and false.

Cycle: All labels of the cycle field.

Please also note that you cannot use the built-in search function to
search for the contents of files that are specified in datatypes or
extern fields. You can search for the file name only.

    The patterns supported by one fieldtype are also documented in the

                fieldtypes documentation
                .



                Wildcards

                Wildcards for Numbers

                Blurred Search
```

1.36 Wildcards

```
                Wildcards

In addition to plain patterns you may use wildcards. String, Var String,
Extern and Datatypes fields support the use of wildcards as used by
AmigaDOS. These wildcards are supported:


?    A single unknown character.
#    Matches the following expression zero or more times.
(ab|cd)    Matches any one of the items seperated by | .
~    Negates the following expression.
[abc]    Character class: matches any of the characters in the class.
[~bc]    Matches any of the characters not in the class.
[a-z]    Character range.
%    Matches zero characters.
*    Synonym for #?. Must be turned on in the
                database settings
                .



    If you want to use the wildcard characters as normal characters in
the string, you have to type a single quoute (') before the character to
escape it.

    If you use
                blurred search
                , only the patterns ? and #? are available.


Examples:

?iasco would match Aiasco, Biasco, Ciasco, 1iasco, etc. ???? would match
entries, which are 4 characters long. #? stands for an unknown number of
unknown characters. A#?,  for example, would match Amiga, Africa, A or
ABCD. ?#? searches for all non-empty entries.
```

1.37 Wildcards for numbers

```
Wildcards for numbers

Integer, slider and float fields support the following wildcards: >, <,
>=, <=, !=. The argument has to be given after the wildcard. > only
search for numbers greater than x, >= only for numbers greater or equal
x, < only for numbers less than x, <= only for numbers less or equal x.
!= searches only for numbers not equal x. There is not a pattern like ==
(equal), because this is represented by the number itself.
```

1.38 Blurred Search

```
                Blurred Search

"Blurred" search  allows you to search for entries that are similar to a
pattern. This enables you to search for entries even if you don't know
the exact spelling. The tolerance of the function may be set by "factor".
0 matches only entries that are exactly equal. 100 matches nearly all
entries. This feature can only be used with the field types that contain
strings, i.e. string, var string, extern and datatypes fields.

    If you use blurred search, only the patterns ? and #? are available.

    The
                count function
                 is very suitable for experiments with "blurred"
search.
```

1.39 Searching by Formulas

```
                Searching by Formulas

Even though the GUI for the search mode By Formula looks simpler than the
GUI of the other mode, this search mode is far more powerful. The formula
specified in the string gadget will be used to examine each record. A
record is considered matching whenever the formula results in a true
value. In Fiasco formulas this is any non-zero value. More on Fiasco
formulas can be found   in
                this section
                .


Examples:

size > 5
searches for all records in which the field size has a value greater
than 5.

size >= 5 && size <= 10
searches for all records in which size has a value ranging from 5 to 10
inclusive.

important && stricmp(name, "Meier") == 0
searches for all records in which the boolean field important is checked
and the field name contains the string Meier.

datediff(currentdate(), date) <= 10 && datediff(currentdate(), date) >= 0
searches for all records in which the field date contains a date that is
within the next ten days.
```

1.40 Search Information

Search Information

Search info can be used to temporarily save search criteria. In the search requester, the Name gadget can be used to specify a name for the search info. The search criteria will be stored when you hit any of the positive response buttons at the bottom of the search requester. To get the stored search criteria back, you have to click on the picker button of the Name gadget and select the desired search info. The criteria will be copied to the search requester. When you enter a name into search info, any pre-existing name will be overwritten. Search info data will remain valid until the database is closed.

1.41 Searching with ARexx

Searching with ARexx

You can also use
Fiasco's ARexx port
to search a database. The ARexx commands for searching make use of
search info
to keep track of search criteria.

Thus, you must have a search info before you can start searching. This search info may be created with

NewSearchInfo Name/K,Fields/S,Formula/K,Var/K
. If you do not specify a Name, Fiasco will chose a name that does not exist yet and will return it in Result. This name has to be specified in the arguments of all following search commands that refer to this search info.

If you want to search by formula, you must specify the formula for the search in the arguments for NewSearchInfo.

If you want to search by fields you have to use the command

SetSearchField SearchInfo/K/A,FieldID/K/A,Blur/K/N,Pattern/K/A/F
to specify a field to be searched for. You may call that command several times to specify several fields.

Now you can use the command

Find SearchInfo/K/A,Record/K/N,Reverse/S,All=Stem/K,Var/K
to search for matching records. You may use the Record argument to search with each call for the next matching record. This argument specifies the record after which the search will start. For example, if you specify Record 0, the search will start at the first record. If a matching record is found, its number will be returned in Result. To continue the search you can call Find with the record number of the found record. In contrast to the GUI search function, found records are not activated.

The other method involves the All argument. Fiasco will search all records and store the numbers of the found records in the stem variable specified after the All argument. The number of found records will be stored in stem.count. Please do not use this method if a very large number of matches is possible; it would be too time and memory intensive.

1.42 Count

Count

Compare/Count
opens a requester similar to the search requester. As in the search requester you have to specify pattern, field and tolerance. If you select Ok the matches will be counted. This way you can get experience with the blurred search feature.

1.43 Replace

Replace

Compare/Replace
is the function that allows you to replace certain values with others. You have to specify as normal the criteria for the search here. Additionally, you have to define at least one field in which a new value will be inserted whenever a record matching the search criteria is found. This replacement may be either a fixed value (select Replace by value) or the result of a Fiasco formula (Replace by formula result.

By using the second mode, you are able to apply updates or corrections to your data. An example for this is a change of a phone area code or some general change of an area code system (which happened recently in Italy). When the area code 0123 is changed to 0456, you have to set the search criteria for the phone number to 0123#?. Then set the replace formula for phone number to strcat("0456", strmid(test, 5, -1)). This formula truncates the first four characters of the phone number and replaces them by 0456. As only phone numbers starting with 0123 are affected by this operation, the codes will be simply exchanged.

If you select the Confirm gadget you will be asked if you really want to replace the value for each record. The record will be displayed while you are asked.

Attention: You can quickly destroy important data with a bad pattern (for example: #?)!!!

1.44 Filter

Filter

Fiasco's filter allows you to display only those records that match a pattern. With Compare/Filter you may open the
filter requester
which has
the same structure as the search requester. If you select Ok only those records that match the specified pattern will be displayed.

You may browse through the records with
Record/Next
and

Record/Previous
. The list also displays only matching records.

A filter is simply an index. Therefore, you may specify the name of the index in the filter requester.

To switch the filter off, you may use the index history. The menu item Database/Previous active index will activate the index that was active before the filter was active.

If you create a new filter while another filter is still active (i. e. you did not select the old index in the index requester), only the records in the old filter will be examined. Thus, you can create filters with two conditions logically combined using "and".

If you create new records while a filter is active the records will be displayed whether or not they match the filter pattern. If you change the contents of an existing record it also will be displayed. If you want to update the filter, you have to select the old index and then call the filter requester and select Ok.

For more advanced use of the filter see the section
Using Indices
.

1.45 Printing a Database

Printing a Database

You can create a print-out of a Fiasco database in several ways. The internal print function is the easiest to use. To increase the quality of the print-outs, you may combine TeX with the print function. If you want to create a print-out that can't be created with the print function, you may use an ARexx script.

Internal Print Function

Printing with TeX

Printing with ARexx

1.46 Internal Print Function

Internal Print Function

The menuitem Project/Print opens the print window of Fiasco. This window is similar to a Fiasco project window in mask mode. It contains elements which can be arranged with the mouse. In the final print-out all records will be laid out that way. When you open the print window Fiasco tries to open a file containing the standard print mask for the project. The file name for such files is Project Name.fpr. Project Name is the file name of the project without .fdb. If the file is not found, Fiasco lays-out the print mask according to the real mask.

To print the database as a list, you should select the menuitem Project/Get from list. This will layout the print mask according to the real list. You may use Project/Get from mask to get the mask layout.

You simply have to select Project/Print to print the project with this layout.

Print Mask files

1.47 The Print Mask

The Print Mask

The print mask has three parts: The head, the body and the foot. The head will be printed before any other data. The body will be printed for each record. It may contain references to project fields. These references will be substituted by the field contents of the records while printing. The foot will be printed last. The print window displays only one of these parts at a time. To change the displayed part, use the Control menu.

The print window can be handled much like the project window in mask mode. To create an element (comparable to fields in the project mask), select a type with Element/Type and select Element/Add or press Return. Depending on the type, a requester will appear which gives you some options for the fields. Fiasco supports three element types:

- Field
- Text
- Formfeed

Field elements are usable only in the print body. They can be used to display the field contents in the print-out. The requester for field

elements contains gadgets to select the field, to set the width, to set print styles like bold, italic or underlined and to activate clipping. Clipping can be used to control whether or not an entry may get wider than the specified width. If clipping is active, every entry which is longer than the width will be clipped to fit in that width. If clipping is not active, the following entries will be shifted.

Text elements are similar to text fields in the mask. They serve to put static text in the print mask. They support print styles like bold, italic and underlined. Text elements are the most important elements in the head and foot parts.

Formfeed elements terminate the page. That means that the data after a formfeed will be printed on a new page. Formfeed elements have no editable options and thus no requester opens after adding such an element. Edit element also is not usable for these elements. Because of the special meaning, the width of formfeed elements is "infinite". Formfeed elements appear as a horizontal line in the mask.

1.48 Print Mask Files

Print Mask Files

Project/Save and Project/Save as in the print window create files which contain the print mask structure. These files can be reopened to restore a particular structure. If you have deleted a field in the database or if you have changed its ID, the print mask file may contain references to "nothing". When you open it Fiasco will try to get these references back. Fiasco uses a requester for that purpose which shows the field ID that was not found and a list of all fields in the current project. If you select one and click on Ok the reference will be changed to the selected field. If you Cancel the requester the element will be deleted. You can easily adopt print masks to other projects this way. Simply load the database, open the print window and load the print mask. Now you can change all elements to the matching fields in the new database.

Besides the layout print mask, files contain the settings made in the

print options requester
.

1.49 Printing with TeX

Printing with TeX

You can use TeX to create high-quality print-outs of Fiasco projects. TeX is a kind of programming language, originally developed by Donald E. Knuth, which can be used to create printed documents. PasTeX is a freely distributable TeX implementation for the Amiga that can be found for example on the Meeting Pearls III CD-ROM. The print function of Fiasco supports TeX using
ARexx

.

    If you select Print with ARexx in the
print options requester
, the
function of the Print menuitem of the print window is changed: After creating the print-out, the ARexx script ARexx/ARexxPrint.rexx is called with the name of the created file as its argument. This script should call TeX to compile and print the file. Because of that you must not write the file to PRT:. You should set Print to in the print options requester to a temporary file, for example T:FiascoPrintOut.tex. The script should look like this:

```
/* ARexxPrint.rexx
 * For use with PasTeX
 */

/* Parse arguments
 */
Parse Arg File

Address Command

File = strip(File,,'"')

/*  Call virtex
 */
'virtex' '"' || File || '"'

/*  Create name of dvi file
 */
dotpos = lastpos(".", File)

if dotpos ~= 0 then
    DVIFile = substr(File, 1, dotpos-1) || ".dvi"
else
    DVIFile = File || ".dvi"

/*  Call dviprint
 */
'dviprint' '"' || DVIFile  || '"'

/* Delete temporary files
 */
call delete(file)
call delete(dvifile)
```

If you want to print with TeX you have to create the print mask in a TeX compatible manner. For instance, you have to include a text element with the text \documentstyle{article} or something similar in the header if you work with LaTeX. Furthermore, the file must not contain any control charaters. Thus, Style attributes and formfeed elements cannot be used. The Fiasco distribution contains several examples for this.

1.50 Printing with ARexx

Printing with ARexx

"Printing with ARexx" is a very comprehensive topic. This section should give you a rough idea of what can be done and how.

One way of printing with ARexx has already been explained in the section Printing with TeX. You may "misuse" ARexxPrint.rexx for purposes other than calling TeX. For example, you may use a script which parses the data for your own purposes or loads it into your word processing program.

If you want to create more complex print-outs, which cannot be created with Fiasco's internal print function, you have to create the print-out with ARexx alone. Such an ARexx script has to go through the whole database and get the data it needs with GetField. After that it may do with the data what it wants.

GraphPrint.rexx

1.51 GraphPrint.rexx

GraphPrint.rexx

The Fiasco distribution contains a complex example for a script, as it was described in the previous section. The script GraphPrint.rexx is located in the ARexx directory and can be used with the GraphDemo project. However, it can be used with any other project that contains the required data. The script reads data from the project and creates an x/y diagram of the data. It automatically adapts to different value ranges. The script uses LaTeX and the eepic extension for the print-out. That means that you have to run a special host program in the background while printing. Because the script performs many mathematical operations it uses the rexxmathlib.library, which is not included in the distribution. To start GraphPrint.rexx, click on the Graphic button in the GraphDemo/Fragments project. To use the script with another project, simply activate the project in Fiasco and start the script from the Workbench or Shell. Several requesters will appear. You have to specify what fields you want to use. You may select whether you want to view or print the TeX file directly or to write it to a specified location. After that the advanced options menu appears. To modify nothing, simply click on Continue. Edit Scale Base allows you to specify a value which will be used by the script as a base value for the scale of one of the axises. For example, if you use 5 (which is the default) you will get a scale of 5, 10, 15, etc. If you use 2 you will get 2, 4, 6, etc. Edit Origin allows you to choose whether the diagram will begin at point (0;0) or at a point which is the best for the project.

1.52 Import and Export

```
                Import and Export

The Import and Export functions of Fiasco provide the ability to load
data from other database programs into Fiasco and to write data with
Fiasco that may be read by other programs.  Such Import/Export-files
contain ASCII data. The fields or records are marked with special
characters that may be freely defined in the Import/Export function of
Fiasco.

    Beginners, please note:  Some basic knowledge is required to be able
to effectively use Fiasco's Import/Export function. If you are familiar
with databases you can skip the following information.  The section

                Special characters
                 describes the special escape sequences used by Fiasco.
  Although other databases may use a similar scheme you should read this
section carefully. The whole Import/Export function of Fiasco relies on
these escape sequences.



                How to specify special characters

                Importing of Data

                Exporting of Data
```

1.53 Structure of Import/Export files

```
Structure of Import/Export files

The names used here refer to the gadget labels in the Import/Export
requesters. Note that some marking characters may be empty. To use the
file with Fiasco you have to define, at minimum, either Field Start/Field
End or Field Separator and either Record Start/Record End or Record
Separator.   However, the import functions of other programs may get
upset, although this structure is correct. When you export data, Fiasco
will filter the characters used as control characters from the exported
data. I.e., if one fields contains 1,2 and the , is used as control
character, Fiasco will export 12.


Record Start
Field Start
Field Data   Contents of the field in ASCII format.
Field End
Field Separator    Separates two fields, not used after the last field of a record ↩
    .
...
Field Start
Field Data
Field End
```

```
Record End
Record Separator   Separates two records,  notused after the last record of a file ↩
    .
...
Record Start
...  (see above)
Record End
End of File
```

If you activate First Record contains IDs, the field IDs will be stored in the first record as if they were fields.

When you export listview fields, Fiasco will separate the entries with the characters specified in Listview/Separator.

An Example of an Import/Export file

Record start and record end are empty. Record separator is a newline character. Field start and field end are double quotes. Field separator is a comma. The first record contains the IDs of the fields. Note the empty field in the last record.

```
"Name","FirstName","Rank","Ship"
"Picard","Jean-Luc","Captain","U.S.S. Enterprise"
"Riker","William Thomas","Commander","U.S.S. Enterprise"
"Data","","Lieutenant Cmdr.","U.S.S. Enterprise"
```

1.54 How to Specify Special Characters

How to Specify Special Characters

You often cannot simply type the characters for marking fields and records as plain text. For example, if you want to use the newline character as a record separator, you cannot simply hit the Return key. Instead, you have to type it in as an escape sequence. Fiasco supports escape sequences similar to the escape sequences of the "C" programming language. The escape sequences are introduced by a \. These are supported:

```
 \n  Newline-character, ASCII 10
\f   Formfeed-character, ASCII 12
\r   Return-character, ASCII 13
\t   Horizontal tabulator, ASCII 9
\v   Vertical tabulator, ASCII 11
\Number  Character with specified ASCII code
 \Char   Character directly copied
```

The last option (\ + Character) makes it possible to use a character, which is reserved for escape-sequences.

In Import, you may also specify character-classes. Character-classes are introduced in Fiasco with an #. These are supported:

#p Printable character.
#a Printable ASCII-character. Without international chars
#c Control-character. Not printable

Export supports to insert some additional information in the export-file. These commands are introduced with an %. These are supported:

%f ID of field
%r Number of record

1.55 Importing of Data

Importing of Data

The import requester is the GUI interface for Fiasco's import function. You can open it using Project/Import. The file you want to import must be specified in File. After having done this you have to specify the structure of the file in the requester.If you are importing a file into Fiasco immediately after export it from another database and still know the structure parameters you can simply copy them into Fiasco's import requester. Otherwise you can display the contents of the file using the View button at the right side of the filename. Fiasco will start either "More" or "MultiView" to display the file. If the file has a standard structure it should not be too difficult to recognize the parameters. Usually, Record Start and Record End are empty and Record Separator is \n. Field Start and Field End are often empty or double quotes ("). Usual values for Field Separator are a comma (,) or a tabulator (\t).

Skip Lines defines the characters that introduce a comment at the beginning of a line. If present, specify the comment introducer here. This may also be used to skip any formatting information present in the file. Fiasco's import function does not use such information. You can use Start Skip to skip any initial comment or similar items in the file. Max. Fields can be used to specify a record end mark if neither Record Separator nor Record End can be used.

Activate First Record contains IDs if the first record of the input file consists of Field IDs rather than real data. If you activate this the IDs will be used by Fiasco either to create fields with these IDs or

to use already existent fields.

The options Append new fields and Overwrite old project control, whether you want to update a project or you want to create a new one. If you want to create a new project, you should activate both options.

If you want to continue using your current settings you may save them with the Save button. Settings may be reloaded with Load. Fiasco already comes with several settings to import data from various sources.

To start the import process, you just have to click on Ok.
Attention: If the input file is too big, or even if the structure parameters are defective, the system may run out of memory! Fiasco has no big problems, if it runs out of memory, but other programs may have problems. For this reason, you should be careful with unsaved data!

If everything went well, the import requester will close and the new project will be activated. You will first want to improve the formatting of the project using the mask mode. If you did not activate First Record contains IDs, you should change the field IDs according to the contents of the fields. In addition, you should create text fields to label the existing fields. At this point you have a nicely formatted project. However, all fields are string fields. You should determine whether some fields may be integer, cycle or other field types. You may change the type of these fields with the Fiasco's
convert
function. In the example
used in
Structure of Import/Export files
, the rank field may be
converted to a cycle field.

If you have followed these steps the project should be saved under a appropriate name.

1.56 Exporting of Data

Exporting of Data

From Fiasco's viewpoint, exporting data is much less complicated than importing. Normally, you can use Fiasco's default parameters (No Record Start and Record End, a newline character for Record Separator, double quotes for Field Start and Field End and a comma for Field Separator). If you use these parameters, you must take care, that you data do not contain any double quotation marks. In addition, you have to be certain that the program you want to import the data supports these parameters. If you select First Record contains IDs, Fiasco will create an additional record at the top of the file which contains the field IDs. The file will contain no other formatting information.

If you select Marked Records only, only the marked records will be written.

Click on Ok to start exporting.

1.57 Updating databases with Im/Export

Updating databases with Im/Export

Fiasco's import and export function can be also used to automatically insert data created in other databases in an already existing database. To do that, you have to have a database in any export format that Fiasco is able to read. Its first record has to contain field IDs for the data following it. These IDs must match the IDs of the already existing Fiasco database, into which you want to import the data. I.e. if you have a field with the ID Name in the existing Fiasco database, the field ID for the respective field specified in the exported file must be also Name. If it is not, you can easyly change the ID with a text editor. The already existing database may contain fields that are not in the file to be imported. If the file contains fields that are not in the Fiasco database, Fiasco's import function will create a new field for these data. To initiate importing, you have to make the apopriate settings for the structure of the file to be imported. Furthermore, you must activate First Record contains IDs (as Fiasco could otherwise not associate the data to the correct fields). Append new fields and Overwrite old project must be switched off.

This method works both with data exported by Fiasco and with data exported or created by any other program, as long it is in a format that can be read by Fiasco.

1.58 Fieldtypes

Fieldtypes

Data are stored in fields. There are only two basic types: "string" and "number". All other types are modifications, more or less, of these types which make working with the database easier. Some fieldtypes may be used in

listview fields

. These are fields, which may contain several entries. Entries can be added or deleted. Each entry can be used like any other normal field of that type.

Fiasco supports the following types:

String

Integer

Float

Boolean

```
                Cycle

                Slider

                Date

                Time

                Extern

                Datatypes

                Var String

                Text

                Button

                Bar

                Listview
                   The descriptions for the field types are based on the

               Standard Attributes
               .   Each field type may add new attributes, change an
attribute or it also might not support a particular attribute.
```

1.59 Standard Attributes

```
               Standard Attributes

These attributes are normally supported by a field type:


Structure/ID: This string identifies a field. It is displayed in mask
   mode in the fields, in the list header, in the search and related
   requesters and in the relation requester. This string must be unique
   in the current project. To access the field by its ID from formulas or
   ARexx scripts, further limitations on the format of the ID apply: It
   may not contain spaces and it may not begin with a number.

Structure/Virtual: The value of the field is not saved on disk, but is
   recalculated every time when the record is required. This is done
   using the init cont attributes and the formula or the ARexx script
   attribute. Please note that these fields occupy the same amount of RAM
   as other fields.

Structure/Listview: The field will be a
               listview
                field. This modifier can
   only be changed when the field is created. To change the listview
   modifier later, you will have to use the
               convert function
               . Listview
   fields cannot be displayed in the list window, thus the List Window
```

attributes are not available, when this modifier is active.

Mask Window/Width: defines the width of the field in the mask in characters.

Mask Window/Height: defines the height of the field in the mask in characters.

Mask Window/Keyboard Shortcut: You may enter a single character here. When you press this character on the keyboard in record mode the field will be activated or somehow changed. The shortcut paired with Shift may change the field in the other direction.

Mask Window/Justification: Controls whether the contents in the field will be displayed left or right justified or centered.

Mask Window/Read Only: The field content will be displayed in a recessed box which cannot be activated or edited.

Mask Window/Hidden: If this option is active the field will be not visible in the mask. To open the fieldrequester for it you will have to use the menuitem

Field/Edit named Field

.

List Window/Width: Specifies the width of the field in the list window. It is measured in characters. You may also change this value directly in the

list window

.

List Window/Justification: Controls whether the contents of the field is displayed in the list window left or right justified or centered.

List Window/Hidden: The field column will be not displayed in the list when this option is active. You may also change this option directly in the

list window

.

Initial Content/Use own value: you may specify a value here which will be used while creating a new record.

Initial Content/Use old value: If you create a new record the value which has been used in the old record will be used in the new record.

Predefined Values: This feature allows you to add a picker button at the right side of a field which can be used to open a list of values the field can be set to. Clicking on the Predefined Values button in the field requester will open a new requester that can be used to edit the settings for this attribute. You may either specify a list of possible values for the field or you may specify an ARexx script that will be executed when the picker button is selected. This ARexx script may for instance open a file requester and use SetField to set the desired field to the selected file. The list of predefined values may be also used from the search requester with the picker button at the right side of the pattern gadget.

Programming/Formula: You may specify a formula here that will be used to
calculate the content of the field. The formula may reference other
fields to make the calculation. Whenever one of the other fields is
changed or a completely new record is added, the formula will be
calculated again. This attribute can be combined nicely with the
Virtual attribute. More about formulas
here
.

Programming/ARexx Script: You may specify an ARexx script here which
will be called when a new record is created, or the content of a field
is changed. It is possible that the initial content attributes will
not have the effect specified in the requester, if the script changes
the contents of the field. If the operation is also possible with
formulas, it is recommended to use formulas instead.

By using
mask stretching
it is possible that the attributes, which
specify the dimensions of the field, will be slightly influenced.

1.60 String Fieldtype

String Fieldtype

A string field takes strings with a designated length.

Standard Attributes
New Attributes:

Structure/Max Chars: determines, how many chars may be typed in this
field. This attribute has direct effect on the size of the project
file.

Changed Attributes:

Mask Window/Height: This attribute is only active when the

listview modifier
is active.

Search equivalent:

correspondents to the content.

```
Supported search patterns:


 ?   -   One unknown character.
#?   -   No or more unknown characters.



Conversion into a string field:

Any field can be converted without loss of data into a string field.
Alternative formats, if supported are specified in parentheses.
Additional notes:


Boolean   -   "Checked" is TRUE (1), otherwise FALSE (0)
Cycle   -   Label (label number) converted
Slider  -   Level converted
Date  -   Date in current locale format converted
Time  -   Time in current locale format converted
```

1.61 Integer Fieldtype

```
                Integer Fieldtype

You may enter integer numbers in the range from -2,147,483,348 to
2,147,483,347 in an integer field.
                Standard Attributes
                New Attributes:


Structure/Max Chars: determines the maximum length of a number in chars.

Initial Content/Use unique Key: puts a number unique to this database in
   this field whenever a new record is created. This Attribute is
   mutually exclusive to use own value and use old value.



Changed Attributes:


Mask Window/Height: This attribute is only active when the

                listview modifier
                 is active.



Search equivalent:
```

```
is equal with the field content.


Supported search patterns:


>    -   greater than
<    -   less than
>=   -   greater or equal
<=   -   less or equal
!=   -   not equal



Conversion into an integer field:

Integer fields only accept the numeric part of the source data. If the
source data begin with a non-numeric character the field will contain 0.
Additional notes:


Float   -   Integer part converted
Boolean   -   "Checked" gets 1, "Unchecked" gets 0
Cycle   -   Label number converted
Slider  -   Level converted
```

1.62 Float Fieldtype

```
                Float Fieldtype

You may enter a real number in a float field.


                Standard Attributes
                New Attributes:


Mask Window/Precision: Number of digits after the decimal point.



Changed Attributes:


Mask Window/Height: This attribute is only active when the

                listview modifier
                 is active.
```

```
Search equivalent:

is equal to the field content


Conversion into a float field:

Float fields only accept the numeric part of the source data. If the
source data begin with a non-numeric character, the field will contain 0.
Additional notes:


Boolean   -   "Checked" gets 1.0, "Unchecked" gets 0.0
Cycle   -   Label number converted




Notes:

Since Fiasco 2.1, float fields are used in 64 bit representation
internally. This feature now allows a reasonable high precision for float
fields. Databases in a pre 2.1 format are automatically converted during
the next saving.
```

1.63 Boolean Fieldtype

```
                Boolean Fieldtype

A Boolean field can contain only one of two values: "True" or "False". It
appears in the mask as a "checkbox gadget".
                Standard Attributes
                Changed Attributes:


Structure/Listview: not applicable.

Mask Window/Justification: not applicable.

Mask Window/Width: always 3.

Mask Window/Height: always 1.

Predefined Values: not applicable.



Search equivalent:

TRUE or 1 - checked field
FALSE or 0 - unchecked field
```

Conversion into a boolean field:

Boolean fields convert all non-0 numbers and TRUE into the checked state.
All other values will be converted to the unchecked state.

Notes:

Under Amiga OS 2.x this field can look a bit strange because the images
are not scalable. Starting with OS 3.0, the size of the field is adjusted
to the font size.

1.64 Cycle Fieldtype

Cycle Fieldtype

Cycle fields have several choices from a freely definable list, this
helps to save memory. There is a maximum of 65536 choices. (I hope that's
enough ;-) A cycle field appears in the mask as a "Cycle gadget" (as the
name implies).
Standard Attributes
New Attributes:

Labels: A list of all choices. There must be at least one entry, two
entries make it a cycle field.

Changed Attributes:

Structure/Listview: not applicable.

Mask Window/Justification: always centered.

Mask Window/Height: always 1.

Predefined Values: not applicable.

Search equivalent:

the number of the label, counting from zero or the entry itself (enter
correctly!)

Conversion into a cycle field:

The values will be converted into labels. If there are equal values they
will get the same label. Data are not lost.

```
Additional notes:


Boolean    -   "Checked" becomes TRUE (1), otherwise FALSE (0)
```

1.65 Slider Fieldtype

```
                Slider Fieldtype

A slider is related to a integer field. It can be used to display integer
numbers graphically. The numbers may range from -32,768 to 32,767 and may
be influenced by several attributes.


                Standard Attributes
                New Attributes:


Structure/Min. Value: defines the smallest value. It corresponds to the
   position of the "knob" at the left or at the upper end of the field.

Structure/Max. Value: defines the highest value. It corresponds to the
   position of the "knob" at the right or at the lower end of the field.

Mask Window/Format: is a format string in style of the "C" programming
   language. The result will be displayed at the right hand of the
   slider. The syntax:  %[-][0][Field][.Maximum][l]Format

·  -: The number is left aligned, the default is right aligned

·  0: The field is padded with zeroes. e.g.: 1 -> 001

·  Field: The minimal field width

·  Maximum: only for strings, no meaning here.

