MANUAL

CONTENTS

AMOUNTS

PURPOSE

A general rule for saving and presenting amounts.

DESCRIPTION

Amount are always saved **without** decimals. For example :US $ 1,000,999.00 is saved as 100099900.
Because of the multi-currency possibilities in SERA the decimals are calculated while printing or showing on the screen. Get from file \SERA\DAT00\VALUTA the right currency and the number of decimals belonging to it. Foreign currencies are stored to a free variable name. The company Number of decimals is already loaded in the variable called ‘V’.
Assume that 100099900 is stored in a field called AMOUNT, then print