·  l: Says that the number is 32 bit wide. This is here always the case.

·  Format:
   c - Char, the ASCII character for the number is displayed.
   d - The number is displayed.
   u - The unsigned number is displayed.
   x - The number is displayed in hexadecimal format.
   There are also the b and s control characters. These take addresses as
   arguments and produce only garbage in this case.



    The formatting is done with the exec-function RawDoFmt().


Mask Window/Format Length :the maximum length of the format. This
```

```
region is in the width region. That means that a higher Format Length
makes the field itself smaller.



Changed Attributes:


Structure/Listview: not applicable.

Mask Window/Justification: not applicable.

Mask Window/Height: always 1.

Predefined Values: not applicable.



Search equivalent:

The number itself.


Supported search patterns:


>    -   greater than
<    -   less than
>=   -   greater or equal
<=   -   less or equal
!=   -   not equal



Conversion into a slider field:

Slider fields only accept the numeric part of the source data. If the
source data begin with a non-numeric character the field will contain 0.
You should check the range attributes after converting -- they could
influence the data.
```

1.66 Date Fieldtype

```
                Date Fieldtype

You may enter a date in a date field. The date will be formatted
according to the specifications in the active locale settings. If
locale.library is not available, the format DD.MM.YYYY will be used.

                Standard Attributes
                New Attributes:
```

```
Initial Content/use current Date: When a new record is created the
   current date is copied in this field.


Changed Attributes:


Mask Window/Height: This attribute is only active when the

                listview modifier
                 is active.


Search equivalent:

is equal to the content.


Conversion into a date field:

Date fields require the data in the current locale format. The single
parts must be numbers. If values are non numeric, the part will get "??".


Note:

Even though it is with most formats possible to enter a two-digit year,
it is recommended to use a four-digit year to avoid any of the problems
the DOS world currently has.
```

1.67 Time Fieldtype

```
                Time Fieldtype

You may enter a time in a time field. The time will be formatted
according to the specifications in the active locale settings. If
locale.library is not available, the format HH:MM:SS will be used.

                Standard Attributes
                New Attributes:


Structure/Duration Format: Normally, the locale.library format will be
   used to format the time. However, in some countries (most notably the
   USA) this is not suitable to display time durations. If this attribute
   is active, always the format HH:MM:SS will be used. Furthermore, the
   range checking for the hour will be turned off. Thus, you will be able
   to enter a duration longer than 24 hours.
```

```
Initial Content/use current Time: the current time will be copied in this
   field when you create a new record.



Changed Attributes:


Mask Window/Height: This attribute is only active when the

                listview modifier
                 is active.



Search equivalent:

is equal to the content.


Conversion into a time field:

Time fields require the data in the format HH:MM:SS. Every element must
be a number. If an element is non numeric, it will be 0.
```

1.68 Extern Fieldtype

```
                Extern Fieldtype

A extern field takes a string (most often a filename) that will be used
on request as argument for a user defined program. This makes it possible
to define additional data for a record.
                Standard Attributes
                New Attributes:


Structure/Max Chars: defines the maximum length of a filename in chars.
   This attribute has direct effect on the size of the project file.

Command/Command: is the name of a program, which is capable of using
   these data. The characters %s are replaced with the content of the
   field. If you don't use %s, no arguments will be submitted. (For
   example type: C:ED %s)

Command/Stack: defines the stack size for a command.

Mask Window/FileReq Gadget: select this attribute to have an gadget at
   the left side of the field that opens a file requester to edit the
   content. Of course, this only makes sense if the contents are
   filenames.
```

```
Changed Attributes:


Structure/Listview: not applicable.

Mask Window/Justification: not applicable.

Mask Window/Height: always 1.

Predefined Values: not applicable.


Search equivalent:

is equal to the content.


Conversion into a extern field:

All fields can be converted without loss of data into an extern field.
However, you have to specify a program that can use these data.
Additional notes:


Boolean   -  "Checked" becomesTRUE (1), otherwise FALSE (0)
Cycle   -  Label (Label number) converted




Notes:

The programs will be called using the AmigaDOS function System(). A
console window will be opened for I/O operations.
```

1.69 Datatypes Fieldtype

```
                Datatypes Fieldtype

A datatypes field is similar to an extern field. The difference is the
use of the datatypes.library. This is the reason you can use these fields
only with Amiga OS 3.0 or greater. The major advantage is that the data
will be displayed directly in the mask. A datatypes field is universally
usable and freely extensible. A "popup"-gadget at the lower right side of
the field makes it possible to edit the contents using a file requester.
If something goes wrong, the error will be displayed in the field.


                Standard Attributes
                New Attributes:
```

Structure/Max Chars: defines the maximal length of the filename. This attribute has a direct effect on the size of the project file.

Mask Window/Options/Display filename: When this option is active the filename is displayed at the bottom of the field in a string gadget. If you deactivate this option you cannot edit the value of the field. Using the ARexx script requestdt.frx with a button, editing the value is indirectly possible.

Mask Window/Options/Scrollbars: Determines if scrollbars will be created at the bottom and at the right border of the field. Without a scrollbar you can only view the upper left of a file. (That is not completely true. Some datatypes scroll their display if you click in their area and drag the mouse in the direction of the hidden part. The picture datatype is one example.)

Mask Window/Options/Save gadget: If you activate this option you will get a second button under the datatypes field. The button will be marked with an S. If you select the button a file requester will appear which lets you choose a file to which the data, which are currently displayed in the field, will be saved. The data will be written in IFF format.

Mask Window/Options/Border: If this option is active Fiasco will render a border around the field. Do not deactivate this option too often because there are no other visual elements which mark the beginning and the end of the field.

Mask Window/Options/Defer loading: If you activate this option, the file of the field will not be immediately loaded when the record is activated. Instead, the message "Deferred" will be displayed in the field. The data will be loaded and displayed only if you activate the string gadget and hit return.

Mask Window/Options/Pictures/Scaling: Allows you to control the scaling of pictures in the field. If the content of the field is not a picture, this attribute has no effect. This options determines whether there will be no scaling at all, only if the picture is bigger than the field or if there will always be scaling.

Mask Window/Options/Pictures/Scale size: Allows the selection of the scaling mode. Full Size will make the picture exactly fit into the field and may thus change the proportions of the picture. Proportional Small uses a size that fills the field in one direction and which completely fits into the field. Furthermore, it keeps the proportions of the picture. Proportional Big uses the minimal size, which entirely fills the field. The proportions will be kept here, too.

Mask Window/Options/Texts/Word Wrap: Select this option to activate word wrapping for texts.

Mask Window/Options/Miscellaneous/Play immediately: Select this option to start playing the data immediately after activating the record. If you activate this option, Defer loading must not be active. Of course, this option is only effective if the datatype supports playing. The animation and the sound datatypes are such datatypes.

```
Mask Window/Options/Miscellaneous/Center: Centers the datatypes object
   within the field.



Changed Attributes:


Structure/Listview: not applicable.

Mask Window/Justification: not applicable.

Predefined Values: not applicable.



Searchequivalent:

Is equal to the filename; You cannot search the content.


Conversion into a datatypes field:

All fields can be converted without loss of data into a datatypes field.
However, the datatypes system requires valid filenames.
Additional notes:


Boolean   -  "Checked" becomesTRUE (1), otherwise FALSE (0)
Cycle   -  Label (Label number) converted




Notes:

The AmigaGuide and the animation datatype seem to have some problems with
relatively small fields.

    AmigaGuide datatype sometimes leaves graphical trash after scrolling
the contents.

    HAM and EHB pictures cannot be displayed in datatypes fields.

    The changing of records gets slower because the data have to be
loaded each time. To avoid that use Defer loading.
```

1.70 Var String Fieldtype

```
                Var String Fieldtype

Var string fields take strings with variable length opposed to string
fields, which use a designated maximum length. Var string fields may
```

```
contain several lines.
                Standard Attributes
                New Attributes:


Mask Window/Scrollbars: Adds a scrollbar at the right side of the field.
   You can use it to scroll the contents of the field.



Changed Attributes:


Structure/Listview: not applicable.

Mask Window/Justification: not supported.

Predefined Values: not applicable.



Search equivalent:

corresponds to the content.


Supported search patterns:

? = One unknown character.
#? = No or more unknown characters.



Conversion into a var string field:

Any field can be converted without loss of data into a var string field.
Alternative formats, if supported, are specified in parentheses.
Additional notes:


Boolean   -   "Checked" is TRUE (1), otherwise FALSE (0)
Cycle   -   Label (label number) converted
Slider  -   Level converted




Notes:

Var string fields are implemented using the textfield.gadget by Mark
Thomas. There is cut and paste support in var string fields: you may mark
text parts using the mouse. This text is cut with A X and copied with A
C. Text from the clipboard can be pasted at the current cursor position
with A V. These shortcuts are only active if the field is active, i.e.
the cursor is visible or some text is marked (this also works with
read-only fields). Otherwise, these shortcuts will invoke other Fiasco
functions.
```

Relatively big var string fields may slow down record changing.

Starting with Fiasco 2.1, the contents of var string fields may be displayed as a single line in the list window. Lines breaks in the field are converted to spaces for the list window. Note, however, that var string fields with a relatively long content may slow down rendering the list substantially. You should hide those fields in the list.

Since the introduction of var string fields in Fiasco 2.00 till Fiasco 2.02 (including) var string fields could not be displayed in the list window. If you load databases created with these versions with Fiasco 2.1 or higher, the var string fields are marked as hidden in the list. To display them, simply deselect the appropriate attribute.

1.71 Text Fieldtype

Text Fieldtype

Text fields are not real fields; these fields only serve to put text in the mask.

This fieldtype supports no standard attributes.

Supported Attributes:

Text: Will be written in the mask. An underscore ("_") placed before any character will cause the character to be printed underlined. Thus, you may mark the keyboard shortcut of a field.

Pen: The color used to write the text. The Normal default is black and the Highlight default is white. The colors can be manipulated with the palette prefs editor.

Bold: Makes the text bold.

Italics: Makes the text italic.

Underlined: Underlines the text.

Search equivalent:

You cannot search for a text field

Conversion into a text field:

You cannot convert any other fieldtype into a text field.

1.72 Button Fieldtype

```
                Button Fieldtype

Button fields only serve to put a button in the mask for a user-definable
action and are not real fields.  This fieldtype only supports the width
and the shortcut
                standard attribute
                .


Supported Attributes:


Text: will be displayed in the button.

Type: Use Type to choose whether the button will execute a CLI or an
   ARexx program. CLI programs may be normal programs, commands or
   scripts (with the "s" attribute). ARexx programs must be ARexx
   scripts.

Command: Use Command to select the program that will be executed when the
   button is activated.

Stack: You may specify the stack size for the program here. The default
   is 4096. The program to be activated will crash if the stack size you
   specify is too small.

Console Window: lets you specify the I/O stream for the program. It may
   be a console-window (CON:), the printer (PRT:), a simple file, or, if
   you don't want any output NIL:. Since Fiasco 2.1, this is also
   implemented for ARexx scripts.



Search equivalent:

You cannot search for a button field


Conversion into a button field:

You cannot convert another fieldtype into a button field.
```

1.73 Bar Fieldtype

```
                Bar Fieldtype

Bar fields only serve to put a visible separation in the mask and are not
real fields.  The bar fieldtype supports no standard attributes.


Supported Attributes:
```

Width/Height: The width or the height of the bar, depending on Freedom.

Freedom: determines whether the bar is drawn in the mask horizontally or
   vertically.

Search equivalent:

You cannot search for a bar field

Conversion into a bar field:

You cannot convert another fieldtype into a bar field.

Special behaviour in groups:

Bar fields are sociable fields, thus several bars in a
                group
                , which touch
each other (only for technical reasons), will optically join. This
feature allows you to create boxes with four bars in a square.

1.74 Listview Fieldtype

                Listview Fieldtype

The listview fieldtype is not a real fieldtype, but a fieldtype modifier.
 It is currently supported by five fieldtypes:

· String

· Integer

· Float

· Date

· Time

To make a field a listview field, you have to activate the Listview
option in the new field requester. The edit field requester, which may
have been opened using Field/Edit Field does not allow changing of this
option, because changing this modifier requires a conversion of the
field. The Convert Field function does that job.

    Listview fields may take any number of entries. These entries
correspond to simple fields. An entry may be activated by clicking on it
in the list part of the field. The active entry may be edited in the
gadget below the list part. To add an entry, you have to click on the '+'

gadget of the field. The '--' gadget removes the active entry.

Changed Attributes:

Mask Window/Height: The height attribute becomes available when you activate the listview modifier.

List Window: An active listview modifier makes all list window attributes unavailable, because listviews cannot be displayed in the list window.

Initial Content: This is the value, which will be inserted into the new added entries. Listviews in new records are always empty.

Predefined Values: Not applicable.

Search Equivalent:

You cannot search for a listview field

Conversion into a listview field:

When you convert a listview field into a listview field of another type, each entry of the listview will be converted as described in the documentation for the new fieldtype.

If you convert a non-listview field into a listview field or vice-versa, the '|' character will be used to separate the entries from each other. Because this character is not available in numeric fields, such conversions do not make much sense (the first number is converted though).

Special behaviour in groups:

Listviews, which have been grouped, will always activate the entries with the same number. Thus, if you activate the second entry in one grouped listview, all other listviews in the group will also activate the second entry. Furthermore, the add and delete functions of listviews are applied to all listviews in a group.

If you place grouped listviews directly beneath each other in the mask and if the listviews have the same height and the same status of the Read Only attribute, these listviews will share their visual appearance.

Notes:

If you make a listview field too narrow, Fiasco will remove the +/-- gadgets to get more space. To get the functionality of these gadgets, you may either group the field with another listview field or use the ARexx

commands AddLVFieldEntry and DeleteLVFieldEntry.

1.75 Fiasco's Graphic User Interface

Fiasco's Graphic User Interface

Fiasco initially opens with an empty window. You can use the pull down menus to work in it. The people who don't like pull down menus may open an additional window using
Control/Service Window
. This window makes the most important operations accessible via a mouse click. Keyboard shortcuts are the third way to execute operations.

The Mask Window

The List Window

The Service Window

Menus and shortcuts

Requesters
Fiasco supports menu help. If you press the help key while you ↩ browse through the menus, a short description will be displayed in an AmigaGuide window (This feature requires amigaguide.library, which is part of the OS since release 3.0. If you use 2.0 or 2.1, you may get it from the PD).

The requesters used by Fiasco have a standard structure. The gadgets at the bottom are for responding. Normally, the left one is a positive response, while the right one is negative. The close gadget of the window is equivalent to a negative response. Nearly all gadgets in the requesters may be accessed using the keyboard. Use the Return key for a positive response and the Esc key for a negative response.

1.76 The Mask Window

The Mask Window

The operation in the mask window differs in Fiasco's two editing modes.

Mask Mode

The main control in the mask mode is the cursor. The cursor may be moved with the cursor keys. Without any qualifier, pressing a cursor key moves the cursor one unit in the direction of the cursor key. If you press a cursor key paired with Shift, the cursor will be moved to the appropriate end of the window, i.e. if you press Shift and Cursor right, the cursor will move to the right border of the window. If you pair a cursor key with Ctrl, the cursor will be moved to the extreme position of the mask.

The cursor right and down keys will move the cursor to the position of the last field in that direction, the cursor left and up keys will move the cursor to the X position 0 or Y position 0, respectively.

If you reach a field with the cursor, it will be activated.

If you click any place in the mask with the mouse, the cursor will be set to that position. If you click on a field, it will be activated. You may select several fields by holding Shift pressed while selecting the fields.

The selected fields may be dragged using the mouse. Hold the left mouse button pressed while moving the mouse. When you release the left mouse button the fields will be placed at the point where you have dragged them. If you press the right mouse button while dragging the fields, the dragging operation will be cancelled, all fields will return to their original positions.

If you double click on a field the
field requester
for that field
will be opened.

Record Mode

In the record mode, all fields in the mask behave much like normal gadgets in normal applications. Fields may have a keyboard shortcut. See the
field documentation
for more information.

Furthermore, pressing the tab key in a String, Integer, Float, Date, Time, Extern, Datatypes or Var String field will cause the next field to be activated. Shift and Tab will activate the previous field. If you press Tab when no field is active the first field in the mask will be activated.

If you prefer using the Enter key to cycle through fields as in Fiasco 1.x, you will have to edit the
user interface settings
.

Mask Stretching

1.77 Mask Stretching

Mask Stretching

Normally, the Fields in a Fiasco mask are placed very close together. This is not very nice and all other "normal" GUIs leave a few pixels between the gadgets. It is possible to place one empty line between the fields, but this wastes quickly a lot of place. For this reason Fiasco makes it possible to leave a few pixels between the gadgets. These values

may be specified in the
options requester
under Sretch X and Stretch Y.
The owl stretching (ehhhmm -- mask stretching %-) makes fields bigger than specified in the field requesters. This is evident in the lines, because most Fiasco fields only expand to this direction. String fields may be bigger than the number of chars they can hold. The biggest problem are text fields, because their width is normally the minimum required. Stretching makes them wider and the text has to be centered. You should specify zero as X value to avoid these problems and use one column as a separator. In Y direction, 4 is the best value.

1.78 The List Window

The List Window

The list window may be opened using
Control/List Window
. The upper part
of the list window is the head of the list. It displays the IDs of the fields, which are represented by the columns. The lines of the list represent the records; the active record is marked with a strong backfill. Marked records are displayed with a thin backfill. Fiasco's list window displays only those records, which are in the active index. The successor of a record is displayed under the record. If you click on one record-line, the record will be activated. However, you must use the mask to edit a record.

The scroll bars of the window can be used to scroll through the contents of it. The vertical scroll bar scrolls through all records, the horizontal scrolls trough the fields in the list.

The layout of a list, which means position and width of the columns, is done automatically. However, you can also control it directly in the list window and in the field-requesters in the mask mode.

To change the position of a column, you have to click in the middle of the column's head. Do not release the mouse button. Two lines will appear at the current position of the column. Now you may drag it over some other column. If you release the mouse button, Fiasco sets the column as near as possible at the new place. Columns, which are overlapped by the column, are shifted to the right. The old place of the column will be filled by shifting columns right of it to the left.

The width of a column may be also changed with the mouse. When you click at the right corner of the head of a column and do not release the mouse button, a line will appear. You may now drag the line with the mouse. The place, where you release the mouse button, will become the new right border of the column. The fields right of it will be shifted accordingly. You may also use the
field requester
of the field to change
the width of its column. The attribute
List Window/Width
serves for this

```
purpose.

    You may also control, whether a field appears as column in the list,
or not. Normally, when a field is created, it is automatically added to
the list. To hide a column, activate the header of a column by clicking
on it and select the menuitem
                List/Hide column
                . You may also hide it by
activating the
                List Window/Hidden
                 attribute in the field requester. To
reveal it, use the menuitem
                List/Show column
                 or deactivate the List
Window/Hidden attribute.

    The menuitem
                List/Show all columns
                 makes all hidden columns visible.


                List/Recalc list
                 calculates the positions and dimensions of all
columns again. You can compare it with Clean up of the Workbench.
Columns, which have been hidden, are kept hidden.
```

1.79 The Service Window

```
                The Service Window

The service window may be opened or closed with
                Control/ServiceWindow
                . If
you want Fiasco to open the service window on every program startup,
select the option Service Window/Open on Startup in the

                user interface settings
                . If Service Window/Fixed Position in the same
requester is inactive, Fiasco will search for a free place on the screen
when Fiasco opens the window. Otherwise, the position of the service
window at the time of saving the settings is used.

    The service window contains these gadgets:


                 Add

                 Del

                |<

                 <

                >
```

```
>|

 <Filename>

  <Status>

<Fieldtype>
```

1.80 Add

```
Add
```

If the current project is in record mode a new record will be created. If mask mode is active a new field will be created. Equivalent to:

```
Record/Add
 in record mode
```

resp.

```
Field/Add field
 in mask mode.
```

1.81 Delete

```
Delete
```

If the current project is in record mode, the current record will be removed. If mask mode is active, the current field will be removed. Attention: This will normally happen without any security request!

Equivalent to:

```
Record/Remove
 in record move
```

resp.

```
Fields/Remove Field
 in mask mode.
```

1.82 First

```
First
```

If the current project is in record mode, the first record will be activated.

Equivalent to:

```
Record/First
```

1.83 Previous

```
                Previous

If the current project is in record mode the previous record will be
activated.

    Equivalent to:

                Records/Previous
```

1.84 Next

```
                Next

If the current project is in record mode the next record will be
activated.

    Equivalent to:

                Records/Next
```

1.85 Last

```
                Last

If the current project is in record mode the last record will be
activated.

    Equivalent to:

                Records/Last
```

1.86 Active project

```
Active project

The name of the current project is displayed here. If two projects only
differ in the path and not in the name, the same name will be displayed.

    You may activate another project by activating the window of a
project.
```

1.87 Status

Status

Status information is displayed here. In the record mode:
number of active record/number of records
A
Filter
may change these numbers.

In the mask mode:
X: X position of cursor, Y: Y position of cursor

1.88 Fieldtype

Fieldtype

If you are in record mode you can select the fieldtype which will be used for subsequent calls of Add Field. Equivalent to:
Fields/Field Type.

1.89 Menus

Menus

Fiasco has these pull down menus:
(from left to the right; menus, which are marked with a '/', may be activated or deactivated)

Name	Keyboardshortcut
Project	
New	A N
Erase	A Z
Open...	A O
Open new...	A L
Save	A S
Save as...	A A
Import...	A I

```
                Export...
                 A E

                Print...
                 A P

                Hide
                 A H

                Reveal...
                 A ^

                About...
                 A ?

                Quit
                 A Q

  Database

                Options...
                 A $

                Statistic...

                Indices...
                 A *

                Prev. active index

                Next active index

                Reorganize...

                Reload Relations

                Functions...

                Constants...
                  Record

                Add Record
                 A +

                Duplicate Record
                 A 2

                Delete Record
                 A -

                Delete all Records
                 A @

                Cut Record
                 A X
```

```
        Copy Record
         A C

        Paste Record
         A V

        Previous
         Cursor Up

        Next
         Cursor Down

        First Record
         Ctrl Cursor Up

        Last Record
         Ctrl Cursor Down

        Goto...
         A G

        Mark Record
         A  .

        Unmark Record
         A  :

        Mark all Records
         A  ,

        Unmark all Records
         A  ;

        Toggle all Marks
          Field

        Field Type        »
                String          Ctrl S
Integer         Ctrl I
Float           Ctrl F
Boolean         Ctrl B
Cycle           Ctrl C
Slider          Ctrl S
Date            Ctrl A
Time            Ctrl M
Extern          Ctrl E
Datatypes       Ctrl D
Var String      Ctrl V
Text            Ctrl T
Button          Ctrl U
Bar             Ctrl R

        Add Field...
         Enter

        Edit active Field..
         Enter
```

```
                Edit named Field...
                 Shift Enter

                Duplicate Field

                Remove Field
                 Del

                Edit Relation...
                 A \&

                Remove Relation
                 A  0

                Create Group
                 A J

                Resolve Group
                 A /

                Convert Field...
                 A "

  List

                Hide column
                 A [

                Show column...
                 A ]

                Show all colums

                Recalc List
                 A %

  Compare

                Find...
                 A F

                Find next
                 A >

                Find previous
                 A <
  Filter...           A ~

                Replace...
                 A R

                Count...
                 A #

                Sort...
                 A =
```

```
                Mark...
                 A K

  Control

                / Record Mode
                 Ctrl F1

                / Mask Mode
                 Ctrl F2

                / ServiceWindow
                 Ctrl F3

                / ListWindow
                 Ctrl F4

                / ARexx-Debug
                  Settings
  Databases...

                User Interface...

                User Menu...

                Display...

                Ext. Programs and Paths

                Save Settings

                Save Settings as...

                Load Settings...
                  User
User menu items
```

1.90 Project/New

```
                Project/New

Shortcut: A  N

Creates a new database project with a mask window. It contains neither
records, nor fields. You may create a new database or
                Open
                 a saved
database.

See also:
```

```
                Open
                ,
                Open new
```

1.91 Project/Erase

```
                Project/Erase

Shortcut: A  Z

Erases all data in the current project. The project will be in a status
like immediately after calling
                Project/New
                . If data have been changed
since last saving, you will be asked before the data is erased.
```

1.92 Project/Open...

```
Project/Open...

Shortcut: A  O

Opens a file requester and loads the selected Fiasco project into the
current project window. If there are any unsaved data you will be asked
whether you want to save them first.
```

1.93 Project/Open new...

```
Project/Open new...

Shortcut: A  L

Opens a file requester and loads the selected Fiasco project into an
automatically created project window.
```

1.94 Project/Save

```
                Project/Save

Shortcut: A  S

Save writes the data of the current project under the same name to disk.
If you want to save the project under a different name you have to use

                Save as
                .
```

1.95 Project/Save As...

```
Project/Save As...

Shortcut: A  A

You may save the current project under a new name here. The name will be
requested using a file requester and will be kept after saving.
```

1.96 Project/Import...

```
                Project/Import...

Shortcut: A  I

Opens the
                import requester
                , the GUI interface for the import function of
Fiasco. You can use import to load data from foreign databases into
Fiasco.
```

1.97 Project/Export...

```
                Project/Export...

Shortcut: A  E

Opens the
                export requester
                , the GUI interface for the export function of
Fiasco. You can use export to save data in a format which can be read by
other databases.
```

1.98 Project/Print...

```
                Project/Print...

Shortcut: A  P

Opens the print window, the main interface to Fiasco's
                print function
                .
You may create a layout for printing here and print it.
```

1.99 Project/Hide

```
                Project/Hide

Shortcut: A  H

Closes all windows of the active project. However, the project data will
not be freed. If the project window was the last open project window,
Fiasco will close its custom screen or unlock its public screen. An icon
will be set up on the Workbench.

    To reopen a project use
                Project/Reveal
                 if another project window is
still open or double click on the Fiasco icon on the Workbench. Both
actions will open the
                reveal project requester
                 which allows you to choose
one of the hidden projects to open.

    Another way to get access to Fiasco is to start Fiasco again. This is
useful, for instance, when Workbench is not running. Fiasco will open an
empty project window from which you have access to the
                Project/Reveal
                menuitem.
```

1.100 Project/Reveal...

```
                Project/Reveal...

Shortcut: A

Opens the
                reveal project requester
                 which allows you to choose a project
hidden with
                Project/Hide
                 to be opened again. If you have closed all
Fiasco windows, you have no access to this menuitem. Therefore, Fiasco
creates an icon on the Workbench. Doubleclicking on it will result in the
same function.

    See also:
                Project/Hide
```

1.101 Project/About...

```
Project/About...

Shortcut: A  ?
```

This item shows a requester that displays informations about version,
copyright and some system internal data.

1.102 Project/Quit

Project/Quit

Shortcut: A Q

This item closes the current project. If it has been changed and has not
been saved yet, you will be asked if you want to do this. If this is the
last open Fiasco project, Fiasco will exit.

1.103 Database/Options...

Database/Options...

Shortcut: A
This menuitem opens the
Options requester
, which can be used for editing
project specific options. That are:

.
Mask stretching
· Name of author and annotations

· project windows

· disk access time for reading records

· RAM usage of records

Before Fiasco 2.1, this menu item was in the Project menu.

1.104 Database/Statistic...

Database/Statistic...

no Shortcut

Shows some information about memory usage, etc. for the current project
in the
statistic requester
.

Before Fiasco 2.1, this menu item was in the Project menu.

1.105 Database/Indices...

```
                Database/Indices...

Shortcut: A  *

Opens the
                indices requester
                , which can be used to select, create or
delete an
                index
                .

    Before Fiasco 2.1, this menu item was in the Project menu.
```

1.106 Database/Previous active Index

```
                Database/Previous active Index

No Shortcut

Goes one step back in the
                index history
                . That means, that this item
activates the index that was active before the currently active index was
activated. To activate this index again, you have to go the step forward
again with the menu item
                Next active Index
                .
```

1.107 Database/Next active Index

```
                Database/Next active Index

No Shortcut

Goes one step in the
                index history
                 forward after is has been gone with

                Previous active Index
                 backwards.
```

1.108 Database/Reorganize...

```
                Database/Reorganize...

No Shortcut
```

```
This menuitem is similar to
                Project/Save
                . The difference is, that
Reorganize rewrites the whole database file and deletes all records
definitely, which are not used in any index of the database. Records,
which are deleted using the delete functions of Fiasco, are only removed
from the index, the data will remain until the next reorganization.
Before Fiasco starts the reorganization, you will be warned with a
requester, which displays the number of records to be deleted.

    If there are relations which search for a key defined in these
records, they will not find it after a reorganization.

    Before Fiasco 2.1, this menu item was in the Project menu.
```

1.109 Database/Reload Relations

```
Database/Reload Relations

No Shortcut

This item reloads all relations in the current project, just as they were
loaded while opening the project. This is particularly useful if you have
deactivated Update Relations in the database settings, changed some keys
and want to see the result.  Before Fiasco 2.1, this menu item was in the
Project menu.
```

1.110 Database/Functions...

```
                Database/Functions...

No Shortcut

Opens the
                functions requester
                 that can be used to edit the

                user-defined functions
                 of the active database.
```

1.111 Database/Constants...

```
                Database/Constants...

No Shortcut

Opens the
                constants requester
                 that can be used to edit the
```

user-defined constants
of the active database.

1.112 Record/Add Record

Record/Add Record

Shortcut: +

Adds a new record to the record list of the current project. Each Field contains then its initial content, which is normally nothing. If the list is open a new line will be inserted.

If a
Filter
is active the new record automatically will be declared valid. If you want that new records are filtered correctly you will have to rebuild the index.

This menuitem may only be selected in
record mode
.

See also:
Record/Remove Record

1.113 Record/Duplicate Record

Record/Duplicate Record

Shortcut: A 2

Creates an exact copy of the current record. All initial content attributes will be ignored. Even a field with the Unique Key attribute will contain the old value. That means that two records with the same "unique" key will exist.

1.114 Record/Delete Record

Record/Delete Record

Shortcut: -

Removes the active record from the active index. The data of it will remain in the database file. To finally delete the record data, no other indices may use it. If this is the case,
Project/Reorganize
will remove

the unused record data.

To recover the record, you have to create a new index, which uses the special entry «No Index» as index source. This index will contain all records of the database file. This will not work, if you have used

Project/Reorganize
, though.

This menuitem may be selected only in
record mode
.

If you have selected Security requesters in the user interface settings, you will be queried before proceeding.

See also:
Record/Add Record
,
Record/Delete all Records
,
Project/Reorganize

1.115 Record/Delete all Records

Record/Delete all Records

Shortcut: A @

Removes all records in the current project. The mask will not be deleted by this function.

Record/Delete all Records may be only called in record mode.

Note: Unlike the functions Delete Record and Remove Field, this menuitem does not put up a security requester, if Security-Requesters is activated. However, if the project has been changed, Fiasco will put up a standard Ok-Save-Cancel-Requester.

See also:
Record/Delete Record

1.116 Record/Cut Record

Record/Cut Record

Shortcut: A X

Copies the current record to the clipboard and removes it from the active index. After that, you may use
Record/Paste Record

```
                 to insert it in the
project, again.

    This function may be called only in record mode.

See also: Record/Delete Record,
                Record/Copy Record
                ,
                Record/Paste Record
                ,
Section Clipboard support of Fiasco
```

1.117 Record/Copy Record

```
                Record/Copy Record

Shortcut: A  C

Copies the current record to the clipboard. You may use

                Record/Paste Record
                 to insert it in the project again.

    This function may be called only in record mode.

See also:
                Record/Cut Record
                ,
                Record/Paste Record
                , Section Clipboard
support of Fiasco
```

1.118 Record/Paste Record

```
                Record/Paste Record

Shortcut: A  P

Creates a new record and pastes the contents of the clipboard into that
record. Normally, you should call
                Record/Cut Record
                 or
                Record/Copy Record
                before calling this function.

    This function may only be called in record mode.

See also:
                Record/Cut Record
                ,
                Record/Copy Record
                , Section Clipboard
```

support of Fiasco

1.119 Record/Previous

Record/Previous

Shortcut: Cursor up

Activates the record predeceding the current record. If the current record is the first one, the display will be "beeped".

The order of records is determined by the active index.

The keyboard shortcut correspondents to the structure of the list which displays the previous record over the current record.

This menuitem may be only selected if record mode is active.

See also:
Next, First, Last, Goto, Find previous

1.120 Record/Next

Record/Next

Shortcut: Cursor down

Activates the record after the current record. If the current record is the last in the list, the display will be "beeped".

The order of records is determined by the active index.

The keyboard shortcut correspondents to the structure of the list, which displays the next record under the current record.

This menuitem may be only selected if record mode

is active.

See also:
Previous
,
First
,
Last
,
Goto
,
Find next

1.121 Record/First Record

Record/First Record

Shortcut: Ctrl Cursor up

Activates the first record of the current project.

The order of records is determined by the active
index
.

This item may be only selected in
record mode
.

See also:
Next
,
Previous
,
Last
,
Goto

1.122 Record/Last Record

Record/Last Record

Shortcut: Ctrl Cursor down

Activates the last record of the current project.

The order of records is determined by the active
index
.

This item may be only selected in

record mode
.

See also:
Next
,
Previous
,
First
,
Goto

1.123 Record/Goto...

Record/Goto...

Shortcut: A G

Opens the
goto requester
which can be used to activate a record using its number. Please note that the record number may be changed by adding or deleting records or by using several
indices
.

This item can only be selected in
record mode
.

See also:
Next
,
Previous
,
First
,
Last

1.124 Record/Mark Record

Record/Mark Record

Shortcut: A .

Marks the current record. If a record is marked, it will displayed highlighted in the list and the character "M" will be displayed in the service window.

This item can only be selected in
record mode

.

See also:
Unmark Record
,
Mark all Records
,
Unmark all Records

1.125 Record/Unmark Record

Record/Unmark Record

Shortcut: A :

Deletes the mark of the current record. It won‘t be displayed highlighted anymore.

This item can only be selected in
record mode
.

See also:
Mark Record
,
Mark all Records
,
Unmark all Records

1.126 Record/Mark all Records

Record/Mark all Records

Shortcut: A ,

Marks all records in the current project. Note that the previous marking of all records will be overwritten.

This item can only be selected in
record mode
.

See also:
Mark Record
,
Unmark Record
,
Unmark all Records
,
Toggle all Marks

1.127 Record/Unmark all Records

```
                Record/Unmark all Records

Shortcut: A  ;

Clears the marks of all records in the current project. Note that the
previous marking of all records will be overwritten.

    This item can only be selected in
                record mode
                .

See also:
                Mark Record
                ,
                Unmark Record
                ,
                Mark all Records
                ,
                Toggle all Marks
```

1.128 Record/Toggle all Marks

```
                Record/Toggle all Marks

No Shortcut

Toggles the marks of all records in the current project. A marked record
will be unmarked and an unmarked will be marked. You can restore the
previous marking of the records by calling this menuitem once again.

    This item can only be selected in
                record mode
                .

See also:
                Mark Record
                ,
                Unmark Record
                ,
                Mark all Records
                ,
                Unmark all Records
```

1.129 Field/Fieldtype

```
                Field/Fieldtype

Select the current fieldtype in this submenu. It will be used if you
create fields. The cycle gadget in the
```

```
                service window
                 has the same
function.  These fieldtypes are available:



                String
                   Ctrl S

                Integer
                   Ctrl I

                Float
                   Ctrl F

                Boolean
                   Ctrl B

                Cycle
                   Ctrl C

                Slider
                   Ctrl S

                Date
                   Ctrl A

                Time
                   Ctrl M

                Extern
                   Ctrl E

                Datatypes
                   Ctrl D

                Var String
                   Ctrl V

                Text
                   Ctrl T
Button       Ctrl U

                Bar
                   Ctrl R
```

1.130 Field/Add Field...

```
                Field/Add Field...

Shortcut: Enter

Opens the
```

```
                field requester
                 for the current field type and inserts the
created field at the current cursor position.

    This item can only be selected in
                mask mode
                .

    If there is already a field at the current cursor position nothing
will be done.

    Please note that Enter is also shortcut for
                Edit active Field
                . Enter
creates a new field, if no field is currently active, otherwise, it opens
the requester for editing the active field.

See also:
                Edit active Field
                ,
                Edit named Field
                ,
                Edit Relations
                ,

                Remove Field
```

1.131 Field/Edit active Field...

```
                Field/Edit active Field...

Shortcut: Enter

Opens the
                field requester
                 for the selected field. The field requester can
be used to change several attributes of the field. If certain changes
would cause the lose of data (e.g. changing max chars of a string field
to a lower number), you will be informed about the problem and given the
opportunity to cancel the change. Field types may not be changed this
way. You have to use
                Convert Field
                .

    Because this function edits the selected field, you cannot edit
hidden fields.
                Edit named Field
                 serves for that purpose.

    Please note that Enter is also a shortcut for
                Add Field
                . Enter calls
Add Field if no field is active and otherwise calls Edit active Field.

    This item can only be selected in
```

mask mode
.

See also:
Edit named Field
,
Add Field
,
Edit Relation

1.132 Field/Edit named Field...

Field/Edit named Field...

Shortcut: Shift Enter

Opens a requester with a list of all fields. When you have picked one, the
field requester
for the selected field will be opened. The field requester can be used to change several attributes of the field. If certain changes would cause the lose of data (e.g. changing max chars of a string field to a lower number), you will be informed about the problem and given the opportunity to cancel the change. Field types may not be changed this way. You have to use
Convert Field
.

This item can only be selected in
mask mode
.

See also:
Edit active Field
,
Add Field
,
Edit Relation

1.133 Field/Duplicate Field

Field/Duplicate Field

No shortcut
Makes an exact copy of the active fields. The fields will be placed as near as possible to the original fields. The ID will be copy_of_ FieldID.

1.134 Field/Remove Field

```
                Field/Remove Field

Shortcut: Del

Removes the selected fields. All data in these fields will be lost.
Relations or ARexx scripts which refer to these fields will be not
functional. Attention: The relations or ARexx scripts will not complain
immediately after removing the fields, but at the first activation.

    This item can only be selected in the
                mask mode
                .

See also:
                Edit Field
                ,
                Edit Relations
                ,
                Add Field
```

1.135 Field/Edit Relation...

```
                Field/Edit Relation...

Shortcut: A  &

This item opens the
                relation requester
                , which adds a
                relation
                 to the
current field.

    This item can only be selected in
                mask mode
                .

See also:
                Field/Remove Relation
```

1.136 Field/Remove Relation

```
                Field/Remove Relation

Shortcut: A  0

This item deletes all relation information for the active field. The data
in this field will be written into the normal file.

    This item can only be selected in
                mask mode
```

.

1.137 Field/Create Group

```
                Field/Create Group

Shortcut: A  J

Creates a
                group
                 of the active fields. If you have selected a group and
other fields or groups, these will be joined in one big group.

    Use
                Field/Resolve Group
                 to make the fields independent again.

See also:
                Resolve Group
                , Groups Section
```

1.138 Field/Resolve Group

```
                Field/Resolve Group

Shortcut: A  /

Resolves the active
                group
                . All fields of the group will get independent.
Groups that have been grouped in this group will also be resolved.

See also:
                Create Group
                , Groups Section
```

1.139 Field/Convert Field...

```
                Field/Convert Field...

Shortcut: A  "

Opens the
                convert requester
                 for the selected field. Using convert you may
change the type of a field.

    This item can only be selected in
```

mask mode
.

See also:
Add Field
,
Edit Field

1.140 List/Hide column

List/Hide column

Shortcut: A [

Hides an activated column of the
list
. You activate a column by clicking
in the topmost line of the list which contains the field IDs. After
hiding a column, the columns at the right side of it will be shifted to
the left. The column may be made visible again by using
Show column
.

This item may only be selected if the list window is open.

1.141 List/Show column...

List/Show column...

Shortcut: A]

This item opens a requester which may be used to reveal the columns
hidden with
Hide column
. Fiasco tries to place the columns as near as
possible at their old positions.

This item may only be selected if the list window is open.

1.142 List/Show all columns

List/Show all columns

no shortcut

Makes all columns visible, which have been hidden using
Hide column
.

```
    This item may only be selected if the list window is open.
```

1.143 List/Recalc List

```
                List/Recalc List

Shortcut: A  %

This menuitem calculates all positions and dimensions of the columns in
the
                list
                . Hidden columns are not revealed.

    This item can be compared with Clean up of the Workbench.

    This item may only be selected if the list window is open.
```

1.144 Compare/Find...

```
                Compare/Find...

Shortcut: A  F

Opens the
                search requester
                 which can be used to define search criterions.

    This item is only selectable, if the
                record mode
                 is active and if the
current project contains at least one record.

See also:
                search requester
                ,
                Find next
                ,
                Find previous
```

1.145 Compare/Find next

```
                Compare/Find next

Shortcut: A  >

Activates the next record, which matches with the search criterions,
specified using the
                search requester
```

```
                . You will be informed if no matching
record is found.

    This item is only selectable if
                record mode
                 is active and if the
current project contains at least one record.

See also:
                Search requester
                ,
                Find
                ,
                Find previous
```

1.146 Compare/Find previous

```
                Compare/Find previous

Shortcut: A  <

Activates the previous record, which matches with the search criteria
specified with the
                search requester
                . You will be informed if no matching
record is found.

    This item is only selectable if
                record mode
                 is active and if the
current project contains at least one record.

See also:
                Search requester
                ,
                Find...
                ,
                Find next
```

1.147 Compare/Filter...

```
                Compare/Filter...

Shortcut: A  ~

Opens the
                filter requester
                , which can be used to create
                filter indices
                .

    This item is only selectable if
```

```
                record mode
                 is active and if the
current project contains at least one record.
```

1.148 Compare/Replace...

```
                Compare/Replace...

Shortcut: A  R

Opens the
                replace requester
                ,which can be used for replacing data.

    This item is only selectable if
                record mode
                 is active and if the
current project contains at least one record.
```

1.149 Compare/Count...

```
                Compare/Count...

Shortcut: A  #

Opens the
                count requester
                , which can be used to determine the number of
the records matching with the specified pattern.

    This item is only selectable if
                record mode
                 is active and if the
current project contains at least one record.

    See also:
                Find
```

1.150 Compare/Sort...

```
                Compare/Sort...

Shortcut: A  =

Opens the
                sort requester
                 which may be used to create a sorted index of
the active database.
```

```
    This item is only selectable if
                record mode
                 is active and if the
current project contains at least one record.
```

1.151 Compare/Mark...

```
                Compare/Mark...

Shortcut: A  K

Opens the
                mark requester
                 which can be used to mark specific records that
match a pattern.

    Existing marks will be overwritten; marked records will be unmarked,
if they do not match.

    This item is only selectable if
                record mode
                 is active and if the
current project contains at least one record.
```

1.152 Control/Record Mode

```
                Control/Record Mode

Shortcut: Ctrl F1

This item switches the current project to
                record mode
                 in which records
and the contents of records can be changed. If this mode is active , a
checkmark will be set to the left side of the item.

See also:
                record mode
                ,
                mask mode
```

1.153 Control/Mask Mode

```
                Control/Mask Mode

Shortcut: Ctrl F2

This item switches the current project to
                mask mode
```

```
                 in which the
                mask
                 can
be changed. If this mode is active , a checkmark will be set to the left
side of the item.

See also:
                Mask mode
                ,
                Record mode
```

1.154 Control/Service Window

```
                Control/Service Window

Shortcut: Ctrl F3

This item controls the
                service window
                . If it is checked the service
window is open. The service window makes the most important record- and
mask-operations easier and displays some status information.

    The service window serves globally for all projects.
```

1.155 Control/List Window

```
                Control/List Window

Shortcut: Ctrl F4

This item controls the
                list window
                , if it is checked, the list is open.

    Each project may have its own list window.
```

1.156 Control/ARexx-Debug

```
                Control/ARexx-Debug

No Shortcut

This activates a special debug mode of Fiasco for the
                ARexx interface
                . If
Fiasco commands fail, Fiasco will create a requester that contains more
detailed information about the error.
```

1.157 Settings/Databases...

```
                Settings/Databases...

No shortcut

Opens the
                database settings requester
                . You may use this requester to edit
certain Fiasco database options.
```

1.158 Settings/User Interface...

```
                Settings/User Interface...

No shortcut

Opens the
                user interface settings requester
                , which controls certain
elements of Fiasco's user interface, such as service window, field
activation, etc.
```

1.159 Settings/User Menu...

```
                Settings/User Menu...

No shortcut

Opens the
                user menu requester
                 which can be used to define user menus.
```

1.160 Settings/Display...

```
                Settings/Display...

no shortcut

Opens the
                display requester
                 which can be used to specify display options
for Fiasco. You can select whether Fiasco will open its windows on a
public screen or on its own custom screen. Furthermore, you may select
fonts for the screen and the mask.
```

1.161 Settings/External Programs and Paths...

```
                Settings/External Programs and Paths...

no shortcut

Opens the
                external programs and paths requester
                 that lets you specify
programs that may be called by Fiasco and paths for various uses.
```

1.162 Settings/Save Settings

```
Settings/Save Settings

Saves the current program settings in the files "env:fiasco.prefs" and
"envarc:fiasco.prefs". The settings "survive" rebooting.
```

1.163 Settings/Save Settings as...

```
Settings/Save Settings as...

Saves the settings in a file, which has been specified with a file
requester. If you save the file in "env:", the settings won't survive a
reboot. If you save them only in "envarc:", they will only become active
after rebooting because Fiasco searches for its current settings in
"env:" and nowhere else.
```

1.164 Settings/Load Settings...

```
                Settings/Load Settings...

Loads and uses a specified settings file. To use them also after reboots
you should select
                Save Settings
                 to write them to "env:" and "envarc:".
```

1.165 The Print Window

```
                The Print Window

The print window can be opened with Project/Print. You can control
Fiasco's print function from there.  More on the print function in the

                print section
                .   The print window contains these menus:
```

```
Menu                Shortcut
Project

                Erase
                  A Z

                Open...
                  A O

                Get from Mask
                  A M

                Get from List
                  A L

                Save
                  A S

                Save as...
                  A A

                Print
                  A P

                Options...
                  A T

                Exit
                  A Q

Element

                Element Type   »
                        Field      Ctrl F
        Text       Ctrl T
        Formfeed   Ctrl O

                Add...
                  Return

                Edit...
                  Return

                Duplicate
                Remove            Del

Control

                Edit Head
                  A H

                Edit Body
                  A B

                Edit Foot
                  A F
```

1.166 Project/Erase

Project/Erase

Shortcut: A Z

Removes all elements from the print window. After using this menuitem the print window will be empty.

1.167 Project/Open...

Project/Open...

Shortcut: A O

Opens an file requester and reads the print layout from the selected file. The old data will be overwritten.

1.168 Project/Get from Mask

Project/Get from Mask

Shortcut: A M

This menuitem tries to fake the project mask's layout in the print window. The old print mask will be overwritten.

1.169 Project/Get from List

Project/Get from List

Shortcut: A L

This menuitem tries to fake the list's layout in the print window. The old print mask will be overwritten.

1.170 Project/Save

Project/Save

Shortcut: A S

Select Save if you want to write the current print mask to a file on
disk. This is the file Project_Name.fpr, if you haven't selected another
using Open or Save as. The file name is displayed in the window title bar
of the print window.

1.171 Project/Save as...

Project/Save as...

Shortcut: A A

Select this menuitem if you want to save the print mask in another file
as the currently selected. The file name is displayed in the window title
bar of the print window.

1.172 Project/Print

Project/Print

Shortcut: A P

This menuitem creates the print-out of the project using the active print
mask. The exact function of this menuitem is dependent on the settings
made in the
print options requester
.

1.173 Project/Options...

Project/Options...

Shortcut: A T

This menuitem opens the
print options requester
. Some print mask-specific
options may be edited here.

1.174 Project/Exit

Project/Exit

Shortcut: A Q

This menuitem closes the print window. The active print mask will be
deleted from memory.

You may also use the window's close gadget for this purpose.

1.175 Element/Element Type

Element/Element Type

Use this submenu to select the active element type. This type will be used by the subsequent
Element/Add
calls.

These element types may be selected:

- Field (Ctrl F)
- Text (Ctrl T)
- Formfeed (Ctrl O)

More information can be found in the
print chapter
.

1.176 Element/Add...

Element/Add...

Shortcut: Return

Creates a new element at the cursor position. The element will be of the type set in the
Element/Element Type
submenu. If the element type supports a requester, the
element requester
will appear.

More information can be found in the
print chapter
.

Note that this menuitem has the same shortcut as
Element/Edit
. This shortcut will Add if no element is active and Edit if an element is active.

1.177 Element/Edit...

```
                Element/Edit...

Shortcut: Return

Opens the
                element requester
                 for the active element. Elements can be made
active using the mouse or the cursor keys.

    Note that this menuitem has the same shortcut as
                Element/Add
                . This
shortcut will Add if no element is active and Edit if an element is
active.
```

1.178 Element/Duplicate

```
Element/Duplicate

No shortcut

Duplicates the active element.
```

1.179 Element/Remove

```
Element/Remove

Shortcut: Del

Deletes the active element. You may only recover this element using a
saved version of the print mask.
```

1.180 Control/Edit Head

```
                Control/Edit Head

Shortcut: A H

Selects the head part of the print mask for editing. The head part will
be printed before any other data. It may not contain field elements. This
menuitem, Edit Body and Edit Foot are mutually exclusive.

    See the
                print chapter
                 for more information.
```

1.181 Control/Edit Body

Control/Edit Body

Shortcut: A B

Selects the body part of the print mask for editing. The body part will be printed for each record. It may contain references to fields in the form of field elements. These references will be substituted while printing by the field contents. This menuitem, Edit Head and Edit Foot are mutually exclusive.

See the print chapter for more information.

1.182 Control/Edit Foot

Control/Edit Foot

Shortcut: A F

Selects the foot part of the print mask for editing. The foot part will be printed after any other data. It may not contain field elements. This menuitem, Edit Body and Edit Head are mutually exclusive.

See the print chapter for more information.

1.183 All Requesters

All Requesters

Requesters are used by Fiasco to get information required for certain operations. Normally, the requesters are created after selecting Fiasco menuitem. So called EasyRequesters, which are used by Fiasco to request a simple choice are not explained here, because they are generally easy to understand and are described in function specific sections. Most requesters can be controlled by using the keyboard. The shortcuts, which are marked with an underscore, are usually single characters without a qualifier.

The gadgets at the lower bottom of a requester are usually for proceeding. Normally, the left-most is a positive response (Ok), while the right-most is a negative response (Cancel). Enter is the shortcut for the positive response. The gadget is additionally emphasized. Esc is the shortcut for the negative response.

Import Requester

Export Requester

Reveal Project Requester

Project Options Requester

Statistic Requester

Indices Requester

New/Edit Index Requester

Functions Requester

Constants Requester

Goto Requester

Field Requester

Popup Gadget Requester

Convert Field Requester

Relation Requester

Formula Requester

Show Column Requester

Search Requester

Filter Requester

Replace Requester

Count Requester

Mark Requester

Sort Requester

Database Settings Requester

User Interface Settings Requester

User Menu Requester

Display Settings Requester

External Programs and Paths Settings Requester

Print Options Requester

Print Element Requester

1.184 Import Requester

```
                Import Requester

The import requester is the GUI interface to the
                import function
                 of
Fiasco. Import allows Fiasco to read data from other database programs.
Usually this cannot be done directly, but the foreign database has to
"export" the data. You may specify various parameters for importing, so
you should be able to read nearly all export formats into Fiasco. The
Fiasco distribution contains several predefined import formats, which can
be loaded using the Load button at the bottom of the import requester.

    The values that may be typed in the gadgets of the import requester
are described in the Import/Export section of this document.


File: Specify here the file that contains the data to import. You may
   use the picker button at the right side to select it using a file
   requester.

View: Click here, if you want to view the contents of the file. Fiasco
   will start asynchronously More or MultiView, if available.

Records/Start: Enter the start characters for records here. Default:
   Empty.

Records/End: Enter the characters at the end of a record here. Default:
   Empty.

Records/Separator: Enter the characters between two records here.
   Default: \n.

Fields/Start: Enter the characters here that fields start with. Default:
   ".

Fields/End: Enter the characters here that fields end with. Default: ".

Fields/Separator: Enter the characters between two fields here. Default:
   \t.

Misc/Skip Lines: Enter introducing characters for remarks here. Default:
   Empty.

Misc/Start skip: Enter the number of lines here that will be skipped at
   the start. Default: 0.

Misc/Max fields: Enter the maximum number of fields in a record here. Can
   also be used if record separators are missing. Default: 100.

Options/First record contains IDs: Activate this gadget if the first
   record of the file contains the IDs of the fields in the project. They
```

will be used by Fiasco then instead of generic IDs.

Options/Append new fields: Activate this, if you want Fiasco to create new fields for your data and not to use existing fields. If you have an entirely empty project you should activate this option. If this option is not active, Fiasco will first try to associate the fields by their IDs. If there are not enough free fields, Fiasco will create new fields nethertheless.

Options/Overwrite old project: Removes the old data in the current project window. If you do not select this, you data will be appended in some manner to the existing project.

Ok: Starts the import process. Note that Fiasco may run out of memory due to bad structure parameters and overly large files. Programs that have problems with low memory should not run during this process.

Save: Saves the current settings in a specified file.

Load: Reads the settings from a specified file and sets them up in the requester.

Cancel: Closes the requester without any further action.

1.185 Export Requester

Export Requester

The export function provides the ability to share data from Fiasco with other databases that cannot read the normal format of Fiasco databases. See the
Import/Export section
of this document for more information about this mechanism.

File: Specify the name of the file here that the data shall be written to. If a file already exists with this name it will be overwritten.

Records/Start: Enter the start characters for records here. Default: Empty.

Records/End: Enter the characters at the end of a record here. Default: Empty.

Records/Separator: Enter the characters between two records here. Default: \n.

Fields/Start: Enter the characters here that fields start with. Default: ".

Fields/End: Enter the characters here that fields end with. Default: ".

Fields/Separator: Enter the characters between two fields here. Default:

```
   \t.

Options/First record contains IDs: Activate this gadget if you want
   Fiasco to write the field IDs in the first record.

Options/Marked records only: Activate this gadget if you want Fiasco to
   write only marked records.

Listviews/Entry Separator: Enter the characters between two entries of a
   listview field here. Default: |.

Ok: Click here to start the export process.

Save: Saves the structure parameters to a selected file.

Load: Loads the structure parameters from a selected file.

Cancel: Closes the requester without any further action.
```

1.186 Reveal Project Requester

```
                Reveal Project Requester

This requester is used to reveal a Fiasco project which has been hidden
using
                Project/Hide
                . The reveal project requester can be opened using

                Project/Reveal
                 or by doubleclicking on the Fiasco Workbench icon.

Project: This is a list of all hidden Fiasco projects. To reveal a
   project, select it and click on Ok or doubleclick on it.

Ok: Closes the requester and opens the selected project.

Cancel: Closes the requester without any further action.
```

1.187 Project Options Requester

```
                Project Options Requester

The option requester contains project-related settings. It may be opened
using the menuitem
                Database/Options
                .
```

Author: You can use this field to enter your own name! It will be stored at the beginning of the project file.

Annotations: Yet another gadget for undefined use. You may store any notes here, for example a version string (with $VER: at the beginning). It will be written to the project file just before the author's name.

Mask Stretching X / Y: These values are added to the width or height of the cursor. The effect of this operation is a stretching of the mask in the X- or Y-direction. More on stretching here.

Windows/Open list on startup: Activate this gadget to instruct Fiasco to open the list window when this project is loaded.

Windows/List position fixed: If this gadget is active, Fiasco remembers the position of the list window when saving the project. The list window will open at this position thereafter.

Windows/Mask position fixed: If this gadget is active, Fiasco remembers the position of the mask window when saving the project. The mask window will open at this position thereafter.

Windows/Close Gadget: If this gadget is active, the mask window of this database will have a close gadget, that can be used for closing the database.

Windows/Depth Gadget: If this gadget is active, the mask window of this database will have a depth gadget, that can be used for depth-arranging the window.

Windows/Drag Bar: If this gadget is active, the mask window of this database will be draggable on the screen.

Windows/Size Gadget: If this gadget is active, the mask window of this database will have a size gadget for sizing the window and proportional gadgets for scrolling the contents. Otherwise, the window borders will be thin.

Records/Max. time for reading: Use this gadget to set the maximum time for disk access when records have to be read from disk. The time is measured in microseconds. 65000 is a good value here. It reads a relatively high number of records and is not disturbing during record changes. Higher values will lead to longer pauses during record changes and more read records. Lower values will make records changes quicker and fewer records will be read. However, Fiasco will always read at least one record (when available).

Records/Max. memory: Set here the maximum amount of RAM, measured in kilobytes, to be used by this Fiasco database for records. This is only a standard value because Fiasco will not always check the value. Lower values will lead to more disk accesses, because more records will have to be re-read. The default for this is 200. You may control the actual RAM usage in the statistic requester.

.

ARexx Scripts/Database Startup: An ARexx script specified here will be
   executed immediately after this database has been opened. Thus, it may
   serve for initialization purposes, etc. Note: The

               User Interface Settings
                contain a control to disable this setting.

ARexx Scripts/Database Shutdown: An ARexx script specified here will be
   executed before this database will be closed. Thus, it may clean up
   things, etc. Note: The
               User Interface Settings
                contain a control to
   disable this setting.

CD ROM mode: If active, this Fiasco database will be in CD ROM mode.
   This mode is for better use of Fiasco databases on CD ROMs (or other
   read only media). It has two effects: All mask fields will be read
   only and all indices created by the filter and sort functions will be
   written to T:. The operation of Save and Save As will remain
   unchanged, in order allow to save a database with CD ROM mode for
   preparing CD ROM distributions.

1.188 Statistic Requester

               Statistic Requester

Apart from getting information, you cannot do much in the statistic
requester. It only serves to display information about certain
statistical values of the active Fiasco database.

Database: The name of the database is displayed here.

Index: The name of the active index of the database is displayed here. If
   no index is active, -- will be displayed here. This is the case in a
   newly created database or a database that is still in Fiasco 1.x
   format.

Records on disk: This is the number of records in the database file.

Records in index: This is the number of records in the active index.

Added records in RAM: This is the number of records that have been added
   to the database since it was last saved.

Disk-loaded records in RAM: This is the number of records that Fiasco has
   read from disk and which are now held in RAM. This value depends on
   the Max. memory setting in the
               project options requester
                and how many
   records Fiasco has been able to read so far.

All records in RAM: This is the number of all records in RAM.

Changed records in RAM: This is the number of records in RAM, which have
   been changed since the databases was last saved. This includes both
   disk-loaded records which have been changed and added records (which
   are always 'changed').

Deleted records in RAM: Even though it sounds paradoxical, this is the
   number of disk-loaded records which have been deleted since the last
   saving of the database. Fiasco holds disk-loaded records which have
   been deleted in RAM in order to be able to modify the index properly
   during the next saving. Added records, which are deleted later, are
   not held in RAM by Fiasco.

Free RAM: This is the total available RAM in bytes.

RAM size of one record: This is the amount of RAM that is required by one
   record. This is only an estimated value, because Fiasco uses pools for
   memory organization.

RAM size of all records: This is simply the RAM size of one record
   multiplied with All records in RAM.

Flush Records: Tries to remove as many records as possible from RAM.
   Records which are counted under Added records, Changed records or
   Deleted records cannot be flushed. You will have to save the database
   before you can flush these.

Ok: Closes the requester.

1.189 Indices Requester

                Indices Requester

The indices requester can be used to select, create new, edit or delete

                indices
                . It can be opened with the menuitem Database/Indices.

Index Files: Select the index here that will be used by the next
   operation. Initially, the active index will be selected here. This
   list will be empty if the database has never been saved yet or is
   still in Fiasco 1.x format. Doubleclicking on one entry will call the
   Edit function.

New: Opens the
                new/edit index requester
                , which will create a new index
   when confirmed with Ok.

Edit: Opens the
                new/edit index requester

for the selected index.

Delete: Deletes the selected index. If the selected index is the last one you will not be able to delete it.

Ok: Closes the requester and executes mall changes specified in this requester, i.e. new indices will be built, edited indices will be changed and indices, which have been removed from the list will be deleted. The selected index will be made active.

Cancel: Closes the requester and ignores all changes specified in this requester. The indices will be the same as they were before opening this requester.

1.190 New/Edit Index Requester

New/Edit Index Requester

This requester is used to specify certain options for indices . It can be opened in two ways: First, by calling New in the indices requester . When you have opened the index requester this way, it will have additional controls for index creation. The second way is to Edit an index in the indices requester.

Name: This gadget is only present in the new index requester. It takes the name of the index to be created.

Index Source: This gadget is only present in the new index requester. Select an index here to be used as the model for the new index. The new index will contain only the records that the selected index contains. If you don't use Index Sorting, the order of the records will be copied. Thus, if you use neither Index Sorting nor Index Filter a copy of the selected index will be made. The special entry «No Index» stands for a virtual index which contains all records in a database.

Index Filter: Clicking on the button will open the filter requester which can be used to specify the criteria which Fiasco will use to decide whether a record should be present in the index or not. This is used when a new index is built and when a new record is added to the database while another index is active and Automatically add new Records is active. In other cases, the filter information is not used, i. e. a record will be always added to the index when it is added while this index is active.

Index Sorting: Clicking on the button will open the

```
                sort requester
                . You
   may specify by which fields the records of the index will be sorted.
   Activate the checkbox if you want Fiasco to keep the index sorted.
   Thus, Fiasco will always take care that after adding or editing
   records the index is properly sorted.

Automatically add new Records: Activate this checkbox if you want Fiasco
   to add all records which are added to the database to this index after
   applying the Index Filter and Index Sorting. Normally, only records
   are added to an index if the index was active when the record was
   added.

Ok: Accepts the settings made in this requester. However, the settings
   will first really become active after confirming the
                indices requester
                   with Ok.

Cancel: closes the requester and ignores the settings.
```

1.191 Functions Requester

```
                Functions Requester

In the functions requester, you may specify

                user-defined functions for formulas
                . The requester may be opened with the
menu item Database/Functions.

    Important Note: Recursive programming is not possible with these
user-defined functions.


Functions: This list shows all functions defined for the active
   project so far. If you select one, you may edit it with the other
   gadgets.

Name: You have to specify the header for a function here. The header is
   built of the function name and the function arguments enclosed in
   parenteses. If the function takes no arguments, you have to specify an
   empty pair of parenteses. Examples: min(a, b) or recordsum().

Definition: Takes the definition of the function. This is a normal
   formula. The arguments specified in the function header may be
   accessed like normal fields from this formula. Example for function
   header min(a, b): a < b ? a : b.

New: Creates a new function and activates it for editing.

Delete: Deletes the currently selected function.

Ok: Checks all function headers and definitions for errors. If no errors
```

were encountered, the new user-defined functions will become active.
If there are already formulas defined for the database, you will be
asked, whether you want to recompile these formulas.

Cancel: Closes the requester without any further action.

1.192 Constants Requester

Constants Requester

In this requester you may specify
user-defined constants for formulas
. It
may be opened with the menu item Database/Constants.

Constants: This list shows all constants defined for the active
project so far. If you select one, you may edit it with the other
gadgets.

Name: You have to specify the name of a constant here. For constants
names the same restrictions as for field IDs in formulas apply.

Definition: Specify here the value for the constant. It may be a numeric
value or a string enclosed in qoutes.

New: Creates a new constant and activates it for editing.

Delete: Deletes the currently selected constant.

Ok: Checks all constant names and definitions for errors. If no errors
were encountered, the new user-defined constants will become active.
If there are already formulas defined for the database, you will be
asked, whether you want to recompile these formulas.

Cancel: Closes the requester without any further action.

1.193 Goto Requester

Goto Requester

The goto requester is one of the simplest of all the Fiasco requesters.
It may be opened with
Record/Goto
and makes it possible to activate a
record using its number. Please note that record numbers may vary with
different
indices

or after adding or deleting records.

Record Number: Takes the number of the record.

Ok: Activates the record with the number.

Cancel: Oh sorry, I just forgot... %-)

1.194 Field Requester

Field Requester

The field requester can be used to change the attributes of a field. Each fieldtype has a different field requester because the gadgets of the field requester represent the supported attributes of each fieldtype. The supported attributes are listed with the
documentation of each field
. The
field requester will be opened if you call
Add Field
,
Edit active Field
or
Edit named Field
or double-click on a field.

If you proceed with Ok, all values will be checked for validity. If one value cannot be used by Fiasco, a requester will explain the problem.

A small summary of the conditions: (presuming that these attributes exist)

- There must be an ID.
- Max Chars must be > 0.
- Width or height must be > 2.

If dimension values cannot be used, because other fields are too near the field, another requester appears with Shift, Squeeze and Cancel gadgets. Cancel does nothing other than return to the field requester. Squeeze makes the field fit the selected space. Shift moves the field to the left to make it fit. It is not always possible to Shift.

If you change an already existing field that stores its contents in strings and supports MaxChars (currently
String
,
Extern
and
Datatypes
),

an additional control is implemented. If you change Max Chars to a value which does not allow to keep all strings in their original length (that means, some strings are longer), you will be asked if you want to truncate these strings or keep the old value.

1.195 Popup Gadget Requester

Popup Gadget Requester

This requester sets the options for the popup gagdet field attribute. It may be only opened from the field requester with the button Predefined Values.

Popup Gadget: Here you can activate a popup gadget for the field and select its working mode.

Predefined Values/Value: This list is only active if you have selected Select predefined value at Popup Gadget. Then you may specify values here that can be later selected with the popup button. These values will be also used in the field requester.

Predefined Values/New: Creates a new entry in the list above.

Predefined Values/Delete: Deletes the active entry in the list above.

ARexx Script: This gadget is only active if you have selected Execute ARexx Script. Then you may specify an ARexx script that will be executed when the popup gadget is selected.

Ok: Accepts the settings.

Cancel: Closes the requester without any further actions.

1.196 Convert Field Requester

Convert Field Requester

The convert field requester can be used to change the type of one field without the need of an ARexx script.

Field ID: This text gadget displays the ID of the field that will be converted. Please check here to see if you have called Convert Field for the correct field.

Old Type: Displays the current type of the field.

New Type: Select the new type of the field here. Please note that conversions between certain field types may cause a lose of data. Consult the

```
                field documentation
                 for more information on this topic.

alternative format: Select this checkbox to active an output-format which
   differs from the normal format. Please see the
                field docs
                 whether this
   gadget has any effect and if so, what.

Ok: Starts the conversion and then closes the requester. If the field
   should be in use by other Fiasco systems, a requester will show you
   the respective systems and give you the choice to continue and to
   remove these uses or the cancel the operation.

Cancel: Simply closes the requester.
```

1.197 Relation Requester

```
                Relation Requester

This is the main interface for
                relation handling
                 in Fiasco. It may be
opened using
                Field/Edit Relations
                .

Type: Set here the type for the relation.  See the
                relation types
                 section
   for descriptions of the types.   The available types depend on the
   type of the selected field. The selected type also influences the
   availability of the other fields as key and real data fields.

Local Key: Select the key in the current project.

Local Real: Displays the ID of the field, whose relations are just now
   edited.

Remote Key: Use this listview to select the key-field in the project that
   has been specified under Related File. This listview displays only
   fields that can contain the key (with same type).

Remote Real: Use this listview to select the field of the project that
   has been specified under Related File and which is supposed to be the
   counterpart of Real here. This field is used to read the Data, which
   will be displayed in Real here. This listview only displays also
   fields that look as if they could contain the data (type and max chars
   must be equal).

Related File: Select the project file here relative to the directory of
   the current project which contains the informations.
```

Ok: Loads the relations. If any errors occur while loading, the requester
   will be activated again, otherwise it will return to the main window.

Cancel: closes the requester without any further action.

1.198 Formula Requester

Formula Requester

This requester serves as a tool for creating formulas for Fiasco. It may
be opened from the field requester and from the search requester. After
you have opened it, you may create a formula. When you exit it with Ok,
the formula will be copied into the calling requester.

Operators: This listview displays all operators supported by Fiasco.
   Clicking on one will insert it at the current cursor position in the
   formula string gadget below.

Functions: This listview displays all built-in and user-defined
   functions. As with all listviews in this window, clicking on one
   function will insert it at the current cursor position in the formula
   string gadget.

Fields: Displays all fields of the respective database. Clicking on one
   field will insert it into the formula.

Constants: Displays all constants of the database. Clicking on one
   constant will insert it at the current cursort position in the formula
   string gadget.

Formula string gadget: Displays the formula to be edited. You may edit it
   manually or with the listviews above. Clicking on one entry in any of
   the listviews will insert it at the current cursor position.

Ok: Checks the formula for errors. If no errors are found, the formula
   will be copied into the calling requester and the requester is closed.

Cancel: Closes the requester without any further action

1.199 Show Column Requester

                Show Column Requester

This requester, which may be reached with
                List/Show column
                , displays the
currently hidden columns in the
                list

. If you select a column and click on
Ok, that column will be inserted in the list at its old position.

Field: All hidden columns are displayed here. Select the column here that
you want to be revealed.

Ok: Inserts the column and re-displays the list.

Cancel: Closes the requester.

1.200 Search Requester

Search Requester

The search requester is the main interface to Fiasco's
search function
.
It may be opened with the menu item Compare/Search. However, there are
many other functions that use the same or a slightly differnt requester.

Mode: Select here the mode you want to use for searching. Each mode has
its own GUI displayed below in the requester. By Fields is the classic
mode in which you may specify for fields patterns. By Formula allows
you to use a
formula
to do a search.

Name: You may specify here a name under which the current search criteria
will be stored. With the select button at the right side of the gadget
you can later select these criteria again.

Mode By Fields:

Fields: This list displays all fields of the active database. Clicking on
a field will cause it to be put in the Search for listview below.
There you may specify a pattern for the field.

Search for: Here all the fields are listed which will be searched. When
you activate a field here, you may specify a search pattern and other
paramters for that field in the gadget group below. To insert a field
in this list, simply click on it in the Fields listview. To remove it
again, activate it and click on the Delete buttom.

Pattern: This string gadget takes the
search pattern
for the currently
selected field in the Search for listview. Clicking on the picker
button at the right side of the gadget will open a list with some
values that can be used as a pattern. That may be the predefined
values, the labels of a cycle field, etc.

Delete: Removes the currently selected field in the Search for listview.

```
Blurred Search: This gadget allows you to activate the
                blurred search
                 and
   to control the tolerance. 0 searches only for exactly matching
   entries, 100 searches for almost all entries.

Mode By Formula:

Formula: This gadget takes the formula for searching. A record will be
   considered matching the formula when the formula returns a true value,
   i.e. a value not equal 0. The picker button at the right side of the
   gadget opens the
                edit formula requester
                 that helps with creating a
   formula.

Response gadgets:

Next: Initiates the search for the next matching entry and activates it.

First: Searches for the first matching entry.

Previous: Searches backwards for the next matching entry.

Cancel: Closes the requester without any further action.
```

1.201 Filter Requester

```
                Filter Requester


                Filters
                 offer the possibility of creating an overview of a group of
records. A filter creates the impression of a database that consists only
of the matching records. Fiasco uses
                indices
                 to create a filter. Like the

                sort requester
                , the filter requester is used in two contexts. If you open
the filter requester using Compare/Filter, the filter function will build
and activate a filtered index according to the criteria set in the filter
requester. To turn the filter off, select the menu item

                Database/Previous active index
                . Another way would be to select the old
index in the
                indices requester
                . The created index will be based on the
active index, thus it will only contain records that the active index
contained and that match the filter criteria.

    The second way to open the filter requester is by calling Index
```

Filter in the
new/edit index requester
. The filter criteria set here will
be used only for building the filter index and when Automatically add new
Records is active, a record is added and another index is active.

Because filters use indices which are created once, records added to
the database while a filter is active will be displayed regardless of
their contents. The same rules apply to record changes.

As the filter requester is almost identical to the
search requester
,
only the different or new gadgets are explained here.

Index: This gadget is only present if you have opened the requester
using Compare/Filter. Enter the name of the index to be created here.
If the index already exists the index will be overwritten after you
have acknowledged Fiasco's warning.

Response gadgets:

Ok: accepts the settings. If the filter requester has been opened with
Compare/Filter the index will be built immediately.

Cancel: closes the requester without any further action.

1.202 Replace Requester

Replace Requester

Most parts of the replace requester are the same as in the

search requester
. Only the changed or new gadgets are explained in this
section.

The replace requester may be opened with the menu item
Compare/Replace
.

Replace/Fields: Clicking on one field in this listview will insert it
in the Replace listview. This is very similar to the Fields and Search
for listviews above.

Replace/Replace: This listview contains all fields in which a new content
will be copied when a matching record is found. To insert a field in
this list, click on it in the Fields listview. To remove it again,
select it in the Replace listview and click on Delete. While a field
is active in this listview, you may edit the replacement settings
below.

Replace/Mode: Use this cycle gadget to either select Replace by value to
   insert a fixed value in the selected field above of matching records
   or Replace by formula result to insert a result of a formula.

Replace/Replacement: You must either enter here a fixed value as
   replacement or a formula that calculates the replacement. This depends
   on the setting of the Mode cycle gadget above.

Replace/Delete: Removes a selected field from the Replace listview.

Replace/Confirm: If you want to be asked for every replacing operation,
   you should select this gadget.

Response gadgets:

All: Starts the search and scans all records in the active index. If you
   have selected Replace/Confirm, you are able to select for each
   matching individually, whether you want to replace the value or not.
   You may even break the whole search.

Next: Only replaces the value in the next matching record and activates
   it.

Cancel: Closes the requester without any further action.

1.203 Count Requester

Count Requester

This requester lets you count records, which match with a patterns. More
on counting
here
. You can open this requester using
Compare/Count
.

As the count requester is almost identical to the
search requester
,
only the different gadgets are explained here.

Response gadgets:

Ok: proceeds and counts the matching records. The number will be
   displayed at the end.

Cancel: closes the requester without any further action.

1.204 Mark Requester

Mark Requester

Fiasco's mark function makes it possible to distinguish specific records. The purpose of the mark requester is to mark all records that match a specific pattern. The mark requester may be opened with
Compare/Mark
.

As the filter requester is almost identical to the
search requester
,
only the different or new gadgets are explained here.

Response gadgets:

Ok: marks the matching records. The old record marks will be lost!

Cancel: closes the requester without any further action.

1.205 Sort Requester

Sort Requester

The sort requester is used for specifying options for sorting a database based on fields.
Indices
are used by Fiasco to control the order of records. The sort requester can be opened in two ways. First, you may call
Compare/Sort
which allows you to quickly sort a Fiasco database. This function builds a new index and activates it. The name of the index may be specified in the Index gadget. This index will be based on the active index, that means it will contain only those records which the active index contains.

The other context in which the sort requester is used is the

index requester
. The sort requester will be opened when you click on the Index Sorting gadget in the index requester. You may edit the sort options for automatic sorting of a particular index here.

Index: This gadget only exists if the requester has been opened with Compare/Sort. This will be the name for the newly created index. If this name already exists the old index will be overwritten. However, Fiasco will warn you in that case before overwriting the index.

```
Fields: A list of all fields of the active project is displayed here.
   Clicking on a field will put it in the Sort by list.

Sort by: This list displays the fields by which the project will be
   sorted. The topmost field has the highest priority while sorting, i.e.
   most data will be sorted according to this field. If there are entries
   which contain equal data the following fields will be used. Use Delete
   to remove a field from this list. The arrows can be used to move the
   field in the list.

Descending: Select this gadget to sort the data from high values to low
   values (i.e. Z, Y, X, ..., C, B, A)

Use Locale Settings: If you select this gadget, Fiasco will use the
   current locale settings to sort alphabetical data. This will guarantee
   the correct sorting of national characters like the German umlauts ä,
   ö, ü, etc. However, please note, that these sorting rules differ from
   country to country. Thus, a database sorted with one locale country,
   may be sorted incorrectly if you change the country.

Ok: accepts the settings. If the sort requester has been opened with
   Compare/Sort, the index will be immediately built.

Cancel: Closes the requester without any further action.
```

1.206 Database Settings Requester

```
                Database Settings Requester

You may control certain database related functions of Fiasco here. This
requester may be opened by using
                Settings/Databases
                . It replaces some of
the Settings menuitems of Fiasco 1.2.

Relations/Write Relations: If this option is active, Fiasco will also
   write
                relations
                 back in their "there" projects. Otherwise, changes
   made in these fields will be lost. This option should be only active
   if Update Relations is also active, or when you always call

                Project/Reload Rels
                 before saving. Otherwise you risk overwriting data
   in the "there" project by invalid data in some fields of the "here"
   project.

Relations/Update Relations: This checkbox determines whether
                relations
                   are updated immediately after the input of a new key. This ↩
                       requires
   disk accesses which may become annoying if Fiasco has to read the data
   from a floppy disk. If you deactivate this checkbox you should also
```

```
   deactivate Write relations, because there may be invalid data in the
   project which would be written into the "there" file. If you want to
   see the changes, you can update the relations using
                Project/Reload Rels
                   .

Miscellaneous/Use ‘*’ as Synonym for ‘\#?’:  Select this option to
   activate the support of the asterisk as a valid search pattern. The *
   then has the same meaning as #?.

Ok: Closes the requester and accepts the changes made in it.

Cancel: Closes the requester and ignores any changes.
```

1.207 User Interface Settings Requester

```
                User Interface Settings Requester

This requester allows you to control certain parts of Fiasco’s user
interface. It may be opened using
                Settings/User Interface
                . It replaces
some of the Settings menuitems of Fiasco 1.x.

Service Window/Open on Startup:  If this checkbox is checked the

                service window
                 will be opened automatically when Fiasco is started.

Service Window/Fixed Position:  Activate this option to make Fiasco open
   the service window in a given position. Otherwise, Fiasco will search
   for a free place on the screen to open the service window.

ARexx Scripts/Program Startup:  An ARexx script specified here, will be
   automatically executed after Fiasco has been started.

ARexx Scripts/Program Shutdown:  An ARexx script specified here, will be
   automatically executed before Fiasco will be shut down.

ARexx Scripts/Database specific startup/shutdown scripts:  You may
   control here, wether startup and shutdown scripts specified in the

                project options
                 should be executed or not. Such scripts may be
   specified locally for each database. Execute automatically always
   executes these scripts, Ask before executing opens a requester before
   executing, which leaves you the choice, whether you want to execute
   the script. Execute never does not execute these scripts.

Miscellaneous/Create Icons: If this option is active, Fiasco will create
   icons while saving projects.

Miscellaneous/Speech: Activate this option if you want Fiasco to use the
```

narrator.device to "speak" certain messages.

Miscellaneous/Ask for database when started empty: If active, Fiasco will open a file requester whenever it is started and no databases were given to be loaded (either by project icons or by specification on the command line).

Miscellaneous/Security Requesters: If this checkbox is active, Fiasco will warn you before deleting any fields or records. This can prevent the erroneous deletion and loss of data.

Miscellaneous/Cycle trough fields with Enter: Activate this option to instruct Fiasco to activate the next field when one field has been left by pressing Enter. This conforms to the Fiasco 1.x usage. Starting with Fiasco 2.0, you may use the Tab key to cycle through the fields.

Miscellaneous/Automatically activate first field: If this option is active, Fiasco will automatically activate the first field when a new record is selected.

Ok: Closes the requester and accepts the changes made in it.

Cancel: Closes the requester and ignores the changes.

1.208 User Menu Requester

User Menu Requester

Fiasco has the ability to create own menuitems and to put CLI programs or ARexx scripts behind them. The defined items also may be selected with the F-Keys as well as with the mouse. F1 to F10 correspond to the first ten items, Shift and F1 to F10 correspond to the items 11 to 20. If you want to define more than 20 items, you will have to select the additional items with the mouse. Furthermore, Intuition limits the number of definable items to 63.

The items defined in this requester may be saved using

Settings/Save Settings
.

Items: This is the list of all existing menu items. You can Add one item and Delete one. < and > serve to change the position of the item in the menu.

Type: allows you to indicate whether the item will call a program or an ARexx script.

Command: Specify the program or the ARexx script here that will be executed.

1.209 Display Settings Requester

```
Display Settings Requester

This requester controls the display elements of Fiasco. You can open your
own screen for Fiasco and choose the fonts for the custom screen and for
the mask.

Screen/Screentype: Select here, whether you want to use a public screen
   or an own custom screen.

Screen/PubScreen Name: Specify here the name of the public screen, you
   want Fiasco to open its windows on. This has only effect, if you
   select PublicScreen for Screentype. If you leave this gadget empty,
   Fiasco will use the default public screen.

Screen/Screen Mode: You may select the display mode here for the custom
   screen. Clicking on the popup gadget will open an ASL screenmode
   requester. This requires asl.library version 38 or higher.

Screen/Screen Font: This gadget controls whether you want to use a custom
   font for the custom screen or the Workbench screen font which is
   controlled by the Font Preferences.

Screen/Custom Font: If you want to use a custom font for the custom
   screen, you may select it here.

Mask Font/Mask Font: This gadget controls whether you want to use a
   custom font for the mask or the system default font which is
   controlled by the Font Preferences.

Mask Font/Custom Font: You may select a custom font for the mask here. It
   must be fixed width.

Images/New look proportional gadgets: Activate this option to get
   proportional gadets in the mask in "new look".  Proportional gadgets
   are used for slider fields and for the scrollbars of datatypes, var
   string and listview fields. Instead of the black bars, new look
   proportional gadget will be white bars with a black border at the
   right and at the bottom, which makes them to appear raised.

Ok: Proceeds with resetting the display of Fiasco. If required, a new
   screen is opened and so on.

Cancel: Closes the requester without any further action.
```

1.210 External Programs and Paths Requester

External Programs and Paths Requester

This requester can be used to edit the settings related to external programs, that Fiasco may call and the settings related to extenal paths.

Programs/Editor Program: Specify your favourite editor program here. You may specify a %s here, which will be replaced with the file name to be edited. The program should start synchronously. With programs like GED, that normally start asynchronously, you should specify the respective arguments in order to stop the program to start that way. For GED this would be the argument sticky.

Programs/Editor Program/Stack: Specify the stack for the editor program here. It should be normally greater than or equal 4000 bytes.

Programs/Text Viewer Program: Takes your text viewer program. This may be More, Most or Multiview.

Programs/Text Viewer Program/Stack: Specify the stack for the text viewer program here. It should be normally greater than or equal 4000 bytes.

Paths/Fiasco Path for Icons: The path specified here will be saved as default tool in Fiasco database icons. Thus, clicking on a Fiasco database icon will start the program specified here. Example: Work:Fiasco/Fiasco

Paths/Default Database Directory: Specify here the default path for databases. The file requesters will search for databases relative to this directory.

Ok: Closes the requester and accepts the changes made in it.

Cancel: Closes the requester and ignores the changes.

1.211 Print Options Requester

Print Options Requester

The print options requester can be opened using Project/Options in the print window. You may control some options for printing here. See the Print section for more information on printing. The settings which have been made here may be saved using the menuitems Project/Save and Project/Save as of the print window.

Print to: Fiasco's print function writes its data to this file. If you want to use conventional printing, you should specify PRT: for the printer here.

Print with ARexx: Activate this gadget if you want Fiasco to call the ARexx script with the name ProgDir:ARexx/ARexxPrint.rexx after writing the file. Fiasco will call the script with the file name specified in

Print to as its argument. In a standard Fiasco installation, this script calls TeX to compile the file into a DVI file and prints this. However, you may change the script to something completely different. If you use Print with ARexx, you must not specify PRT: in Print to. A temporary file, e.g. T:FiascoPrint, would be the best.

Only marked records: If you activate this gadget the print function will print only records with a mark.

1.212 Print Element Requester

Print Element Requester

You can control several options of a print element in the print window with this requester. It appears when you add with Element/Add or edit with Element/Edit an element. The layout of a requester depends on the in Element/Type selected element type. See the print section on more information about elements.

Field Elements:

Field: Select here the field, whose data will be inserted at the position of the print element.

Width: The width of the print element in the print mask.

Height: The height of the print element in the print mask. Only available for var string fields or listview fields.

Clip: If this checkbox is active, the data of the field will be clipped if they would be larger than the size of the print element. If it is not active, the size of the print element will be extended to show all data. If neccessary, other elements will be shifted to the right or -- if the element needs more vertical space -- down.

Style/Bold: If selected, the field data will be printed in bold face.

Style/Italics: If selected, the field data will be printed in italics.

Style/Underlined: If selected, the field data will be printed underlined. Note: The Bold, Italics and Underlined effect is generated by escape sequences. It is not compatible with printing by TeX.

Style/Justification: You may select here, whether the contents of the element will be justified to the left or right border or will be centered in the element. Default: Left. If Clip is not active, Fiasco may fall back to left justification when the field contents exceed the element borders.

```
Text Elements:



Text: The text for the element. The width of the element is adjusted
   to the length of the text.

Style/Bold: If selected, the text will be printed in bold face.

Style/Italics: If selected, the text will be printed in italics.

Style/Underlined: If selected, the text will be printed underlined. Note:
   The Bold, Italics and Underlined effect is generated by escape
   sequences. It is not compatible with printing by TeX.



Response Gadgets:



Ok: Acceppts the settings. If certain settings are not possible or
   need to be adjusted, you will be notified about that.

Cancel: Ignores all changes.

There is no requester for formfeed elements.
```

1.213 Formulas

```
                Formulas

Starting with release 2.1, Fiasco supports formulas. Formulas can be used
to calculate the content of fields automatically or to do complex
searches. Formulas for fields may be specified in the
                field requester
                ,
formulas for searches in the
                search requester
                . From both requesters you
have access to the
                edit formula requester
                . This requester can be used as
tool for creating formulas. A formula may look like this:

    floor(price * vat / 100) + price

    The parts a formula is built of are described in the following
sections.
```

1.214 Constant Values

Constant Values

Constant values are defined directly in the formula. In the formula above the 100 is a definition of a constant numeric value. You can also define strings as constant values by specifying the string in double quotes. For example: "Test String".

In Fiasco formulas, strings and numbers are exchangable. Whenever a number is required and a string is available, the string will be converted to a number. The reverse is also true when a string is required and only a number is available.

Rules for constant values also apply to the following elements of formulas that "return" some values. Figuratively speaking, an element of a formula that has returned a value during calculation, is replaced by the returned value.

The final result of a formula is, from that point of view, the value that has finally been returned.

Boolean Values

The boolean values "true" and "false" as used in logical operations are represented in Fiasco formulas by numbers. Numbers not equal zero represent a true value, zero represents "false".

1.215 Fields

Fields

Fields in this context are references to fields in the database and are replaced during the calculation by the content of the field in the current record. If the content of a field in the formula of another field is changed, the formula will be recalculated automatically. Fields appear in formulas like variables in normal, mathematical formulas. A field in the example above is price. The contents of fields of the types

· String

· Date

· Time

· Extern

· Datatypes

· Var String

are returned as strings, the contents of the remaining field types are returned as numbers. Cycle fields return the number of the selected label and boolean fields return a 1 for a checked field and a 0 for an unchecked field. Fields have to be specified using their IDs in the formulas. To be recognized as fields, the field IDs may not begin with numbers and may not contain spaces.

The elements of listview fields can be addressed with square brackets after the field ID containing the number of the element to be used. The number of the entry has the base 0, thus the first entry has the index number 0, the second 1, etc. namelist[5] for example, represents the sixth entry in the listview field with the ID namelist.

1.216 Constants

Constants

Constants have the same appearance as fields in Fiasco formulas. The difference is that they are globally defined values that are equal for the whole database. The
constants requester
can be used to define new
constants. vat could be a constant in the example.

1.217 Operators

Operators

Operators are the symbols that define the most important mathematical or logical operations. An operator normally requires two operands that have to be placed at both sides of the operator. Some operators require only one or three operands. Operators in the example above are *, / and +. Furthermore, operators in Fiasco formulas have evaluation priorities. Operations with operators of higher priority are calculated before operations with operators of lower priority. The order rule stating that multiplications or divisions be performed before additions is an example of operator priority.

See the table for the operators currently supported by Fiasco.

Pri	Op.	Description	Syntax	Result
10	!	Logical not	! boolvalue	boolean value
9	^	Power	number ^ exponent	number
8	*	Multiplication	4 * 4	number
	/	Division	10 / 2	number
7	+	Addition	2 + 3	number
	-	Subtraction	5 - 1	number
6	<	Less than	value1 < value2	boolean value
	>	Greater than	value1 > value2	boolean value
	<=	Less than or equal	value1 <= value2	boolean value
	>=	Greater than or equal	value1 >= value2	boolean value
5	==	Equal	value1 == value2	boolean value
	!=	Not equal	value1 != value2	boolean value
4	&&	Logical and	bool && bool	boolean value
	||	Logical or	bool || bool	boolean value
3	? :	cond. (see below)	bool ? expr. : expr.	result of one expr.
2	numentries()	number of entries in lv	numentries(fieldid)	integer number
	active()	active entry of listv.	active(fieldid)	integer number
	sum()	sum of listv. (see below)	sum(fieldid[,s][,e])	number
	current()	current entry (see below)	current()	integer number

Conditional Operator

The conditional operator is a short version of the if-else construct in normal programming languages. If you are familiar with the programming

language ‘C’ you should already know that operator.

The operand before the question mark should result in a boolean value. If the value is true (i. e. non-zero) the operator will be replaced by the result of the expression after the ? and before the :. If the boolean value is false (i. e. equal to zero) the expression after the : will be used.

A formula that returns the larger one of two numbers would look like that:

```
a > b ? a : b
```

Of course, you can use several conditional operators in a formula. A formula that determines the sign of a number would look like this:

```
n > 0 ? "Positive" : n < 0 ? "Negative" : "Zero"
```

If n is greater than 0, the string Positive will be returned by the formula. If this is not true, n will be tested, whether it is less than 0. If this is true, the string Negative will be returned. If this is not true the number must be Zero.

Listview Sum Operator

The operator sum() can be used to calculate the sum of entries in a listview field. Regardless of the type of the field, Fiasco will try to sum up the data as numbers. The simple syntax of sum() requires only the field ID of the field to be summed, for example sum(prices). You may, however, also specify the entry at which summing should be started in a second argument and the entry at which the calculation should be ended in a third argument. The numbers of the entries must be specified -- like accessing single entries of listviews with square brackets -- on base zero, thus the first entry is entry 0, and so on. For example, sum(prices, 2, 4) adds the third, fourth and fith entries of the listview field with the ID prices.

Current Entry Operator

The operator current() can be only used within formulas, which are used to calculate the values of a listview field. It returns the number of the entry, which is currently calculated by this formula. This number is on base zero. Thus, if the first entry of a listview is currently calculated, current() will return 0.

This way, you are able to calculate a value across grouped listview fields. An example is an invoice in which the single item price and the number of items is specified in grouped listview fields and the price of the ordered number of items is to be calculated:

```
singleprice[current()] * itemnumber[current()]
```

1.218 Functions

Functions

Functions are defined by the function name and the arguments of the function enclosed in parenteses. If a function has no arguments an empty pair of parenteses has to be specified, nethertheless. A function may take some arguments, do some calculations on them and return a new value. This value will be used for the further calculations. In the example above, floor() is a function. It takes only one argument. The argument is the result of the calculation price * vat / 100. For a list of all built-in functions of Fiasco, see the function reference. Beneath the built-in functions of Fiasco, the user may also define own functions, based on other formulas. Thus, you may define a function with the name

```
min(a, b)
```

that does the following:

```
a < b ? a : b
```

This function determines what number is the smallest and returns it. If you have defined this function in the function requester you can use it in all formulas of the particular database.

As calls to user-defined functions are internally resolved before calls to built-in functions, you may replace a built-in function by a user-defined function with the same name.

Essentially, you can change the name of a built-in function by simply redirecting the call with a user-defined function. If you defined a function with the name

```
length(a)
```

to do

```
strlen(a)
```

you can use a function named length() that does the same as the function strlen().

Note:
recursive programming. Better not try it!

1.219 Function Reference

Function Reference

Fiasco currently has these built-in functions:

```
abs()

activerecord()

asin()

acos()

atan()

ceil()

cos()

currentdate()

currenttime()

datediff()

day()

floor()

formatdate()

formattime()

hour()

left()

lg()

ln()

minute()

month()

numrecords()

printf()

rand()

right()

round()

second()

sign()

sin()
```

sqrt()

strcat()

strcmp()

stricmp()

strlen()

strmid()

strrev()

strstr()

tan()

tolower()

toupper()

version()

year()

1.220 abs()

abs()

Name: abs() - Absolute value

Synopsis: abs(value)

Function: Removes the sign of the argument.

Arguments: value - number

Result: a positive number

1.221 activerecord()

activerecord()

Name: activerecord() - Get the number of the active record (8)

Synopsis: activerecord()

```
Function: Returns the number of the active record. Note that the active
   record is not necessarily the record of the field whose value is just
   calculated by this formula.

Arguments: none

Result: a positive number
```

1.222 asin()

```
asin()


Name: asin() - Arcsine function

Synopsis: asin(value)

Function: Calculates the arcsine of a value and returns an angle in
   radians. (2 pi represents a circle)

Arguments: value - must be between -1 and 1

Result: An angle between -pi/2 and pi/2
```

1.223 acos()

```
acos()


Name: acos() - Arccosine function

Synopsis: acos(value)

Function: Calculates the arccosine of a value and returns an angle in
   radians. (2 pi represents a cirle)

Arguments: value - must be between -1 and 1

Result: An angle between -pi/2 and pi/2
```

1.224 atan()

```
atan()


Name: atan() - Arctangent function
```

```
Synopsis: atan(value)

Function: Calculates the arctangent of a value and returns an angle in
   radians. (2 pi represents a cirle)

Arguments: value

Result: An angle between -pi and pi
```

1.225 ceil()

```
ceil()


Name: ceil() - rounding

Synopsis: ceil(value)

Function: Returns the smallest whole number not less than the specified
   number.

Arguments: value - real number

Result: a whole number
```

1.226 cos()

```
cos()


Name: cos() - cosine function

Synopsis: cos(angle)

Function: Calculates the cosine of an angle in radians. (2 pi represents
   a cirle)

Arguments: angle - Angle in radians.

Result: The cosine of the angle. Between -1 and 1
```

1.227 currentdate()

```
currentdate()
```

```
Name: currentdate() - get the current date

Synopsis: currendate()

Function: Returns the current date as a string in the format of the
   locale prefs.

Arguments: None

Result: A date string
```

1.228 currenttime()

```
currenttime()


Name: currenttime() - get the current time

Synopsis: currentime()

Function: Returns the current time as a string in the format of the
   locale prefs.

Arguments: None

Result: A time string
```

1.229 datediff()

```
datediff()


Name: datediff() - Get the time between two dates

Synopsis: datediff(date1, date2)

Function: Calculates the time span in days between the two given dates.
   If one date has an unspecified element (shown as ??) the element of
   the other date will be used. Always the time span from date1 to date2
   will be calculated. If date2 is before date1, a negative result will
   be returned.

Arguments: date1, date2 - two date strings

Result: Number of days between the two dates.
```

1.230 day()

```
day()


Name: day() - Get the day of a date

Synopsis: day(datestring)

Function: Parses a date string in the format of the locale prefs and
   returns the day of the date.

Arguments: datestring - A date string

Result: The day of the date string.
```

1.231 floor()

```
floor()


Name: floor() - rounding

Synopsis: floor(value)

Function: Returns the largest whole number not greater than the specified
   number.

Arguments: value - real number

Result: a whole number
```

1.232 formatdate()

```
formatdate()


Name: formatdate() - create a date string

Synopsis: formatdate(year, month, day)

Function: Formats the specified date to a date string that uses the
   format specified in the locale prefs. All other date functions of
   Fiasco use this format.

Arguments: year, month, day - the elements of the date

Result: a formatted date string.
```

1.233 formattime()

```
formattime()


Name: formattime() - create a time string

Synopsis: formattime(hour, minute, second)

Function: Formats the specified time to a time string that uses the
   format specified in the locale prefs. All other time functions of
   Fiasco use this format.

Arguments: hour, minute, second - the elements of the time

Result: a formatted time string.
```

1.234 hour()

```
hour()


Name: hour() - Get the hour of a given time

Synopsis: hour(timestring)

Function: Parses a time string in the format of the locale prefs and
   returns the hour of it.

Arguments: timestring - a time string

Results: The hour in 24 hour format
```

1.235 left()

```
left()


Name: left() - Get the left part of a string

Synopsis: left(string, length)

Function: Returns the left part of a string with the specified length. If
   length is greater than the string itself, the whole string will be
   returned.

Arguments: string - string to be scanned
   length - length of the part to be returned

Result: A substring of string
```

1.236 lg()

```
lg()


Name: lg() - base 10 logarithm

Synopsis: lg(value)

Function: Returns the logarithm on the base 10.

Arguments: value - a posititive real number

Result: a real number
```

1.237 ln()

```
ln()


Name: ln() - natural logarithm

Synopsis: ln(value)

Function: Returns the logarithm on the base e.

Arguments: value - a posititive real number

Result: a real number
```

1.238 minute()

```
minute()


Name: minute() - Get the minute of a time

Synopsis: minute(timestring)

Function: Parses a time string in the format of the locale prefs and
   returns the minute of it.

Arguments: timestring - a time string

Results: a whole number
```

1.239 month()

```
month()


Name: month() - Get the month of a date

Synopsis: month(datestring)

Function: Parses a date string in the format of the locale prefs and
   returns the month of the date.

Arguments: datestring - A date string

Result: The month of the date string.
```

1.240 numrecords()

```
numrecords()


Name: numrecords() - Get the number of records (8)

Synopsis: numrecords()

Function: Returns the number of records in the active index in the
   database.

Arguments: None.

Result: The number of records. >= 0
```

1.241 printf()

```
printf()


Name: printf() - Create a formatted string

Synopsis: printf(formatstring, ...)

Function: Formats the data specified as arguments according to the
   formatting rules specified in formatstring. Formatstring is a string,
   that may contain control sequences that will be replaced by formatted
   data that have been specified as arguments. All control sequences are
   introduced by a percent sign. A control sequence has the format
   (fields in brackets are optional):
   %[flags][width][.precision]type
   The arguments are associated with the
   control sequences using the order.
```

```
Flags: can be one of the following:
   - the result will be left-justified within the width
   + If specified, a plus sign will be added before a positive number
   0 The with of the field will be padded with zeros if it is a
   number

Width: The width is the minimum number of characters that this formatting
   item should use.

Precision: A whole number. The precision of the output, depending of the
   type of the control sequence:
   d,u,o,x,X - minimum number of digits
   e,f - number of digits after of decimal point
   g - maximum number of significant digits
   s - maximum length of the string to be copied


Type: can be one of the following:
   c - One character of the specified ASCII value will be generated.
   d - A signed integer value will be generated.
   u - An unsigned integer value will be generated.
   o - An octal number will be generated.
   x - A hexadecimal number will be generated. The characters a-f will be  ↩
      lowercase.
   X - A hexadecimal number will be generated. The characters
   a-f will be uppercase.
   e - A fractional number with the format d.dde-ddd will be generated.
   f - A fractional number with the format dd.dd will be generated.
   g - A fractional number will be generated. The format will
   be chosen according to the magnitude of the number.
   s - A string will be inserted.

Arguments: The formatstring and the further arguments for the
   formatstring.

Result: A formatted string
```

1.242 rand()

```
rand()


Name: rand() - random number generator

Synopsis: rand()

Functions: Returns a random number.

Arguments: none

Result: A fractional random number. 0.0 <= r < 1.0.
```

1.243 right()

```
right()


Name: right() - Get the right part of a string

Synopsis: right(string, length)

Function: Returns the right part of a string with the specified length.
   If length is greater than the real length of the string, the whole
   string will be returned.

Arguments: string - string to be scanned
   length - length of the part to be returned

Result: A substring of string
```

1.244 round()

```
round()


Name: round() - rounding

Synopsis: round(value)

Function: Returns a mathematically rounded number of value.

Arguments: value - real number

Result: a whole number
```

1.245 second()

```
second()


Name: second() - Get the second of a time

Synopsis: second(timestring)

Function: Parses a time string in the format of the locale prefs and
   returns the second of it.

Arguments: timestring - a time string

Results: a whole number
```

1.246 sign()

```
sign()


Name: sign() - sign of a number

Synopsis: sign(value)

Function: Checks the sign of a number

Arguments: value - a number

Result: -1 if value is negative, 1 if positive, 0 if zero
```

1.247 sin()

```
sin()


Name: sin() - sine function

Synopsis: sin(angle)

Function: Calculates the sine of an angle in radians. (2 pi represents a
   cirle)

Arguments: angle - Angle in radians.

Result: The sine of the angle. Between -1 and 1
```

1.248 sqrt()

```
sqrt()


Name: sqrt() - Square root

Synopsis: sqrt(value)

Function: Returns the square root of the specified number.

Arguments: value - Must be >= 0.

Result: a positive number
```

1.249 strcat()

```
strcat()


Name: strcat() - Concatenate strings

Synopsis: strcat(string1, string2, ...)

Function: Concatenates two or more strings to one string, that is
   returned.

Arguments: string1 ... stringn - The strings to be concatenated. At least
   two, no maximum.

Result: The concatenated string.
```

1.250 strcmp()

```
                strcmp()


Name: strcmp() - Compare two strings

Synopsis: strcmp(string1, string2)

Function: Checks whether the specified strings are equal. The comparision
   is done case sensitive, i.e. also the case of the strings has to be
   equal when the two strings should be considered equal. A case
   insensitive comparision can be done with
                stricmp
                .

Arguments: string1, string2{} - strings to be compared

Result: 0 if both strings are equal, -1 if string1 < string2 and +1 if
   string1 > string2
```

1.251 stricmp()

```
                stricmp()


Name: stricmp() - Compare two strings

Synopsis: stricmp(string1, string2)

Function: Checks, whether the specified strings are equal or not. The
   comparision is done case insensitive, i.e. the case of the strings
```

```
   does not matter during comparision. A case sensitive comparision can
   be done with
                strcmp
                .

Arguments: string1, string2{} - strings to be compared

Result: 0 if both strings are equal, -1 if string1 < string2 and +1 if
   string1 > string2
```

1.252 strlen()

```
strlen()


Name: strlen() - Length of a string.

Synopsis: streln(string)

Function: Returns the length in characters of the specified string.

Arguments: string - String to be measured

Result: Length of the string in characters
```

1.253 strmid()

```
strmid()


Name: strmid() - Copy a part of a string

Synopsis: strmid(string, start, length)

Function: Returns the specified section of the specified string.

Arguments: string - String to be scanned
   start - Start of the substring, counting from zero.
   length - Length of the string to be returned.

Result: Substring of the specified string
```

1.254 strrev()

```
strrev()


Name: strrev() - Create a reverse string

Synopsis: strrev(string)

Function: Returns a string those characters have the reverse order of the
   argument. Calling strrev(strrev(string)) has effectively no effect.

Arguments: string - string to be reversed

Result: reversed string
```

1.255 strstr()

```
strstr()


Name: strstr() - Search for a string in a string

Synopsis: strstr(string, substring)

Function: Searches for substring in string and returns the position at
   which it was found.

Arguments: string - String to be searched
   substring - String for which will be searched

Result: Position at which substring was found. Counting from zero, thus 0
   indicates the first character. Returns -1 if the string was not found.
```

1.256 tan()

```
tan()


Name: tan() - Tangent function

Synopsis: tan(angle)

Function: Calculates the tangent of an angle in radians. (2 pi represents
   a cirle)

Arguments: angle - Angle in radians.

Result: The tangent of the angle.
```

1.257 tolower()

```
tolower()
```

```
Name: tolower() - Convert a string to lowercase

Synopsis: tolower(string)

Function: Converts all characters in the string to lowercase.

Arguments: string to be converted

Result: string with only lowercase characters
```

1.258 toupper()

```
toupper()
```

```
Name: toupper() - Convert a string to uppercase

Synopsis: toupper(string)

Function: Converts all characters in the string to uppercase.

Arguments: string to be converted

Result: string with only uppercase characters
```

1.259 version()

```
version()
```

```
Name: version() -Get the Fiasco version

Synopsis: version()

Function: Returns the internal version of Fiasco. For Fiasco 2.2, this is
   8.

Arguments: none

Result: Fiasco's internal version as whole number
```

1.260 year()

```
year()


Name: year() - Get the year of a date

Synopsis: year(datestring)

Function: Parses a date string in the format of the locale prefs and
   returns the year of the date.

Arguments: datestring - A date string

Result: The year of the date string.
```

1.261 The ARexx Port

```
                The ARexx Port

ARexx is a macro programming language capable of connecting different
programs. ARexx has been developed by William S. Hawes and has been part
of the system software since OS 2.0.

    The ARexx port of Fiasco may be accessed externally from a script or
ARexx scripts can be called by Fiasco. For example: you can specify an
ARexx script in the command attribute of a button and then click on the
button -- this script is then executed and may do certain operations.

    For Fiasco 2.1, Fiasco's ARexx port has been completely revised to be
more powerful and style guide compliant. ARexx scripts written for Fiasco
1.x or 2.0x, will continue to work, though. "Old" scripts may be
recognized with the command Address FIASCO at the beginning of the script
and the commands with the prefix F_ before the Fiasco commands. The old
commands are no longer documented. See the 2.0x versions of this document
for documentation of these commands. Please also note, that you cannot
mix old and new scripts. Fiasco ARexx scripts must use either completly
the new commands or completely the old commands.

    Scripts that conform to the new standard do not address the port with
the name FIASCO. The new ARexx port assigns for each database an own
ARexx port. The names of the ports are FIASCO.n, where n is a number. If
the script is directly started by Fiasco (e.g. by clicking on a button)
the script will be addressed automatically to the ARexx port of the
corresponding database. Thus, no additional Address command is required.
More about accessing Fiasco's ARexx port   in the section

                Accessing the Port
                .

    Nearly all operations that can be used with Fiasco's GUI can be used
with the ARexx commands. Additionally, Fiasco's functions may be extended
with ARexx. There are many ARexx commands which perform exactly the same
```

as their GUI "brothers". In other words, certain commands may open a requester under certain conditions. There are also commands which always open a requester.

1.262 Style Conventions

Style Conventions

This section documents some stylistic conventions for Fiasco ARexx scripts. Many are not required, even though they are recommended.

ARexx allows to chose an own suffix for application specific scripts. For Fiasco 2.1 ARexx scripts, the suffix .frx should be used instead of .rexx.

Fiasco ARexx scripts should always lock Fiasco's GUI with the command

LockGUI
 to avoid any influences by the user. Thus, the ARexx script should have extensive error checking to avoid the case that the script was stopped because of an error and the user is locked out of Fiasco's GUI. The script dummy.frx of the Fiasco distribution contains already all neccessary checking and can be used as basis for new ARexx scripts.

1.263 Accessing the Port

Accessing the Port

As of Fiasco 2.1, ARexx scripts started by Fiasco are already addressed to the ARexx port of the Fiasco database from that the script has been started. Thus, you do not need to worry about the name of the ARexx port you have to access. If you want to address an ARexx script to the ARexx port of another program, and later return to Fiasco's ARexx port, two techniques are available:

If you address the script to another ARexx port, you may later use the command Address without any arguments to return to the previously active ARexx port, which was, if the script was started by Fiasco, the port of a Fiasco database. For example:

```
/* A script that was started by Fiasco */

Address MULTIVIEW.1

/* Do something with MultiView */

Address

/* You are now addressed to Fiasco again */
```

Another way is to save the name of the active ARexx port in a variable and to later use Address with that variable to return to the original port. You have to use Address Value Variable to make ARexx recognize that you have specified a variable name and not a literal port name. The advantage is that you may address several ports before going back to the original port. The other method forgets about the Fiasco port after the second Address Port command. For example:

```
/* A script that was started by Fiasco */

fiasco_port = address()

Address MULTIVIEW.1

/* Do something with MultiView */

Address Value fiasco_port

/* You are now addressing Fiasco again */
```

The script dummy.frx of the Fiasco distribution already initializes a variable with the name fiasco_port to the correct ARexx port.

Getting the Port of the active Project

If you want to start a Fiasco 2.1 ARexx script from outside Fiasco, e.g., by double clicking on a project icon with rx as default tool, the script has to search for the port of the active Fiasco database.

This can be done this way:

```
/* Get a list of all available ports */

ports = show("Ports")

/* Search for a port of Fiasco */

do i = 1 to words(ports)
```

```
    if abbrev(word(ports, i), "FIASCO.") then
    do
        /* A port of Fiasco has been found.
         * Now query Fiasco to return the port
         * name of the active database.
         */

        Address Value word(ports, i)

        GetAttr Project Name Active ARexx

        Address Value Result

        break
    end
end
```

This procedure is already part of the script dummy.frx.

1.264 Arguments of Commands

```
                Arguments of Commands
```

Fiasco interprets the data after the command as arguments or parameters. Fiasco uses ReadArgs() of the dos.library to parse the parameters. Thus, the same argument conventions used by CLI commands apply to Fiasco. Arguments are separated by white spaces. If single arguments are supposed to contain spaces, simply enclosing them in quotation marks does not work. This is because ARexx swallows all quotation marks. To avoid this you should enclose quotation marks within additional, single quotation, e.g. Open ' "Test File" '. You have to use the single quotation marks in the outer position because Fiasco (ReadArgs()) can only handle double quotation marks. Be sure not to use variables inside of any quotations. To use them you have to close the outer quotation, write the variable and open the quotation again, if required. These issues do not apply for arguments which have the /F modifier.

```
                Results of Commands
```

1.265 Results of Commands

```
                Results of Commands
```

If a command returns a value, this is normally stored in the variable Result. To use Result, you have to put the line Options Results at the beginning of a script. The script dummy.frx already does that. A value returned in Result may be also redirected to another variable. To do that, you have to specify the argument Var followed by the name of the variable with the arguments of the command. Note: If you call commands with the same Var argument in sequence and you do not specify the

argument in quotes, ARexx will replace at the second command the variable name by the result of the first command. Commands that return more complex results return them as stem variables. If you use such commands, you have to specify a name for the variable to be set. If the command returns a list of a variable number of items, the values of the items will be returned in stem.1 . . . stem.n. stem.count will contain the number of items returned.

Fiasco only returns a result in the case of success. Success is indicated by the variable rc, which is always set after the execution of a command. 0 means success, 5 is -- by definition -- a warning, 10 an error and 20 a fatal failure.

Starting with Fiasco 2.0, Fiasco supports a special ARexx variable named FIASCO.LASTERROR, which will be set on failure of a Fiasco ARexx command.

The error codes that may be returned by Fiasco are documented in the appendix

Error Codes

.

The ARexx command

Fault

can be used to convert an error code into human-readable text.

Debugging ARexx Scripts

1.266 Debugging ARexx Scripts

Debugging ARexx Scripts

Fiasco has additional features to make debugging of ARexx scripts as easy as possible. Whenever a Fiasco command fails to work, Fiasco will return in the variable rc a number not equal zero. Furthermore, the special variable Fiasco.LastError will be set to a Fiasco error code. If you use error signals, the ARexx script may catch the error and display a comfortable requester with the error using Fiasco's ARexx port. An example for this is the script dummy.frx in the Fiasco distribution.

One step further goes the ARexx-Debug function of Fiasco. You can activate that function by selecting the menu item

Control/ARexx Debug

.

Whenever now a Fiasco ARexx command fails, Fiasco will suspend the execution of the ARexx script and display a requester that explains the error, shows the correct command template and gives you the possibility to open the AmigaGuide help for the ARexx command. This is done if you click on Help. The other buttons Continue and Ignore Error continue the script, while Continue sets rc to the error code and Ignore Error sets rc to zero, making the script assume, that the command has worked successfully.

1.267 ARexx commands by alphabet

ARexx commands by alphabet

Command index by function

ActivateDBWindow

ActivateField

ActiveIndex

ActiveRecord

AddLVFieldEntry

AddRecord

CalculateFormula

Clear

CloneRecord

Close

CloseListWindow

CloseServiceWindow

ConvertField

CopyRecord

CountRecords

CreateField

CutRecord

DeleteAllRecords

DeleteConstant

DeleteLVFieldEntry

DeleteRecord

Export

Fault

```
Filter

Find

FlushRecords

GetAttr

GetConstant

GetField

GetRecordMark

HideProject

Import

LoadDTFieldObject

LockGUI

MarkMatch

MarkRecord

MenuControl

MoveRecord

New

NewSearchInfo

Open

OpenListWindow

OpenServiceWindow

PasteRecord

Progress

Quit

ReadSettings

RecompileFormulas

RequestChoice

RequestField

RequestFile

RequestNumber
```

RequestString

ResetStatus

RevealProject

Save

SaveAs

SaveSettings

SetAttr

SetConstant

SetField

SetMode

SetSearchField

SetStatus

Sort

UnlockGUI

1.268 ARexx commands by function

ARexx commands by function

Command index by alphabet
Records

ActiveRecord

AddRecord

CloneRecord

CopyRecord

CountRecords

CutRecord

DeleteAllRecords

DeleteRecord

PasteRecord
Record data

AddLVFieldEntry

DeleteLVFieldEntry

SetField

GetField

GetRecordMark

MarkRecord

MoveRecord
Index

ActiveIndex
Mask

ConvertField

CreateField

GetAttr

SetAttr
Searching / Sorting

Filter

Find

MarkMatch

NewSearchInfo

SetSearchField

Sort
Databases / Projects

Clear

Close

GetAttr

New

Open

```
Quit

Save

SaveAs

SetAttr

SetMode
GUI


ActivateDBWindow

ActivateField

LockGUI

UnlockGUI

OpenListWindow

OpenServiceWindow

CloseListWindow

CloseServiceWindow

HideProject

LoadDTFieldObject

MenuControl

Progress

RequestChoice

RequestField

RequestFile

RequestNumber

RequestString

ResetStatus

RevealProject

SetStatus
Data Exchange


Export
```

Import
Formula Support

CalculateFormula

RecompileFormulas

SetConstant

GetConstant

DeleteConstant
Settings

ReadSettings

SaveSettings
Miscellaneous

Fault

FlushRecords

1.269 ActivateDBWindow

ActivateDBWindow

Name: ActivateDBWindow -- Activate a database's window (8)

Synopsis: ActivateDBWindow Project/K
   RC = Success

Function: Activates the mask window of the specified database. The
   user's input focus will change to this window. This has no effect on
   the working of the calling ARexx script. If the database is hidden,
   this command does nothing.

Arguments: Project - Name of database to be activated.

Results: RC - Success

1.270 ActivateField

ActivateField

```
Name: ActivateField - Activate a field in the mask.

Synopsis: ActivateField FieldID/A
   RC = Success

Function: Activates the field with the specified ID in the mask. Only
   fields, which appear as a string or longint gadget may be activated.
   If project or window is not active, the field cannot be activated.
   This command may be only called in record mode.

Arguments: FieldID - ID of the field to be activated.

Results: RC - Zero if the field has been activated.
```

1.271 ActiveIndex

```
               ActiveIndex


Name: ActiveIndex -- Control the active index

Synopsis: ActiveIndex Index/K,Next/S,Prev/S,NoHistory/S,Force/S,Var/K
   RC = Success
   Result = Active Index

Function: This command controls the active index. You may specify an
   index by name or activate the next or previous index in the

               index history
               . The name of the newly activated index will be returned.
   If you do not specify any of Index, Next or Prev, only the name of the
   active index will be returned.

Arguments: Index - Activate specified index.
   Next - Activate next index in history.
   Prev - Activate previous index in history.
   NoHistory - Do not track this change in the index history.
   Force - Suppress all warnings.

Results: RC - Success. Result - Name of the now active index.
```

1.272 ActiveRecord

```
ActiveRecord


Name: ActiveRecord -- Control the active record

Synopsis: ActiveRecord Record/K/N,First/S,Last/S,Next/S,Prev/S,Var/K
```

```
   RC = Success
   Result = New active record

Function: With ActiveRecord you may control the active record in the
   addressed database. You may activate a record specified by record
   number, the first, the last or the next or the previous record
   relative to the currently active record. The number of the new active
   record will be returned. If you do not specify any of the five first
   arguments, only the number of the active record will be returned and
   the position won't be changed.  This command may be only called in
   record mode.

Arguments: Record - Activate specified record.
   First - Activate first record.
   Last - Activate last record.
   Next - Activate the record after the currently active one.
   Prev - Activate the record before the currently active one.


Results: RC - Success
   Result - Number of now active record.
```

1.273 AddLVFieldEntry

```
                AddLVFieldEntry


Name: AddLVFieldEntry - Add an entry to a listview field

Synopsis: AddLVFieldEntry FieldID/A,Record/K/N,ListEntry/K/N
   RC = Success

Function: Adds a new entry after the specified entries of the specified
   listview field. If no entry is specified, the entry will be added as
   the last entry. It will contain the Initial Content set in the field
   requester. To change the content, use
                SetField
                .  This command may be
   only called in record mode.

Arguments: FieldID - ID of listview field
   Record - Number of record to be changed
   ListEntry - Number of entry after which the entry will be added.
   Counting from 1. 0 for adding an entry at the top of the list.

Results: RC - Success
```

1.274 AddRecord

```
AddRecord


Name: AddRecord - Add a new record to a database

Synopsis: AddRecord Record/K/N,Inactive/S,Var/K
   RC = Success
   Result = Record Position

Function: Creates a new record and adds it to the addressed database. The
   record will contain the initial values.  This command may be only
   called in record mode.

Arguments: Record - Record after which the new record will be inserted
   into the database. Please note that Fiasco may choose another position
   for the record. This may be for instance with automatic sorting the
   case. If not specified, the record will be inserted after the active
   one.
   Inactive - Record will not be activated.


Results: RC - Success
   Result - The real position of the new record.
```

1.275 CalculateFormula

```
CalculateFormula


Name: CalculateFormula - Calculates a formula.

Synopsis: CalculateFormula Formula/A,Record/K/N,Var/K
   RC = Success
   Result = Result of calculation

Function: Calculates a formula and returns the result. If fields are
   referenced in the formula, the field values of the current or the
   specified record in the addressed database will be used.  This command
   may be only called in record mode.

Arguments: Formula - Formula to be calculated.
   Record - Record to be used as current one for calculation.

Results: RC - Success.
   Result - Result of calculation.
```

1.276 Clear

```
               Clear
```

```
Name: Clear -- Clear a project

Synopsis: Clear Force/S
   RC = Success

Function: Deletes all data in the addressed project. It will be in a
   state much like after a New.  If you do not specify Force, this
   command does exactly the same as
               Project/Erase
               . That means, it is
   possible, that a requester opens, which asks you whether you want to
   save the current project before proceeding or cancel.  To prevent
   this, specify the Force parameter. This will suppress all warnings.

Arguments: Force - Suppress all warnings.

Results: RC - Not equal zero, if user cancelled warning requester.
```

1.277 CloneRecord

```
CloneRecord


Name: CloneRecord -- copy a record

Synopsis: CloneRecord Record/K/N,To/K/N
   RC = Success

Function: Creates an exact copy of the active or specified record and
   inserts it at the specified position in the active index. May also
   affect other indices.  This command may be only called in record mode.

Arguments: Record - Number of record to be cloned.
   To - Number of record after which the new record will be
   inserted.

Results: RC - Success
```

1.278 Close

```
               Close


Name: Close -- Close a database project

Synopsis: Close Force/S
   RC = Success
```

```
Function: Closes the addressed project. If you do not specify Force, this
   command does exactly the same as
               Project/Quit
               . That means, it is
   possible, that a requester opens, which asks you whether you want to
   save the current project before proceeding or cancel.  To prevent
   this, specify the Force parameter. This will suppress all warnings.

Arguments: Force - Suppress all warnings.

Results: RC - Not equal zero, if user cancelled the warning requester.
```

1.279 CloseListWindow

```
CloseListWindow


Name: CloseListWindow -- Close the list window

Synopsis: CloseListWindow

Function: Closes the list window of the currently addressed project. If
   it is already closed, nothing will be done.

Arguments: None.

Results: None.
```

1.280 CloseServiceWindow

```
               CloseServiceWindow


Name: CloseServiceWindow - Close the service window

Synopsis: CloseServiceWindow

Function: Closes the
               service window
               , if it is not already closed.

Arguments: None.

Results: None.
```

1.281 ConvertField

```
ConvertField


Name: CovertField - Change the type of a field

Synopsis: ConvertField ID/A,NewType/A,Listview/S,AltFormat/S
   RC = Success

Function: Changes the type of the specified field. You cannot convert
   text, button or bar fields. The command may be only called in mask
   mode.

Arguments: ID - ID of the field.
   NewType - New type of the field (e.g. String).
   Listview - Determines, wether the field becomes a listview or not.
   AltFormat - Specify if you want to get an alternative format.

Results:
```

1.282 CopyRecord

```
CopyRecord


Name: CopyRecord -- Copy a record to the clipboard

Synopsis: CopyRecord Record/K/N,Unit/K/N
   RC = Success

Function: Copies the active or specified record to the clipboard.  This
   command may be only called in record mode.

Arguments: Record - Number of record to be copied. If not specified, the
   active record will be copied.
   Unit - Number of clipboard unit. Default: 0


Results: RC - Success
```

1.283 CountRecords

```
CountRecords


Name: CountRecords - Get the number of records.

Synopsis: CountRecords Var/K
   RC = Success
```

```
   Result = Number of records

Function: Returns the number of records in the active index of the
   addressed database.  This command may be only called in record mode.

Arguments:

Results: RC - Success
   Result - Number of records in the active index.
```

1.284 CreateField

```
               CreateField


Name: CreateField - Create a field


Synopsis: CreateField Type/A,Listview/S,ID/A
   RC = Success

Function: Creates a new field in the addressed database. The attributes
   will contain default values. It will be both in list and mask hidden.
   To change the attributes for your own needs and to make it visible,
   you may use the command
               SetAttr
               .  This command may be only called in
   mask mode.

Arguments: Type - Type of field.
   Listview - Specifiy, if field will get a listview field.
   ID - ID of new field. The command will fail if the ID already exists.


Results: RC - Success.
```

1.285 CutRecord

```
CutRecord


Name: CutRecord -- Cut a record to the clipboard

Synopsis: CutRecord Record/K/N,Unit/K/N,Force/S
   RC = Success

Function: Copies the active or specified record to the clipboard and
   removes it from the active index.  This command may be only called in
   record mode.
```

```
Arguments: Record - Number of record to be cut. Default: active record.
   Unit - Number of clipboard unit. Default: 0
   Force - Suppress all warnings.


Results: RC - Success
```

1.286 DeleteAllRecords

```
DeleteAllRecords


Name: DeleteAllRecords -- Remove all records

Synopsis: DeleteAllRecords Force/S
   RC = Success

Function: Removes all records from the active index of the addressed
   database. If you do not specify the Force parameter, this command does
   exactly the same as Record/Remove all. That means, that a requester
   may show up, which will ask you, whether you really want to remove all
   records. To prevent this behavior, specify Force.  This command may be
   only called in record mode.

Arguments: Force - Suppress all warnings

Results: RC - Success
```

1.287 DeleteConstant

```
               DeleteConstant


Name: DeleteConstant - Delete a constant (7)

Synopsis: DeleteConstant Name/A
   RC = Success

Function: Deletes the specified constant in the addressed database. If a
   constant of the specified name does not exist, an error will be
   returned.  Constants are most commonly used in Fiasco formulas, but
   may be also used for other purposes. The user may view and change the
   constants of a database using the
               Constants requester
               .

Arguments: Name - Name of the constant to be deleted.

Results: RC - Success.
```

1.288 DeleteLVFieldEntry

```
DeleteLVFieldEntry


Name: DeleteLVFieldEntry - Delete a listview entry

Synopsis: DeleteLVFieldEntry FieldID/A,Record/K/N,ListEntry/A/N
   RC = Success

Function: Deletes the specified entry in a listview field.  This command
   may be only called in record mode.

Arguments: FieldID - ID of a listview field
   ListEntry - Number of the entry in the listview to be deleted
   Record - Number of the record, in which the entry will be deleted. If
   you omit this argument, the active record will be used.

Results: RC - Success
```

1.289 DeleteRecord

```
DeleteRecord


Name: DeleteRecord -- Delete a record

Synopsis: DeleteRecord Record/K/N,Force/S
   RC = Success

Function: Removes the active or specified record from the active index.
   The the active record is remove, another record will be activated. If
   you do not specify the Force parameter, this command does exactly the
   same as Record/Remove. That means, that a requester may show up, which
   will ask you, whether you really want to remove this record. To
   prevent this behavior, specify Force.  This command may be only called
   in record mode.

Arguments: Record - Record to be deleted. The active one will be used if
   not specified.
   Force - No warnings.

Results: RC - Success
```

1.290 Export

```
Export


Name: Export -- Export a database

Synopsis: Export File/A,RecStart=RS/K,RecEnd=RE/K,RecSep=RP/K,
   FieldStart=FS/K,FieldEnd=FE/K,FieldSep=FP,FirstRecIDs/K,
   MarkedOnly/S
   RC = Success

Function: Calls the export function of Fiasco. See the Import/Export
   chapter for more information about exporting. If you do not specify a
   parameter, it will be empty.

Arguments: File - File to write
   RecStart,RecEnd,RecSep,FieldStart,FieldEnd,FieldSep - structure parameters
   LVEntrySep - structure parameter for listview fields
   FirstRecIDs - First Record will contain field IDs
   MarkedOnly - Exports only marked records

Results: RC - Success
```

1.291 Fault

```
Fault


Name: Fault -- Convert an error code to text

Synopsis: Fault ErrorCode/N,Var/K
   RC = Success
   Result = Error text

Function: Converts a Fiasco error number as found in FIASCO.LASTERROR
   into a human-readable format. It will be localized. Because
   FIASCO.LASTERROR can also return Amiga DOS error codes, this function
   also handles these codes.

Arguments: ErrorCode - Fiasco error code

Results: RC - Success.
   Result - Localized error text.
```

1.292 Filter

```
Filter
```

```
Name: Filter - Create a filtered index

Synopsis: Filter SearchInfo/K/A,Index/K
   RC = Success

Function: Creates a filter according to the search criteria specified in
   the search info. It will be activated after it has been created.  This
   command may be only called in record mode.

Arguments: SearchInfo - Name of search info to use.
   Index - Name of index to use. Default: ARexxFilter.fidx.

Results: RC - Success.
```

1.293 Find

```
               Find


Name: Find - Search for data

Synopsis: Find SearchInfo/K/A,Record/K/N,Reverse/S,All=Stem/K,Var/K
   RC = Success
   Result = Found Record


Function: Searches for a pattern that is specified in the SearchInfo.
   This search info may be created with the commands
               NewSearchInfo
                and

               SetSearchField
               .  This command has two modes: You may search for a
   single matching record. In that mode you may specify the Record after
   which the search will begin. The number of the next matching record or
   of the previous matching record, if Reverse was specified, will be
   returned in Result. By calling that command again with the Record
   argument set to the last match, you may search for the next match. See
   the example for the use of this mode.  The second mode allows you to
   search the whole database at one time. If you specify the name of a
   stem variable after the keyword All, this function will search the
   whole database and store the record numbers of the matches in the stem
   elements stem.1, . . . stem.n. The number of matches will be stored in
   stem.count. Important: Do not use that mode if a very large number of
   matches is predictable. In that case, the first mode is faster and
   more memory efficient.  More about searching with ARexx can be found
   in the section
               Searching with ARexx
               .    This command may be only
   called in record mode.

Arguments: SearchInfo - The name of the search info to use.
   Record - Number of record after which the search will begin.
   Mutually exclusive to All.
```

```
   Reverse - Search the database backwards.
   All - Search the whole database once and store all matches in the
   specified stem variable. Mutually exclusive to Record.

Results: RC - Zero if a match has been found
   Result - Number of found record

Example:

/* Find-Example.frx */

/* Create a new search info */

NewSearchInfo Name "ARexxSI"

/* Set the search info to search for the first record
 * in which the field with the ID Test is not empty
 */

SetSearchField SearchInfo "ARexxSI" FieldID "Test" Pattern "?#?"

rc = 0
startrecord = 0

/* Continue searching until nothing is found */

do while rc = 0

   Find SearchInfo "ARexxSI" Record startrecord Var "startrecord"

   if rc = 0 then
   do
       /* The number of the match will be
        * stored in startrecord. Now do something
        * with it.
        */

       say startrecord
   end
end

/* All records done */
```

1.294 FlushRecords

```
FlushRecords


Name: FlushRecords - Clean up memory (7)
```

```
Synopsis: FlushRecords

Function: Tries to free as much memory as possible. This is done by
   removing records from ram, which can be reloaded from disk.

Arguments: None.

Results: None.
```

1.295 GetAttr

```
GetAttr


Name: GetAttr -- Get an Fiasco attribute

Synopsis: GetAttr Object/A,Name/K,Attribute,Stem/K,Var/K
   RC = Success
   Result = Attribute value

Function: Reads the specified attribute and returns it.

Arguments: Object - Object type to be examined.
   Name - Some object types require a name to be specified.
   Attribute - Name of attribute to be returned.
   Stem - The attributes of an object will be returned in a stem variable.


Results: RC - Success.
   Result - Value of the requested attribute.

Objects: Application - Data related to Fiasco.
   Projects - Returns all the open databases in a stem variable.
   No further attributes.
   Project - Data related to one Fiasco database. Name may be
   specified. If you specify the special value Active, the active (in
   contrast to the addressed) project will be examined. Default:
   Addressed project.
   Window - Data related to one Fiasco database window. Name may
   be specified. If you specify the special value Active, the active (in
   contrast to the addressed) project will be examined. Default:
   Addressed project.
   Fields - Returns all the fields of the addressed database in a
   stem variable. No further attributes.
   Field - Data related to a Fiasco field in the addressed
   database. Name required.

Attributes for Application: Version - Internal version of Fiasco in the
   format ver.rev.
   ReleaseVersion - Release version of Fiasco.
   Screen - Name of public screen Fiasco runs on.
   RegUser - Name of registered user of Fiasco.
```

```
Attributes for Project: ARexx - Name of ARexx port.
   FileName - Complete file name of database.
   Path - Path of database.
   File - File name of database.
   Changes - Not equal zero if database has been changed.


Attributes for Window: Left - Left edge of window.
   Top - Top edge of window.
   Width - Width of window.
   Height - Height of window.
   Title - Title of window.
   Screen - Public screen of window.

Attributes for Field: Left - Left edge of field.
   Top - Top edge of field.
   Width - Width of field.
   Height - Height of field.
   ARexxScript - ARexx script of field.
   PickerARexx - ARexx script for picker button.
   Formula - Formula for field.
   Type - Type of field.
   ListLeft - Left edge in list window.
   ListWidth - Width in list window.
   Shortcut - Keyboard shortcut for field.
   InitContType - Type of initial content. One of LAST, KEY or OWN.
   InitContValue - Value of initial content.
   MaxChars - Max chars of field. Not supported by all fields.
   MinValue - Minimum value of field. Not supported by all fields.
   MaxValue - Maximum value of field. Not supported by all fields.
   Format - Format of field. Not supported by all fields.
   Labels - Labels of field. Not supported by all fields. Will be
   returned in stem variable.
   Precision - Precision of field. Not supported by all fields.
   Command - Command for field. Not supported by all fields.
   Stack - Stack for field. Not supported by all fields.
   Virtual - Status of virtual flag of field. 1 or 0.
   ListHidden - Is field hidden in the list? 1 or 0.
   Hidden - Is field hidden in the mask? 1 or 0.
   ReadOnly - Is field read only? 1 or 0.
   Listview - Is field a listview field? 1 or 0.
```

1.296 GetConstant

```
                GetConstant


Name: GetConstant - Get the value of a constant (7)

Synopsis: GetConstant Name/A,Var/K
   RC = Success
   Result = Value of constant

Function: Reads the value of a constant in the addressed Fiasco database.
```

```
   If the constant does not exist, an error will be returned.  Constants
   are most commonly used in Fiasco formulas, but may be also used for
   other purposes. The user may view and change the constants of a
   database using the
                Constants requester
                .

Arguments: Name - Name of the constant.

Results: RC - Success.
   Result - The value of the specified constant
```

1.297 GetField

```
                GetField


Name: GetField -- Read the content of a field

Synopsis: GetField FieldID,Record/K/N,ListEntry/K/N,ListEntryCount/S,
   ExtFormat/S,Stem/K,Var/K
   RC = Success
   Result = FieldContent

Function: Reads the content of the specified Field in the active or
   specified record and returns it in result.  If you specify a Stem
   variable, the contents of all fields will be set into that ARexx stem
   variable. The stem elements are the field IDs. You may change after
   this call the values of some variables and use
                SetField
                 with its Stem
   argument to write the values back to the database. Please note that
   with the Stem argument the commands need much more execution time than
   the simple calls. Thus, you should use these commands only if you need
   to set several fields at a time.  This command may be only called in
   record mode.

Arguments: FieldID - ID of Field
   Record - Number of record
   ListEntry - Only if field is a listview field. Specifies the
   number of the entry in the listview, whose content should be returned.
   Counting from 1.
   ListEntryCount - Only if field is a listview field. Returns the
   number of entries in the listview.
   ExtFormat - If specified, the field content will be returned in an
   extended format, suitable for printing, etc. This extended format may
   use locale settings, etc. You should not make any assumptions about
   the resulting format (for example by parsing it in ARexx scripts). See
   also the Result table. (Fiasco 2.2)
   Stem - Return all contents in the specified stem variable.

Results: RC - Success
    Result - is equal to the current content of the field, if rc = 0.
```

```
   The format (In brackets: ExtFormat):
   String - the string itself
   Integer - the number itself
   Float - the fp number
   Slider - the value of the slider
   Cycle - the number of the active label (the active label string)
   Date - the date in the format DD.MM.[YY]YY (in the current locale format)
   Time - the time in the format HH:MM:SS (in the current locale format)
   Extern - the string itself.
   Datatyp.- the string itself.
   Var String- the string itself. Newlines will be converted to *n.
    If the field is a listview field, you have to use the ListEntry
   argument to select the entry to be returned. The format will be then
   the format of the underlying fieldtype.

Notes: Before release 2.2, this document claimed that the date and time
   field data would be returned in the current locale format. However, it
   has always been the case, that - if ExtFormat is not specified - these
   values are formatted independantly from the locale setting.
```

1.298 GetRecordMark

```
GetRecordMark


Name: GetRecordMark - Check the mark of a record

Synopsis: GetRecordMark Record/K/N,Var/K
   RC = Success
   Result = Marked or not marked

Function: Returns the current state of the mark of a record in the
   addressed Fiasco database.  This command may be only called in record
   mode.

Arguments: Record - Number of record to be checked. Default: active.

Results: RC - Success
   Result - 1 if record is marked, otherwise 0.
```

1.299 HideProject

```
HideProject


Name: HideProject - Hide a Fiasco project

Synopsis: HideProject Project/K
   RC = Success
```

```
Function: Closes all windows of the addressed or specified project.

Arguments: Project - Name of project to be hidden. May be the file name
   or the full path.

Results: RC - Success
```

1.300 Import

```
                Import


Name: Import -- Import a database

Synopsis: Import File/A,RecStart=RS/K,RecEnd=RE/K,RecSep=RP/K,
   FieldStart=FS/K,FieldEnd=FE/K,FieldSep=FP,

   SkipLines/K,StartLine/N/K,FirstRecIDs/K,AppendFields/S
   RC = Success

Function: Calls the import function of Fiasco. The specified file will be
   imported into the current project using the specified parameters. For
   more information on import and export see section
                Import and Export
                .
    You may also use the escape sequences of Fiasco. If you do not
   specify a parameter, it will be empty.

Arguments: File - Name of File
   RecStart,RecEnd,RecSep,FieldStart,FieldEnd,FieldSep - the
   structuring characters
   SkipLines - Comment introducer
   StartLine - Length of initial comment
   FirstRecIDs - First Record contains IDs
   AppendFields - Append new fields


Results: Result - Success

Notes: The option Overwrite old project of the import requester is not
   directly supported. You have to emulate it using
                Clear
                .
```

1.301 LoadDTFieldObject

```
LoadDTFieldObject
```

```
Name: LoadDTFieldObject - Load a datatypes field

Synopsis: LoadDTFieldObject FieldID/A
   RC = Success

Function: Loads the contents of a datatypes field, which was "deferred".
   This command may be only called in record mode.

Arguments: FieldID - ID of datatypes field

Results: RC - Success
```

1.302 LockGUI

```
               LockGUI


Name: LockGUI - Lock Fiasco's GUI

Synopsis: LockGUI

Function: Locks the GUI of Fiasco. The pointer will appear as a "wait
   clock". After locking the GUI, the ARexx script can run, without the
   danger of being influenced by the user. Before the script ends,

               UnlockGUI
                must be called in order to give the control back to the
   user. LockGUI and UnlockGUI may be nested.

Arguments: None.

Results: None.
```

1.303 MarkMatch

```
MarkMatch


Name: MarkMatch - Mark matching records

Synopsis: MarkMatch SearchInfo/K/A
   RC = Success

Function: Marks the records that match the specified search info. The old
   marks will be overwritten.  This command may be only called in record
   mode.

Arguments: SearchInfo - Name of the search info to be used.
```

```
Results: RC - Success
```

1.304 MarkRecord

```
MarkRecord


Name: MarkRecord -- Control marking of records

Synopsis: MarkRecord Record/K/N,All/S,Set/S,UnSet/S,Toggle/S
   RC = Success

Function: MarkRecord can be used to change the marking of one (specified
   or active) or all records in the addressed database. You may set,
   unset or toggle the marking.  This command may be only called in
   record mode.

Arguments: Record - Number of record to be affected. Default: Active.
   All - Affect all records. Mutually exlusive to Record
   Set - Set the mark.
   UnSet - Remove the mark.
   Toggle - Marked will become unmarked and vice-versa.


Results: RC - Success
```

1.305 MenuControl

```
MenuControl


Name: MenuControl -- Switch pull down menus on or off (8)

Synopsis: MenuControl On/S,Off/S
   RC = Success

Function: This command can be used to switch the pull down menus locally
   to one database on or off to limit the possible actions by the user.

Arguments: On - Switch menus on
   Off - Switch menus off

Results: RC - Success
```

1.306 MoveRecord

```
MoveRecord


Name: MoveRecord -- Change a record's position

Synopsis: MoveRecord Record/K/N,To/A/N
   RC = Success

Function: Moves the active or specified record in the active index to the
   new specified position. This command does not work if the active index
   does automatic sorting. This command does not affect other indices.
   This command may be only called in record mode.

Arguments: Record - Number of record to be moved. If not specified, the
   active record will be used.
   To - Number of record after which the record will be inserted.
   You have to specify the number of the record before the record was
   moved.

Results:
```

1.307 New

```
New


Name: New -- Create a new project

Synopsis: New Project/K,Iconified/S,Var/K
   RC = Success
   Result = New ARexx Port

Function: Creates a new database project. If Iconified is not specified,
   a new window is opened and activated. It is then entirely empty.

Arguments: Project - File name of new project. If not specified,
   Unnamed.fdb will be used.
   Iconified - The project window will be not opened.


Results: rc - Success.
   Result - The name of the new ARexx port. Can be used with Address.
```

1.308 NewSearchInfo

```
               NewSearchInfo


Name: NewSearchInfo - Create a new search info
```

```
Synopsis: NewSearchInfo Name/K,Fields/S,Formula/K,Var/K
   RC = Success
   Result = Search info name

Function: Creates a new search info. You may specify an own name for the
   new search info with the argument Name. If a search info with that
   name should already exist, it will be deleted. If you do not specify
   the Name argument, this function will create an unique name and return
   it in Result.  If you want to search for fields with that search info,
   you will have to use the command
               SetSearchField
                to set the fields and
   patterns to search for. If you want to search by a formula, you have
   to specify it now with the Formula argument.

Arguments: Name - Name of the search info to create. If not specified, a
   unique name will be created.
   Field - Specify if you want to search by fields.
   Formula - If you want to search by formula, specify the formula after
   this keyword.

Results: RC - Success
   Result - Name of the newly created search info
```

1.309 Open

```
Open


Name: Open - Open a Fiasco database

Synopsis: Open File/A,New/S,Iconified/S,Force/S,Var/K
   RC = Success
   Result = ARexx Port Name


Function: Tries to read a fiasco database. If you specify only the File
   argument, the data will be loaded into the addressed project. If there
   are changed data in it, a warning requester will be opened before. If
   you want to suppress this behaviour, specify the Force argument. If
   you specify New, Fiasco will load the data into a new project. If you
   specify Iconified, Fiasco will leave the project window closed.

Arguments: Name - File name of database to be opened.
   New - Create a new project.
   Iconified - Do not open the new project window. Only in
   combination with New.
   Force - No warnings.

Results: RC - Success.
   Result - Name of ARexx port of the project the database has
   been loaded into. Useful with New.
```

1.310 OpenListWindow

```
OpenListWindow


Name: OpenListWindow -- Open the list window

Synopsis: OpenListWindow

Function: Opens the list window of the currently addressed project. If it
   is already open, nothing will be done.

Arguments: None.

Results: None.
```

1.311 OpenServiceWindow

```
                OpenServiceWindow


Name: OpenServiceWindow - Open the service window

Synopsis: OpenServiceWindow

Function: Opens the
                service window
                , if it is not already open.

Arguments: None.

Results: None.
```

1.312 PasteRecord

```
                PasteRecord


Name: PasteRecord -- Paste a record from the clipboard

Synopsis: PasteRecord Record/K/N,Unit/K/N,Inactive/S
   RC = Success

Function: Inserts a record from the clipboard into the addressed Fiasco
   database. The record may be written to the clipboard by
```

```
                CopyRecord
                 or

                CutRecord
                .  This command may be only called in record mode.

Arguments: Record - Number of record after which the new record will be
   inserted. Default: the active record.
   Unit - Number of clipboard unit. Default: 0.
   Inactive - Do not activate the new record.


Results: RC - Success
```

1.313 Progress

```
                Progress


Name: Progress -- Show a progress bar

Synopsis: Progress Done/A/N,Max/A/N
   RC = Success

Function: Displays a nice progress bar in the service window, as known of
   Sort or Open Project. You should reset the status gadget with

                ResetStatus
                 when the operation has completed.

Arguments: Done -- the number of data items currently processed.
   Max -- the number of all data items.

Results: None.
```

1.314 Quit

```
Quit


Name: Quit -- Exit Fiasco

Synopsis: Quit

Function: Closes all databases and exits Fiasco. No warnings will be
   done. After this command you cannot access any ARexx ports of Fiasco.

Arguments: None.
```

```
Results: None.
```

1.315 ReadSettings

```
ReadSettings


Name: ReadSettings -- Read Fiasco settings (8)

Synopsis: ReadSettings File/A
   RC = Success

Function: Reads Fiasco settings from the specified file and activates
   these settings.

Arguments: File - File to read Fiasco settings from.

Results: RC - Success
```

1.316 RecompileFormulas

```
               RecompileFormulas


Name: RecompileFormulas - Update all formulas (7)

Synopsis: RecompileFormulas
   RC = Success

Function: Recompiles (i.e. updates) all formulas in the addressed
   database. This especially useful after a call to
               SetConstant
               .

Arguments: None.

Results: RC - Success.
```

1.317 RequestChoice

```
RequestChoice


Name: RequestChoice - Request a choice from the user
```

```
Synopsis: RequestChoice Body/A,Gadgets/A,Title/K,Var/K
   RC = Success
   Result = Choice

Function: Creates an intuition easy-requester with the specified
   parameters. Works very similar to the CLI command Requestchoice. The
   differences: Slightly different parameters, puts the requester up on
   Fiasco's screen.

Arguments: Body - Main text of requester.
   Gadgets - Gadgets at the bottom of requester. Each choice
   must be separated by a |.
   Title - Title of requester.

Results: RC - Success.
   Result - Number of selected gadget, 0 for the rightmost one.
```

1.318 RequestField

```
RequestField


Name: RequestField - Request a field from the user

Synopsis: RequestField Default/K,Text/A,Var/K
   RC = Success
   Result = Selected field

Function: Opens a requester with a list of all fields of the adressed
   project. The requester can display an additional message given in the
   Text argument. The user can select one field and click on Ok or can
   Cancel the requester.

Arguments: Default - The field that will be selected when the requester opens.
   Text - The text to be displayed.

Results: RC - Success. Will be 5 if user cancelled requester.
   Result - ID of the selected field.
```

1.319 RequestFile

```
RequestFile


Name: RequestFile -- Open a file requester

Synopsis: RequestFile File,Pattern/K,Title/K,SaveMode/S,DrawersOnly/S,NoIcons/S,
   ProjectRelative/S,Var/K
   RC = Success
   Result = Selected file
```

```
Function: Puts up an ASL file requester. Works very similar to the CLI
   command Requestfile. The differences: Slightly different parameters,
   puts the requester up on Fiasco's screen.

Arguments: File - Initial File including path for the requester
   Pattern - Initial Pattern
   Title - Title for the requester
   SaveMode - Activates savemode: Black background, no selection
   via doubleclick
   DrawersOnly - Displays only Drawers
   NoIcons - Filters Icons
   ProjectRelative - The requested file will be relative to the addressed
   database.

Results: RC - Success. Will be 5 if user cancelled requester.
   Result - Selected file.
```

1.320 RequestNumber

```
RequestNumber


Name: RequestNumber -- Request a number

Synopsis: RequestNumber Default/K/N,Title/K,Text/K/A,Var/K
   RC = Success
   Result = Requested number

Function: Asks the user to input an integer number. He may cancel the
   request. You can supply additional information using the Text
   argument.

Arguments: DefaultValue - Value of integer gadget on startup. Will be
   zero if not specified.
   Title - Optional. Window title of requester.
   Text - Additional text to display in requester. May contain
   newlines (*n). Please note that this argument is required and must be
   specified with a leading keyword. This is for compability with future
   versions of Fiasco which may not require the Text argument.

Results: RC - Success. Will be 5 if user cancelled requester.
   Result - Requested number.
```

1.321 RequestString

```
RequestString


Name: RequestString -- Request a string from the user
```

```
Synopsis: RequestString Default/K,Title/K,Text/K/A,Var/K
   RC = Success
   Result = Requested string

Function: Asks the user to input a string. He may cancel the request. You
   can supply additional information using the Text argument.

Arguments: DefaultValue - Value of string gadget on startup. Will be
   empty if not specified.
   Title - Optional. Window title of requester.
   Text - Additional text to display in requester. May contain
   newlines (*n). Please note that this argument is required and must be
   specified with a leading keyword. This is for compability with future
   versions of Fiasco which may not require the Text argument.

Results: RC - Success. Will be 5 if user cancelled requester.
   Result - Requested string.
```

1.322 ResetStatus

```
ResetStatus


Name: ResetStatus -- Reset the status gadget

Synopsis: ResetStatus

Function: Resets the status gadget in the service window. It will show
   the default information after this call.  If the service window is not
   open, nothing will be done.

Arguments: None.

Results: None.
```

1.323 RevealProject

```
RevealProject


Name: RevealProject -- Reveal a hidden project

Synopsis: RevealProject Project/K
   RC = Success

Function: Open all windows of a hidden Fiasco project. This project will
   get the active one. If a project is specified, this project will be
   revealed, otherwise the addressed project.
```

```
Arguments: Project - File name or full path of project.

Results: RC - Success
```

1.324 Save

```
Save


Name: Save -- Save a Fiasco database

Synopsis: Save
   RC = Success

Function: Saves the addressed Fiasco database under the old name on disk.

Arguments: none.

Results: RC - Success.
```

1.325 SaveAs

```
SaveAs


Name: SaveAs -- Save a Fiasco database under a new name.

Synopsis: SaveAs Name/K
   RC = Success

Function: Saves the addressed database under a given name on disk. If you
   specify a new name after the Name keyword, this name will be used. If
   you do not specify Name, the user will be prompted with a file
   requester to specify a new name.

Arguments: Name - New file name. If not specified, a file requester will
   be opened.

Results: RC - Success.
```

1.326 SaveSettings

```
SaveSettings


Name: SaveSettings -- Save Fiasco settings (8)
```

```
Synopsis: SaveSettings File/A
   RC = Success

Function: Saves the active Fiasco settings to the specified file.

Arguments: File - File to save the settings to.

Results: RC - Success
```

1.327 SetAttr

```
SetAttr


Name: SetAttr -- Set an Fiasco attribute

Synopsis: SetAttr Object/A,Name/K,Attribute,Value/A
   RC = Success

Function: Sets the specified attribute.

Arguments: Object - Object type to be set.
   Name - Some object types require a name to be specified.
   Attribute - Name of attribute to be set.
   Value - New value for attribute.

Results: RC - Success.


Objects: Field - Data related to a Fiasco field in the addressed
   database. Name required. May be only used in mask mode.

Attributes for Field: Left - Left edge of field.
   Top - Top edge of field.
   Width - Width of field.
   Height - Height of field.
   ARexxScript - ARexx script of field.
   PickerARexx - ARexx script for picker button.
   Formula - Formula for field.
   ListLeft - Left edge in list window.
   ListWidth - Width in list window.
   Shortcut - Keyboard shortcut for field.
   InitContType - Type of initial content. One of LAST, KEY or OWN.
   InitContValue - Value of initial content.
   MaxChars - Max chars of field. Not supported by all fields.
   MinValue - Minimum value of field. Not supported by all fields.
   MaxValue - Maximum value of field. Not supported by all fields.
   Format - Format of field. Not supported by all fields.
   Precision - Precision of field. Not supported by all fields.
   Command - Command for field. Not supported by all fields.
   Stack - Stack for field. Not supported by all fields.
   Virtual - Status of virtual flag of field. 1 or 0.
   ListHidden - Is field hidden in the list? 1 or 0.
   Hidden - Is field hidden in the mask? 1 or 0.
```

```
   ReadOnly - Is field read only? 1 or 0.
```

1.328 SetConstant

```
                SetConstant


Name: SetConstant - Set a constant (7)

Synopsis: SetConstant Name/A,Value/A/F
   RC = Success

Function: Sets a constant to the specified value in the addressed
   project. If a constant of the same name already exists, it will be
   overwritten, otherwise a new constant will be created.  Constants are
   most commonly used in Fiasco formulas, but may be also used for other
   purposes. The user may view and change the constants of a database
   using the
                Constants requester
                .  If the formulas of the database are
   supposed to use the new or changed constant, you have to call

                RecompileFormulas
                 after this command.

Arguments: Name - Name of the constant to set.
   Value - Value for the constant.

Results: RC - Success.
```

1.329 SetField

```
                SetField


Name: SetField - Set the content of a field

Synopsis: SetField FieldID,Record/K/N,ListEntry/K/N,CreateListEntries/S,
   ExtFormat/S,Stem/K,Cont/F
   RC = Success

Function: Sets the content of the specified field in the active or
   specified record to the specified content.  If Stem is used, Fiasco
   will copy the values of the ARexx variables with the names
   stemname.fieldID to the respective fields. If a variable is not set,
   the field will not be changed. Useful with
                GetField Stem
                 see there for
```

```
   a description.  Entries in listview fields are normally neither
   created, nor removed by this command. Thus, if you have a listview
   field with three entries and you call SetField Stem MyStem with MyStem
   initialized with four entries for the listview field, only the first
   three will be copied and the fourth entry will be ignored. If you have
   initialized MyStem to contain less entries for the listview than it
   contains, the last entries in the listview will be left unchanged.
   However, if you use the CreateListEntries switch, this command will
   remove all old entries from the listview and replace them by the
   entries in the stem variable.    This command may be only called in
   record mode.

Arguments: FieldID - ID of a single field to be set
   Record - Number of record
   ListEntry - Only if field is a listview field. Number of the
   entry in the listview to be set to the new value. Counting from 1.
   CreateListEntries - See above for explanation. Only in combination
   with Stem. New in Fiasco 2.2.
   ExtFormat - If specified, SetField uses the extended format that
   is returned by
               GetField ExtFormat
               . New in Fiasco 2.2. Cont - New
   content of the field. This argument takes the whole input inclusive
   spaces. The interpretation of this arg depends on the fieldtype (In
   brackets: ExtFormat):
   String - is copied directly
   Integer - Numbers are read directly, other things are 0
   Float - dto.
   Boolean - 1 or TRUE = selected, 0 or FALSE = not selected
   Slider - Number is read. Bad numbers will be adjusted.
   Cycle - Number or name of label is taken.
   Date - Date in the format DD.MM.[YY]YY (current locale format)
   Time - Time in the format HH:MM:SS (current locale format)
   Extern - is copied directly
   Datat. - is copied directly
   Var String - is copied directly. The string *n will be converted to a
   line break. This works, however, only if the string is enclosed in
   quotes.
    If the field is a listview field, use ListEntry to set the content of
   a particular entry. Use the format of the underlying fieldtype. To add
   entries to listviews, either use the CreateListEntries switch or use

               AddLVFieldEntry
               .

Results: RC - Success

Notes: Before release 2.2, this document claimed that the date and time
   field data would be required in the current locale format. However, it
   has always been the case, that - if ExtFormat is not specified - these
   values are formatted independantly from the locale setting.
```

1.330 SetMode

```
SetMode


Name: SetMode - Set the working mode

Synopsis: SetMode Mask=MaskMode/S,Record=RecordMode/S
   RC = Success

Function: Sets the addressed Fiasco project to the specified mode.

Arguments: MaskMode - Activate mask mode.
   RecordMode - Activate record mode.


Results: RC - Success.
```

1.331 SetSearchField

```
SetSearchField


Name: SetSearchField - Add a field to a search info

Synopsis: SetSearchField SearchInfo/K/A,FieldID/K/A,Blur/K/N,Pattern/K/A/F
   RC = Success

Function: Adds a field and the pattern to a search info in field mode.
   The specified field must match the pattern in order to have the search
   info match a record.

Arguments: SearchInfo - Name of the search info to be modified.
   FieldID - ID of the field to be searched.
   Blur - Blur factor. When not specified, blurred search will be turned off.
   Pattern - Pattern for the field.


Results: RC - Success
```

1.332 SetStatus

```
SetStatus


Name: SetStatus - Show a status string

Synopsis: SetStatus Text/A
   RC = Success
```

```
Function: Displays the specified text in the status gadget of the service
   window.

Arguments: Text - Text to be shown.

Results: RC - Success.
```

1.333 Sort

```
Sort


Name: Sort -- Sort a database

Synopsis: Sort Fields/A/M,Descending/S,Index/K
   RC = Success

Function: Creates an sorted index for the addressed database according to
   the alphabetical or equivalent priority of the contents of the
   specified fields.  This command may be only called in record mode.

Arguments: Fields - Fields after which the database will be sorted. First
   field has the highest priority.
   Descending - Sort backwards.
   Index - Name of index to be created. Default:
   ARexxSort.fidx.

Results:
```

1.334 UnlockGUI

```
                UnlockGUI


Name: UnlockGUI - Unlock Fiasco's GUI

Synopsis: UnlockGUI
   RC = Success

Function: Unlock the GUI, which has been previously locked using
                LockGUI
                .
   The user has again access to Fiasco. LockGUI and UnlockGUI may be
   nested.

Arguments: None.

Results: RC - Not equal zero if GUI was not locked.
```

1.335 Example Projects

```
                Example Projects

The directory Databases of the Fiasco distribution contains several
Fiasco projects. Some of them may be also used for own purposes. The
following sections contain descriptions of some of the databases included
in the Fiasco 2.2 distribution.

                Organizer
                 Addressbook and reminder

                FamilyTree
                 Data about your ancestors

                PD Disks
                 Cataloges your PD disks

                Videos
                 Catalog of video tapes

                Picture Database
                 Manages your pictures

                Mulitmedia Database
                 A mini-encyclopedia

                Mailing List Archive
                 Discussion archive for the WWW
```

1.336 Organizer

```
Organizer

The organizer directory contains two databases: An address book database
and a appointments databases, that can be used as a reminder program.

    The address book is an extended version of the addresses database of
previous Fiasco releases. It contains the classic fields such as Name,
Address, Phone, etc.

    The fields for Phone, Fax or Zipcode are string fields, because they
also have to take characters like "/" or must have a leading "0" (which
would be swallowed by a integer field).

    Furthermore, action buttons are assigned to the fields Phone, Fax and
eMail. All buttons call ARexx scripts. The script for the phone field
calls the script dial.frx that tries to call the phone number with a
modem. When the connection is being built, a requester opens with the
response gadget Hang up. You may take up your phone and click on Hang up
to be able to speak with the person. More details about dial.frx are
```

directly contained in the script as comments. If you want to use it with your own address database, you can simply adopt it to it.

The fax button can be used to send a fax. The script fax.frx called by it currently supports only the program STFax. If you have another fax program with an ARexx port, adding support for it to the script, should be not difficult.

The mail button calls a mailer to send an eMail to the eMail address. The script mail.frx currently supports only the mailer YAM by Marcel Beck.

The other database in the directory can be used to manage appointments. Simply fill in your appointments. You may set an explicit date for appointments or only a weekday. Whether you use the date or a weekday is controlled by the boolean fields at the right side of the respective fields.

The button Check Appointments scans the database for appointments that are current. With the Ann. field you may set the time in days before the appointment at which you will be reminded about a appointment.

1.337 FamilyTree

FamilyTree

The family tree consists of the projects "persons.fdb" and "families.fdb". "persons.fdb" contains all persons, which are used in the family tree. You may also enter sex, date of birth, etc. here. These data are used by "families.fdb" with relations, to get names of spouses, children, etc. Additionally, there are fields for marriage and divorce. Caused by the intensive use of relations, this project only contains 7 "real" fields, which are stored on disk. The other 6 fields are loaded from "persons".

1.338 PD Disks

PD Disks

This database stores the descriptions of programs on PD libraries, such as the Fish Disks. It contains fields for library name, disk number, program name, program version, description and author. The button Scan disk can be used to import a contents file of a library automatically into the database. The ReadFish.rexx script currently supports the formats of Fish Disks, SaarAG disks and of PD disks of the german Amiga Magazin. It also supports contents files, which have been joined to one big file.

1.339 Videos

Videos

The video database can be used to manage your homevideo collection. The database consists of two projects: "movies.fdb" and "tapes.fdb". "Movies" takes the informations for each movie (Genre, Director, etc.). The field "Tape" connects each film with one tape, which can be found in "tapes". Here is the play length of each tape defined. A relation calculates the used space on the tapes.

1.340 Picture Database

Picture Database

Fiasco's picture database uses datatypes fields, thus you need Amiga OS 3.0 to use it. It's mask contains eight datatypes fields, which are used to display thumbnail images of the pictures. The picture database is controlled by ARexx scripts.

When the database inserts a new file, which can be done with the Add Dir or Add File button, the program CreateThumbnail creates a smaller version of the picture in IFF format and puts it in the TN directory. CreateThumbnail also uses the datatypes.library. Thus, all you can insert all picture formats, for which you have installed a datatype, into the picture database. However, CreateThumbnail cannot handle HAM or EHB pictures.

With Add File you can pick a picture with a file requester, which will be automatically inserted into the database.

Add Dir serves to insert the pictures in a whole directory. After you have selected the directory to scan, another requester opens, where you can choose, whether you only want to scan the selected directory, all subdirectories if it or even the Lha archives in it. The picture database handles Lha archives transparently, you do not need an archive file handler. With some modifications to the ARexx scripts, you may also use other archive programs, such as LZX.

Below each datatypes field, there are three buttons for controlling the field. Dis uses MultiView to display the picture. Of course, by changing the ARexx script display.rexx you may also choose another program. Inf displays additional information about the picture, such as size and depth. From here, you may also choose to display the picture, to display the picture with VT or to copy the picture to another location. Del finally, removes the picture from the database.

The Search button at the upper right of the mask can be used to search for a filename.

1.341 Multimedia Database

Multimedia Database

This is a database that uses the multimedia features of Fiasco for a kind of encyclopedia. The data included as example in this database are a bit spare for a encyclopedia, though. However, this databases shows what is possible and is open for your own data.

Each record represents an entry for a Term. The name of the term can be specified in the field at the upper left of the mask. Below it, there is a var string field that can be used to specify a longer text describing the item.

No encyclopedia without cross references. The listview field at the lower left may contain several names of other terms in the database. To go to any reference, you have to activate it in the listview and click on the Goto Cross Reference button that starts an ARexx script that uses Fiasco's search function to search for the specified term. The Cross References listview uses the Select Only field attribute. If you want to edit the entries, you have to deactivate this attribute.

The right part of the mask serves for the multimedia. For each record you can specify Documents. Selecting any entry in the listview will cause the document to be automatically displayed in the datatypes field with the ID Document_Display below. The documents may not be only pictures, you can also specify sound files and all other files for which datatypes are available.

The mechanism behind this multimedia part is a bit more complicated than the other parts of the mask, but also easy to understand. The Documents listview displays only names for the documents that can be freely assigned to the documents. Then, there is another listview, that is normally hidden. Its ID is Doc_Files. There you have to specify the file names for all the documents specified in the Documents listview. To display the selected document in the datatypes field, a formula is used. It is placed in the datatypes field, which has also set the virtual attribute, because the contents of this field are only important for runtime. The formula is:

```
active(documents) != -1 ? doc_files[active(documents)] : ""
```

Thus, the
datatypes field looks, whether a entry is selected in the Documents listview. If it is, the file name at the same position in the doc_files listview is copied into the datatypes field, which will display it. If no entry is active, an empty string will be copied into it, which causes no file to be displayed.

To edit these data, you have to switch off the Select Only attribute of the Documents listview and you have to reveal the Doc_files listview.

1.342 Mailing List Archive

Mailing List Archive

If you run a mailing list, this Fiasco database may be useful for you. It helps you to archive all messages from the list and -- which is the most

important point -- creates automatically a html version of the archive, ready for use on a web page. Thus, the visitors of your web page will be able to view the contents of the mailing list. To create the database, you have to export the mails from your mail program. You have to ensure, that the mails contain the complete headers. Otherwise, the project will not be able to import the mails. With MicroDot II you can export mails including the headers this way: Open the message reader of the individual mail, select (whole message) in the listview in the upper right of the window and click on Save.

To import this mail into the mailing list archive, click on the button Import Mail in the mask window. You will be prompted to specify the file to import. If you have saved several mail files in one directory, you can use the Scan Directory button. When the mailing list archive imports the mails, it creates for each mail one record. The header data for each mail are displayed at the right side of the mask and the mail body at the bottom of the mask.

The Fiasco mailing list archive allows you to customize the layout of the html output. Headlines and footers for each page the archive creates, can be defined in the files tpl1.html and tpl2.html. These files will be inserted at the start or at the end, respectively, of all pages. The Fiasco distribution comes with example versions of these files. The name of the mailing list, document colors and some other settings may be directly defined in the createhtml.frx file, which is also located in the project directory.

To create the html pages, click on the Create HTML button. It is recommended, that you create a new directory solely for the output of the archive. After exporting, the main page can be accessed with directory/index.html. Now you can watch with any WWW browser the pages locally on your computer. With an FTP program, you may transfer these files to your internet site.

A "real life" example of this database can be found on the Fiasco Support Site:
http://www.amigaworld.com/support/fiasco

1.343 Legal Things

Legal Things

The Program "Fiasco" and associated files, hereafter called Fiasco, are provided "as is". No representations or warranties are made regarding to accuracy, reliability or correctness of Fiasco, either expressed or implied. In no case am I responsible for any damages or data losses caused by this software. If you store important data on your computer, you should create in any case backup copies of these data!

Fiasco is not Public Domain. I reserve all rights.

Fiasco may be redistributed under the following conditions:

· The program package has to be complete. Starting with release 2.0, the

distribution has been due to its size divided into several parts. The archive Fiasco_main.lha contains the main program, libraries, locale catalogs and example databases. The archives Fiasco_doc_eng.lha, Fiasco_doc_deu.lha and Fiasco_doc_ita.lha contain the Fiasco documentation in the respective languages in AmigaGuide and TeX-DVI format. The section

file list

contains a listing of all archives of the Fiasco 2.1 distribution. Because of the seperate archives for documentation, the main archive does not contain any documentation. Thus, you must distribute at least one of the language archives. It is still strongly recommended, that you distribute all archives of the Fiasco distribution, especially when you distribute Fiasco on a CD-ROM. Distributing Fiasco in unarchived form is allowed, as long you keep the conditions stated above.

- Fiasco may not be distributed for commercial purposes without a written permission by the author. This includes the distribution of Fiasco for excessively high prices. You may only charge a small fee for media and copying. The distribution on CD-ROMs is allowed, if the price of a single CD-ROM is not higher than USD 20 or DM 30. Distribution on cover disks or cover CDs of magazines is allowed, if the price of the magazine is less than USD 10 or DM 12 in the case of floppy disks or USD 12 or DM 16 in the case of compact disks.

I grant herby special permission to distribute Fiasco on the "Meeting Pearls" CD-ROMs and on the "Aminet" CD-ROMs.

There is a special floppy disk based distribution of Fiasco 2.1. It consists of two disks with archived contents and an installer script which has been adapted to handle extraction and installation automatically. This distribution is not available via Aminet. If you are interested in it, send me a mail and the shipping costs listed in the

shareware section

. You will also recieve this distribution if you order a registered Fiasco version on DD disks.

If you include Fiasco in your PD collection, coverdisk, etc. and a copy is left over, you may feel free to send me this copy.

The keyfile, which you recieve after registering for Fiasco, is not freely redistributable. You may use it only on your own computer system. Manipulating the keyfile is also prohibited.

textfield.gadget 3.1 is Copyright © 1995 by Mark Thomas. The complete textfield.gadget distribution, including developer information, can be found on Aminet or on the Aminet Set 2C CD-ROM in dev/gui/textfield.lha.

gtlayout.library is by Olaf Barthel. It may be freely distributed and freely used. New versions of the gtlayout.library appear frequently with 'term'.

WBPath is Copyright © 1994 by Ralph Babel. The WBPath developer information is distributed with Fiasco in the Development/WBPath directory.

LX is written by Jonathan Forbes and Copyright © 1993 by Xenomiga Technology. LX is used by the installer script of the Fiasco disk distribution. The complete LX distribution can be found in Aminet or on the Aminet Set 2A CD-ROM in util/arc/lx103.lha.

Installer and Installer project icon Copyright © 1995-1996 ESCOM AG. All Rights Reserved. Reproduced and distributed under license from ESCOM AG.

Installer software is provided "as-is" and subject to change; no warranties are made. All use is at your own risk. No liability or responsibility is assumed.

Shareware

1.344 Shareware

Shareware

Starting with release 2.0, Fiasco is shareware. That means, that you are allowed to test Fiasco for a period of 30 days. If you want to continue using Fiasco after that period, you will have to register for Fiasco. You will recieve a "keyfile", which allows you to save more than 15 records.

Price List

Registration fee for Fiasco 2.2	USD 25.00	DM 30.00
Shipping costs Europe	USD 4.00	DM 5.00
Shipping costs International	USD 6.00	DM 8.00
Shipping costs eMail	free	free

Shipping costs include two DD disks or one HD disk, which contain the keyfile and the latest release version of Fiasco. You may choose to recieve your keyfile by eMail. In that case you do not have to pay any shipping costs, however, you will only recieve the keyfile, not the latest Fiasco version.

Paying methods

There are three methods of paying for Fiasco:

- Cash: Simply put the money into your letter.
- Euro cheques: Put the cheque in your letter. Please do not send other cheques!
- Transfer to my bank account:
  Kasseler Sparkasse; bank code (BLZ) 520 503 53; Account no. 1100353258

```
   Please include your name and address in the transfer. This
   is the only possible paying method if you order by eMail. Delivery
   will start when I have recieved the money.

The only accepted currencies are German DM or US Dollar.

    Please send the filled-in registration form, which is included in the
Fiasco distribution, to one of the following addresses:

Nils Bandener
Dekanatsgasse 4
D-34369 Hofgeismar
Germany

Internet: Nils@dinoex.sub.org


Senders of Gifts for Fiasco 1.x

If you already have sent me a gift for Fiasco 1.x until December, 31st
1996, you can get a free keyfile for Fiasco 2.2. You do not have to pay
anything if you choose to recieve the keyfile by eMail. If you want to
recieve it by snail mail, you will have to pay the shipping costs listed
above.


Polish Residents

If you are a Polish resident, you can get a Fiasco registration from WMFH
for a special price. Information addresses:
WWW: http://amiga.com.pl/fiasco/
eMail: silverdr@amiga.com.pl
```

1.345 File List

```
File List

The Fiasco release 2.2 distribution consists of these files: Archive
Fiasco_main.lha:

Fiasco_2.2/ARexx.info
Fiasco_2.2/ARexx/age.frx
Fiasco_2.2/ARexx/age.frx.info
Fiasco_2.2/ARexx/arexxprint.rexx
Fiasco_2.2/ARexx/arexxprint.rexx.info
Fiasco_2.2/ARexx/cap.frx
Fiasco_2.2/ARexx/cap.frx.info
Fiasco_2.2/ARexx/converttolistview.frx
Fiasco_2.2/ARexx/converttolistview.frx.info
Fiasco_2.2/ARexx/dbstructure.frx
Fiasco_2.2/ARexx/dbstructure.frx.info
Fiasco_2.2/ARexx/dummy.frx
Fiasco_2.2/ARexx/dummy.frx.info
Fiasco_2.2/ARexx/graphprint.frx
Fiasco_2.2/ARexx/graphprint.frx.info
```

```
Fiasco_2.2/ARexx/importcolumn.frx
Fiasco_2.2/ARexx/importcolumn.frx.info
Fiasco_2.2/ARexx/print.frx
Fiasco_2.2/ARexx/print.frx.info
Fiasco_2.2/ARexx/requestdt.frx
Fiasco_2.2/ARexx/requestdt.frx.info
Fiasco_2.2/ARexx/unlockgui.frx
Fiasco_2.2/ARexx/unlockgui.frx.info
Fiasco_2.2/Catalogs/Dansk/Fiasco.catalog
Fiasco_2.2/Catalogs/deutsch/fiasco.catalog
Fiasco_2.2/Catalogs/español/fiasco.catalog
Fiasco_2.2/Catalogs/Italiano/fiasco.catalog
Fiasco_2.2/Catalogs/svenska/fiasco.catalog
Fiasco_2.2/Databases.info
Fiasco_2.2/Databases/Addresses2.info
Fiasco_2.2/Databases/Addresses2/Adressen.fdb
Fiasco_2.2/Databases/Addresses2/Adressen.fdb.info
Fiasco_2.2/Databases/Addresses2/Adressen.frec
Fiasco_2.2/Databases/Addresses2/Adressen/Standard.fidx
Fiasco_2.2/Databases/Addresses2/Adressmanager-Konv.rexx
Fiasco_2.2/Databases/Addresses2/Adressmanager-Konv.rexx.info
Fiasco_2.2/Databases/Aminet.info
Fiasco_2.2/Databases/Aminet/Aminet.fdb
Fiasco_2.2/Databases/Aminet/Aminet.fdb.info
Fiasco_2.2/Databases/Aminet/Aminet.frec
Fiasco_2.2/Databases/Aminet/Aminet/Standard.fidx
Fiasco_2.2/Databases/Aminet/copyarc.frx
Fiasco_2.2/Databases/Aminet/extract.frx
Fiasco_2.2/Databases/Aminet/Scancont.frx
Fiasco_2.2/Databases/FamilyTree.info
Fiasco_2.2/Databases/FamilyTree/Families.fdat
Fiasco_2.2/Databases/FamilyTree/Families.fdb
Fiasco_2.2/Databases/FamilyTree/Families.fdb.info
Fiasco_2.2/Databases/FamilyTree/Families.frec
Fiasco_2.2/Databases/FamilyTree/Families/Standard.fidx
Fiasco_2.2/Databases/FamilyTree/Persons.fdb
Fiasco_2.2/Databases/FamilyTree/Persons.fdb.info
Fiasco_2.2/Databases/FamilyTree/Persons.frec
Fiasco_2.2/Databases/FamilyTree/Persons/Standard.fidx
Fiasco_2.2/Databases/GraphDemo.info
Fiasco_2.2/Databases/GraphDemo/Fragments.fdb
Fiasco_2.2/Databases/GraphDemo/Fragments.fdb.info
Fiasco_2.2/Databases/GraphDemo/Fragments.frec
Fiasco_2.2/Databases/GraphDemo/Fragments/Standard.fidx
Fiasco_2.2/Databases/Mailing List Archive.info
Fiasco_2.2/Databases/Mailing List Archive/createhtml.frx
Fiasco_2.2/Databases/Mailing List Archive/CreateHTML.frx.info
Fiasco_2.2/Databases/Mailing List Archive/Example Output.info
Fiasco_2.2/Databases/Mailing List Archive/Example Output/1.html
Fiasco_2.2/Databases/Mailing List Archive/Example Output/2.html
Fiasco_2.2/Databases/Mailing List Archive/Example Output/index.html
Fiasco_2.2/Databases/Mailing List Archive/importmails.frx
Fiasco_2.2/Databases/Mailing List Archive/MLArchive.fdat
Fiasco_2.2/Databases/Mailing List Archive/MLArchive.fdb
Fiasco_2.2/Databases/Mailing List Archive/MLArchive.fdb.info
Fiasco_2.2/Databases/Mailing List Archive/MLArchive.frec
Fiasco_2.2/Databases/Mailing List Archive/MLArchive/Standard.fidx
```

```
Fiasco_2.2/Databases/Mailing List Archive/scandir.frx
Fiasco_2.2/Databases/Mailing List Archive/tpl1.html
Fiasco_2.2/Databases/Mailing List Archive/tpl1.html.info
Fiasco_2.2/Databases/Mailing List Archive/tpl2.html
Fiasco_2.2/Databases/Mailing List Archive/tpl2.html.info
Fiasco_2.2/Databases/Multimedia.info
Fiasco_2.2/Databases/Multimedia/A4000T.iff
Fiasco_2.2/Databases/Multimedia/Amiga.iff
Fiasco_2.2/Databases/Multimedia/gotoxref.frx
Fiasco_2.2/Databases/Multimedia/MMEnc.fdat
Fiasco_2.2/Databases/Multimedia/MMEnc.fdb
Fiasco_2.2/Databases/Multimedia/MMEnc.frec
Fiasco_2.2/Databases/Multimedia/MMEnc/Standard.fidx
Fiasco_2.2/Databases/Organizer.info
Fiasco_2.2/Databases/Organizer/Addresses.fdat
Fiasco_2.2/Databases/Organizer/Addresses.fdb
Fiasco_2.2/Databases/Organizer/Addresses.fdb.info
Fiasco_2.2/Databases/Organizer/Addresses.frec
Fiasco_2.2/Databases/Organizer/Addresses/Standard.fidx
Fiasco_2.2/Databases/Organizer/Appointments.fdat
Fiasco_2.2/Databases/Organizer/Appointments.fdb
Fiasco_2.2/Databases/Organizer/Appointments.fdb.info
Fiasco_2.2/Databases/Organizer/Appointments.frec
Fiasco_2.2/Databases/Organizer/Appointments/Standard.fidx
Fiasco_2.2/Databases/Organizer/checkappointments.frx
Fiasco_2.2/Databases/Organizer/dial.frx
Fiasco_2.2/Databases/Organizer/fax.frx
Fiasco_2.2/Databases/Organizer/Labels.fpr
Fiasco_2.2/Databases/Organizer/Labels.fpr.info
Fiasco_2.2/Databases/Organizer/ListLaTeX.fpr
Fiasco_2.2/Databases/Organizer/ListLaTeX.fpr.info
Fiasco_2.2/Databases/Organizer/mail.frx
Fiasco_2.2/Databases/PD-Disks.info
Fiasco_2.2/Databases/PD-Disks/Disks.fdat
Fiasco_2.2/Databases/PD-Disks/Disks.fdb
Fiasco_2.2/Databases/PD-Disks/Disks.fdb.info
Fiasco_2.2/Databases/PD-Disks/Disks.frec
Fiasco_2.2/Databases/PD-Disks/Disks/Standard.fidx
Fiasco_2.2/Databases/PD-Disks/ReadFish.rexx
Fiasco_2.2/Databases/PD-Disks/ReadFish.rexx.info
Fiasco_2.2/Databases/PictureDatabase.info
Fiasco_2.2/Databases/PictureDatabase/AddPicture.frx
Fiasco_2.2/Databases/PictureDatabase/CreateThumbnail
Fiasco_2.2/Databases/PictureDatabase/DelPicture.frx
Fiasco_2.2/Databases/PictureDatabase/Display.frx
Fiasco_2.2/Databases/PictureDatabase/PicInfo.frx
Fiasco_2.2/Databases/PictureDatabase/Pictures.fdb
Fiasco_2.2/Databases/PictureDatabase/Pictures.fdb.info
Fiasco_2.2/Databases/PictureDatabase/Pictures.frec
Fiasco_2.2/Databases/PictureDatabase/Pictures/Standard.fidx
Fiasco_2.2/Databases/PictureDatabase/ScanDir.frx
Fiasco_2.2/Databases/PictureDatabase/SearchPicture.frx
Fiasco_2.2/Databases/PictureDatabase/StartProg
Fiasco_2.2/Databases/PictureDatabase/TN/TN_1_0
Fiasco_2.2/Databases/PictureDatabase/TN/TN_1_1
Fiasco_2.2/Databases/Videos.info
Fiasco_2.2/Databases/Videos/Movies.fdat
```

```
Fiasco_2.2/Databases/Videos/Movies.fdb
Fiasco_2.2/Databases/Videos/Movies.fdb.info
Fiasco_2.2/Databases/Videos/Movies.frec
Fiasco_2.2/Databases/Videos/Movies/Standard.fidx
Fiasco_2.2/Databases/Videos/Tapes.fdat
Fiasco_2.2/Databases/Videos/Tapes.fdb
Fiasco_2.2/Databases/Videos/Tapes.fdb.info
Fiasco_2.2/Databases/Videos/Tapes.frec
Fiasco_2.2/Databases/Videos/Tapes/Standard.fidx
Fiasco_2.2/Development.info
Fiasco_2.2/Development/Locale.info
Fiasco_2.2/Development/Locale/Fiasco.cd
Fiasco_2.2/Development/Locale/Fiasco.cd.info
Fiasco_2.2/Development/Locale/Fiasco.ct
Fiasco_2.2/Development/Locale/Fiasco.ct.info
Fiasco_2.2/Development/Locale/Locale.readme
Fiasco_2.2/Development/Locale/Locale.readme.info
Fiasco_2.2/Development/Locale/v5_v6_changes.txt
Fiasco_2.2/Development/Locale/v5_v6_changes.txt.info
Fiasco_2.2/Development/Locale/v6_v8_changes.txt
Fiasco_2.2/Development/Locale/v6_v8_changes.txt.info
Fiasco_2.2/Development/WBPath.info
Fiasco_2.2/Development/WBPath/pathtest
Fiasco_2.2/Development/WBPath/pathtest.c
Fiasco_2.2/Development/WBPath/pathtest.c.info
Fiasco_2.2/Development/WBPath/pathtest.info
Fiasco_2.2/Development/WBPath/wbpath.h
Fiasco_2.2/Development/WBPath/wbpath.h.info
Fiasco_2.2/Development/WBPath/wbpath.o
Fiasco_2.2/Development/WBPath/wbpath.o.info
Fiasco_2.2/Documentation.info
Fiasco_2.2/Fiasco
Fiasco_2.2/Fiasco.info
Fiasco_2.2/gadgets/textfield.gadget
Fiasco_2.2/gtlayout.library
Fiasco_2.2/icons/ARexx.info
Fiasco_2.2/icons/ARexxScript.info
Fiasco_2.2/icons/Databases.info
Fiasco_2.2/icons/def_FiascoPrint.info
Fiasco_2.2/icons/Documentation.info
Fiasco_2.2/icons/Drawer.info
Fiasco_2.2/icons/Fiasco.dvi.info
Fiasco_2.2/icons/Fiasco.guide.info
Fiasco_2.2/icons/Fiasco.info
Fiasco_2.2/icons/FiascoProject.info
Fiasco_2.2/icons/XPort.info
Fiasco_2.2/icons/XPortData.info
Fiasco_2.2/Install.info
Fiasco_2.2/Install/Deutsch.info
Fiasco_2.2/Install/English.info
Fiasco_2.2/Install/Install
Fiasco_2.2/Libs/MC68020.info
Fiasco_2.2/Libs/MC68020/gtlayout.library
Fiasco_2.2/ReadMe.txt
Fiasco_2.2/ReadMe.txt.info
Fiasco_2.2/RegForm.txt
Fiasco_2.2/RegForm.txt.info
```

```
Fiasco_2.2/XPort.info
Fiasco_2.2/XPort/mpearls_III_findpeals.fxp
Fiasco_2.2/XPort/mpearls_III_findpeals.fxp.info
Fiasco_2.2/XPort/RFF.fxp
Fiasco_2.2/XPort/RFF.fxp.info
Fiasco_2.2/XPort/StdTwist.fxp
Fiasco_2.2/XPort/StdTwist.fxp.info


 Archive Fiasco_doc_eng.lha:

Fiasco_2.2/Documentation/English/Fiasco.dvi
Fiasco_2.2/Documentation/English/Fiasco.dvi.info
Fiasco_2.2/Documentation/English/Fiasco.guide
Fiasco_2.2/Documentation/English/Fiasco.guide.info
Fiasco_2.2/Documentation/English.info


 Archive Fiasco_doc_deu.lha:
Fiasco_2.2/Documentation/Deutsch/Fiasco.dvi
Fiasco_2.2/Documentation/Deutsch/Fiasco.dvi.info
Fiasco_2.2/Documentation/Deutsch/Fiasco.guide
Fiasco_2.2/Documentation/Deutsch/Fiasco.guide.info
Fiasco_2.2/Documentation/Deutsch.info
```

1.346 Error Codes

```
                Error Codes

This section lists all error codes that may be generated by Fiasco 2.2.
In most cases the error codes are only available to the user by Fiasco's
ARexx port and the variable FIASCO.LASTERROR. In some cases, Fiasco error
requesters also show the error code. The ARexx command
                Fault
                 can be used
to convert the code into a localized error text.


1000 - Unknown command or function: An ARexx command or a formula
   function is unknown for Fiasco. May be caused by misspelling or using
   an obsolete Fiasco version.

1001 - Bad arguments: The arguments to an ARexx command or a formula
   function are not valid.

1002 - Unknown field ID: The field ID specified for an ARexx command or
   used in a formula does not exist in the respective database.

1003 - No record: A record is required and not available.

1004 - Unknown project: The specified project is currently not loaded by
   Fiasco.
```

1005 - Wrong mode: A command requires that Fiasco is in a specific mode that is currently not active.

1006 - Wrong field type: An ARexx command or a formula tried to access a field that does not allow such an access because of its type.

1007 - Search failed: The search function did not found a match.

1008 - Project already active: The project is already active.

1009 - GUI not locked: UnlockGUI tried to unlock a GUI that was not locked.

1010 - Could not activate field: ActivateField could not activate a field.

1011 - Not listview: An ARexx command or a formula tried to access a field as a listview field, although it is not a listview field.

1012 - Index out of range: The index/entry number for a listview field does not exist.

1013 - Unknown index: Index file not found.

1014 - Could not activate index: Index file could not be activated.

1015 - Unknown search info: An ARexx command tried to used a search info that does not exist.

1016 - ARexx server not running: The RexxMast process does not exist.

1017 - ARexx error: Universal ARexx error.

1018 - Unknown field type: A field type was specified that does not exist. Could be caused by misspelling or by using an obsolete Fiasco version.

1019 - Field already exists: CreateField tried to create a field with the ID of an already existant field.

1101 - Unmatched parenteses: An opening brace was not matched by a closing brace in a formula.

1102 - Missing operand: You specified in a formula an operator but left out a operand that is required.

1103 - Illegal operand: A operator cannot use the operand associated to it.

1104 - Syntax error: General syntax error.

1105 - Expression expected: An expression was expected but did not exist.

1106 - If without else: The ? operator was found without a matching :.

1107 - Else without if: A : was found without a matching ?.

```
1200 - Math error: General math error.

1201 - Underflow: A number is too small to be used.

1202 - Overflow: A number is too large to be used.

1203 - Division by zero: Numbers may not be divided by zero.

1204 - Not a valid number: A number is not valid.

1205 - Not comparable: Numbers cannot be compared.

1206 - Domain: The argument of a mathematic function is not valid.

1207 - Out of range: A number is out of its range.

1301 - Unknown object type: Object type for GetAttr or SetAttr is not
   known. Could be caused by misspelling or an obsolete Fiasco version.

1302 - Object name missing: An object for GetAttr or SetAttr requires an
   object name and none has been specified.

1303 - Unknown object name: The name for an object is not known. May be
   caused by misspelling.

1304 - Unknown attribute: An attribut is not known by Fiasco. May be
   caused by misspelling or an obsolete Fiasco version.

Fiasco can also generate Amiga DOS error codes. These are the most
common ones:


103 - No free store: Not enough memory for a operation. Try to free
   some memory and try again or expand your system.

115 - Bad number: A number was requested but Fiasco got something
   different.

116 - Required argument missing: Fiasco requires more arguments than
   currently specified.

117 - Keyword needs argument: A keyword has been specified without an
   argument.

118 - Too many arguments: You specified more arguments than allowed.

119 - Unmatched quotes: A string is not terminated by ".

304 - Break: The user cancelled an operation.
```

1.347 Relation Checklist

Relation Checklist

· create key field "there". Optionally activate "unique key".

· create real field "there". In case of string, extern or datatypes, remember "max chars".

· save project.

· create key field "here". Must be the same type as "there".

· create real field "here". Must be the same type as "there". In the case of string, extern or datatypes, "max chars" must be equal.

· save project.

· open relation requester for real field "here".

· select key "here"

· select relation file

· select key and real field "there". If the correct field is not displayed, check type and in case of string, extern or datatypes max chars.

· select Ok

1.348 Technical Information

Technical Information

This chapter contains information about the internal working of some Fiasco functions. This may be useful for a better understanding of these functions.

Implementation of Clipboard Support

1.349 Implementation of the Clipboard support

Implementation of the Clipboard support

The menuitems
Cut Record
,
Copy Record
and
Paste Record
use the clipboard
to store data temporarily. The clipboard of the Amiga OS is meant to provide a interface for different programs to share certain types of

data. To make this possible, the clipboard may only contain IFF data.

Fiasco uses unit 0 of the clipboard and stores its data in IFF-FTXT files with a specific format. Each field gets a separate chunk. In this chunk the field content is stored in ASCII format.

The order of the chunks depends on the internal field list of Fiasco. Fiasco also uses this order to find out, which data belongs to which field while pasting the clipboard-contents.

With most other programs, you cannot create such structured IFF-FTXT files. The pasting in other programs is better supported. For example the conclip- program pastes the data correctly, while MultiView displays only the first chunk.

1.350 Bugs

Bugs

If you find any bug in Fiasco, please send a detailed description to me
.
Please include information about your processor, OS version and other configuration. This is best made with the ShowConfig program in the Tools drawer.

These bugs are currently known:

- Weird bug in the print function: On some systems, Fiasco does only print the first record if position 0;0 is occopied by a field. If you move that field one position to the left, printing will work normally.
- The frame of the list window flashes sometimes in a weird way under Kickstart 37.x
- Seems to leave sometimes some memory allocated.
- Produces with asl.library 40.6 and Kickstart 40.70 MungWall hits after closing a filerequester. I think this is a bug of asl or intuition but not of Fiasco.

1.351 To do

To do

Fiasco is of course not perfect, at all. Here is a list of all things, which will be perhaps added at a later point (no guarantee!). If you have

an Idea, send it to
me
!

- Complete rewrite of Fiasco's mask user interface. This could feature a smoth scrolling, more configuration options like a overwrite mode for string fields and the possibility to have several masks for one database.
- Internal multitasking.
- Completely configurable menus.
- A mark by pattern function that does not overwrite the old marks.
- "Packing" of projects: search for unused fields and make used as small as possible.
- Checking, whether a similar record already exists (automatically)
- "Input only once" field attribute
- Option to save a Fiasco project in one file (as it was with Fiasco 1.x)
- Option to protect databases with passwords. Perhaps with several "user levels", i.e. read access and write access.
- Datatypes fields, which load their contents after the user did nothing for a short time period. This would make browsing through records faster.
- "Cache" for datatypes fields data.
- Fiasco should display the number of the active record and the number of all records (as in the service window) in the window title bar.
- There should be some control (e.g. from ARexx) whether fields are disabled or not.
- The order how string fields are activated should be controlable.
- Better project handling.
- Change of index handling.
- Undo function.
- More GUI support ARexx commands, such as a generic listview requester.
- Possibility to cancel mask operations from an ARexx script or similar.
- Fiasco's ARexx port and perhaps also other functions should have another method to identify records than by its record number.
- More flexibile field copying function.

Furthermore, I am working on a pOS port of Fiasco.

1.352 Credits

Credits

I would like to thank the following people
who have helped me in some way to create Fiasco:

```
Reinhard Katzmann    Betatesting
Giuseppe Sacco     Betatesting
Ulrich Scholz      Betatesting
Carsten Klein      Betatesting
Curtis Stanton     Revision of my English docs
Dirk Hartstein     Betatesting
Gregor B. Rosenauer    Betatesting
Lutz Kalkof      Betatesting and advertising ;-)
Giuseppe Chillemi    Betatesting and Italian catalog
Claudio Mazzuco    Italian catalog
Thomas Schwarz     Betatesting
Michael A. Krehan    Betatesting
Martin Sahlén      Swedish catalog
Per Torp     Danish catalog
Ralf Terber      Betatesting
Javier Romero      Spanish catalog
Olaf Barthel     gtlayout.library
Mark Thomas      textfield.gadget
Ralph Babel      WBPath
Jay Miner      Inventing the Amiga
Commodore      Producing the Amiga
ESCOM        Well, Amiga Developer CD?
VisCORP      Having wanted to buy the Amiga
Gateway 2000     Let's see...
proDAD       Creating pOS
```

Thanks of course also to all registered users!

1.353 Support for Fiasco

Support for Fiasco

Updates

New versions of Fiasco will be uploaded to Aminet first. They should be
available in the directory biz/dbase. For more information about Aminet,
send an eMail with the word Help in the mail body to
Aminet-Server@aminet.org.

Mailing List

To subscribe to the Fiasco mailing list, send a mail with the text subscribe fiasco in the mail body to majordomo@in-tec.de. To write messages into the list, write to fiasco@amigaworld.com.

If you have problems while subscribing to the mailing list, you may try to subscribe to the backup mailing list. This mailing list will be used in the case when the normal mailing list is be down. Send a mail with the subject subscribe fiasco to listserv@wanderer.gun.de . Please also read the introduction mail from the mailing list.

Word Wide Web

There is also an internet homepage for Fiasco. You may obtain the latest news about Fiasco there. The URL is http://www.amigaworld.com/support/fiasco/.

Author

The author of Fiasco can be reached at the following addresses:
Nils Bandener
Dekanatsgasse 4
D-34369 Hofgeismar
Germany

Internet: nils@dinoex.sub.org
Some AmigaGuide versions apparently handle the At-Sign different from others. Thus, here is the address once again, if the address above should be not readable:

Internet: nils@dinoex.sub.org

1.354 Index

Index

```
ActivateDBWindow
  ActivateDBWindow

ActivateField
  ActivateField

ActiveIndex
  ActiveIndex

ActiveRecord
  ActiveRecord

activerecord()
  activerecord()

add element menuitem
  Element/Add...

add field menuitem
  Field/Add Field...

Add gadget
  Add

add record menuitem
  Record/Add Record

AddLVFieldEntry
  AddLVFieldEntry

AddRecord
  AddRecord

advanced usage
  Advanced Usage of Fiasco Databases

alternative format
  Converting Fields

Amiga DOS
  Wildcards

AmigaGuide
  Fiasco's Graphic User Interface

AmigaGuide
  Requirements

annotations
  Project Options Requester

appicon
  Project/Hide

ARexx
  The ARexx Port
```

```
ARexx debug menuitem
  Control/ARexx-Debug

ARexx/accessing
  Accessing the Port

ARexx/database shutdown
  Project Options Requester

ARexx/database shutdown
  User Interface Settings Requester

ARexx/database startup
  Project Options Requester

ARexx/database startup
  User Interface Settings Requester

ARexx/debugging
  Debugging ARexx Scripts

ARexx/Fiasco 2.1
  The ARexx Port

ARexx/LastError
  Results of Commands

ARexx/port
  Accessing the Port

ARexx/print
  Printing with ARexx

ARexx/program shutdown
  User Interface Settings Requester

ARexx/program startup
  User Interface Settings Requester

ARexx/quotes
  Arguments of Commands

ARexx/rc
  Results of Commands

ARexx/Result
  Results of Commands

ARexx/searching with
  Searching with ARexx

ARexx/stem variables
  Results of Commands

ARexx/success
  Results of Commands
```

```
count requester
  Count Requester

counting matches
  Count

CountRecords
  CountRecords

create group menuitem
  Field/Create Group

create icons
  User Interface Settings Requester

CreateField
  CreateField

CreateThumbnail
  Picture Database

current entry operator
  Operators

current()
  Operators

currentdate()
  currentdate()

currenttime()
  currenttime()

cursor
  The Mask Window

cursor keys
  The Mask Window

cut record menuitem
  Record/Cut Record

CutRecord
  CutRecord

cycle
  Cycle Fieldtype

cycle trough fields with enter
  User Interface Settings Requester

data structure
  Basic elements of a Database

database settings menuitem
  Settings/Databases...
```

```
database settings requester
  Database Settings Requester

database shutdown script
  Project Options Requester

database startup script
  Project Options Requester

datatypes
  Datatypes Fieldtype

datatypes/animation
  Datatypes Fieldtype

datatypes/immediate playing
  Datatypes Fieldtype

datatypes/scrolling
  Datatypes Fieldtype

datatypes/sound
  Datatypes Fieldtype

datatypes/speeding up record changes
  Datatypes Fieldtype

date
  Date Fieldtype

datediff()
  datediff()

day()
  day()

debugging of ARexx scripts
  Debugging ARexx Scripts

delete all records menuitem
  Record/Delete all Records

Delete gadget
  Delete

delete record menuitem
  Record/Delete Record

DeleteAllRecords
  DeleteAllRecords

DeleteConstant
  DeleteConstant

DeleteLVFieldEntry
  DeleteLVFieldEntry
```

```
edit index requester
  New/Edit Index Requester

edit named field menuitem
  Field/Edit named Field...

edit relation menuitem
  Field/Edit Relation...

edit usermenu menuitem
  Settings/User Menu...

edit usermenu requester
  User Menu Requester

editing the print mask
  The Print Mask

editor menu item
  Settings/External Programs and Paths...

eepic
  GraphPrint.rexx

EHB
  Picture Database

element type submenu
  Element/Element Type

enter
  User Interface Settings Requester

erase menuitem
  Project/Erase

erase menuitem in print window
  Project/Erase

error codes
  Error Codes

escape sequences in im-export
  How to Specify Special Characters

escape wildcards
  Wildcards

exit menuitem
  Project/Exit

Export
  Export

export
  Import and Export
```

```
export menuitem
  Project/Export...

export/requester
  Export Requester

export/required marking chars
  Structure of Import/Export files

export/structure of files
  Structure of Import/Export files

extern
  Extern Fieldtype

external data
  Import and Export

external programs and paths
  External Programs and Paths Requester

external programs and paths menu item
  Settings/External Programs and Paths...

factor
  Blurred Search

false
  Boolean Fieldtype

Fault
  Fault

Fiasco 1.x
  Indices Requester

FIASCO.LASTERROR
  Results of Commands

fiasco_port
  Accessing the Port

field requester
  Field Requester

field type cycle gadget
  Mask Mode

fields
  Fields

fields/ARexx
  Standard Attributes

fields/attributes
  Field Requester
```

```
fields/integer
  Integer Fieldtype

fields/listview
  Listview Fieldtype

fields/picker button
  Standard Attributes

fields/predefined values
  Standard Attributes

fields/programming
  Standard Attributes

fields/selecting multiple
  The Mask Window

fields/shifting
  Field Requester

fields/slider
  Slider Fieldtype

fields/squeezing
  Field Requester

fields/string
  String Fieldtype

fields/tab cycling
  The Mask Window

fields/text
  Text Fieldtype

fields/time
  Time Fieldtype

fields/validity of attributes
  Field Requester

fields/var string
  Var String Fieldtype

fields/virtual
  Standard Attributes

fields/virtual
  Virtual Fields

fields/width
  Standard Attributes

fieldtype menuitem
  Field/Fieldtype
```

```
formatdate()
  formatdate()

formatstring
  Slider Fieldtype

formattime()
  formattime()

formula requester
  Formula Requester

formulas
  Formulas

formulas/boolean values
  Constant Values

formulas/constants
  Constants

formulas/fields
  Fields

formulas/functions
  Functions

formulas/index numbers
  Fields

formulas/listview fields
  Fields

formulas/numbers
  Constant Values

formulas/operators
  Operators

formulas/returned values
  Constant Values

formulas/strings
  Constant Values

formulas/value conversions
  Constant Values

formulas/values
  Constant Values

function keys
  User Menu Requester

functions
  Functions
```

```
functions
  Function Reference

functions menu item
  Database/Functions...

functions requester
  Functions Requester

gadgets
  Mask

gadtools.library
  Mask

get from list menuitem
  Project/Get from List

get from mask menuitem
  Project/Get from Mask

GetAttr
  GetAttr

GetConstant
  GetConstant

GetField
  GetField

GetRecordMark
  GetRecordMark

goto record menuitem
  Record/Goto...

goto record requester
  Goto Requester

GraphPrint.rexx
  GraphPrint.rexx

grouping groups
  Groups

grouping groups
  Field/Create Group

groups
  Groups

groups/grouping
  Groups

groups/grouping groups
  Field/Create Group
```

```
MarkRecord
  MarkRecord

marks
  Using Marks

mask
  Mask

mask mode
  Mask Mode

mask mode menuitem
  Control/Mask Mode

mask/stretching
  Mask Stretching

memory pools
  Requirements

MenuControl
  MenuControl

menuhelp
  Fiasco's Graphic User Interface

minute()
  minute()

month()
  month()

mouse
  The Mask Window

MoveRecord
  MoveRecord

multimedia database
  Multimedia Database

name of author
  Project Options Requester

narrator.device
  User Interface Settings Requester

New
  New

new index requester
  New/Edit Index Requester

new look proportional gadgets
  Display Settings Requester
```

```
new menuitem
  Project/New

NewSearchInfo
  NewSearchInfo

next act. index menu item
  Database/Next active Index

next record menuitem
  Record/Next

No Index
  Record/Delete Record

numrecords()
  numrecords()

online help
  Requirements

online-help
  About this Document

Open
  Open

open menuitem
  Project/Open...

open menuitem in print window
  Project/Open...

open new menuitem
  Project/Open new...

OpenListWindow
  OpenListWindow

OpenServiceWindow
  OpenServiceWindow

operands
  Operators

operators
  Operators

options menu item
  Database/Options...

options menuitem in print window
  Project/Options...

options requester
  Project Options Requester
```

```
overwrite old project
  Import

paste record menuitem
  Record/Paste Record

PasteRecord
  PasteRecord

paths menu item
  Settings/External Programs and Paths...

paths requester
  External Programs and Paths Requester

picture database
  Picture Database

pools
  Requirements

popup gadget requester
  Popup Gadget Requester

pOS
  To do

prev. act. index menu item
  Database/Previous active Index

previous record menuitem
  Record/Previous

print
  Printing a Database

print mask files
  Print Mask Files

print menuitem
  Project/Print...

print menuitem in print window
  Project/Print

print/ARexx
  Printing with ARexx

print/clipping
  The Print Mask

print/editing the print mask
  The Print Mask

print/element requester
  Print Element Requester
```

```
records
  Records

records/activating
  The List Window

records/cloning
  Working with Records

records/copy
  Working with Records

records/creating
  Working with Records

records/cut
  Working with Records

records/delete
  Working with Records

records/deleted
  Database/Reorganize...

records/list
  The List Window

records/numbers
  Goto Requester

records/paste
  Working with Records

records/recover
  Record/Delete Record

recover deleted records
  Record/Delete Record

relation requester
  Relation Requester

relation/remote
  Relations

relations
  Relations

relations/1:L
  Relation Types

relations/1:N
  Relation Types

relations/here
  Relations
```

```
relations/local
  Relations

relations/N:L
  Relation Types

relations/N:Sum
  Relation Types

relations/only read
  Relation Types

relations/there
  Relations

relations/types
  Relation Types

relations/updating
  Database/Reload Relations

relations/updating
  Database Settings Requester

relations/writing
  Database Settings Requester

reload relations menu item
  Database/Reload Relations

remote data
  Relations

remove element menuitem
  Element/Remove

remove field menuitem
  Field/Remove Field

remove relation menuitem
  Field/Remove Relation

reorganize menu item
  Database/Reorganize...

replace
  Replace

replace menuitem
  Compare/Replace...

replace requester
  Replace Requester

request database
  User Interface Settings Requester
```

```
RequestChoice
  RequestChoice

RequestField
  RequestField

RequestFile
  RequestFile

RequestNumber
  RequestNumber

RequestString
  RequestString

ResetStatus
  ResetStatus

resolve group menuitem
  Field/Resolve Group

Result
  Results of Commands

reveal project menuitem
  Project/Reveal...

reveal project requester
  Reveal Project Requester

RevealProject
  RevealProject

right()
  right()

round()
  round()

Save
  Save

save as menuitem
  Project/Save As...

save as menuitem in print window
  Project/Save as...

save menuitem
  Project/Save

save menuitem in print window
  Project/Save

save settings as menuitem
  Settings/Save Settings as...
```

```
save settings menuitem
  Settings/Save Settings

SaveAs
  SaveAs

SaveSettings
  SaveSettings

saving disk space
  Relations

screenmode requester
  Requirements

screenmode requester
  Display Settings Requester

search
  Searching in a Database

search requester
  Search Requester

search/ARexx
  Searching with ARexx

search/by fields
  Searching by Fields

search/by formulas
  Searching by Formulas

search/joker
  Wildcards

search/matching records
  Searching by Fields

search/mode
  Searching in a Database

search/patterns
  Patterns

search/requester
  Searching in a Database

search/search info
  Search Information

search/wildcards
  Wildcards

search/wildcards for numbers
  Wildcards for numbers
```

```
second()
  second()

security requesters
  Working with Records

security requesters
  User Interface Settings Requester

select index requester
  Indices Requester

service window
  Progress

service window
  SetStatus

service window
  The Service Window

service window menuitem
  Control/Service Window

service window/fixed position
  User Interface Settings Requester

service window/M
  Using Marks

service window/marks
  Using Marks

service window/open on startup
  User Interface Settings Requester

SetAttr
  SetAttr

SetConstant
  SetConstant

SetField
  SetField

SetMode
  SetMode

SetSearchField
  SetSearchField

SetStatus
  SetStatus

shareware
  Shareware
```

```
UnlockGUI
  UnlockGUI

unmark all records menuitem
  Record/Unmark all Records

unmark record menuitem
  Record/Unmark Record

update relations
  Database Settings Requester

updating databases
  Updating databases with Im/Export

use * as pattern
  Database Settings Requester

user interface settings
  Settings/User Interface...

user interface settings
  User Interface Settings Requester

user-defined functions
  Functions

usermenu requester
  User Menu Requester

using indices
  Using Indices

var string
  Var String Fieldtype

version()
  version()

virtual fields
  Standard Attributes

virtual fields
  Virtual Fields

wait clock
  LockGUI

WBPath
  Legal Things

workbench
  Project/Hide

write relations
  Database Settings Requester
```

```
XFH
  Requirements

year()
  year()
```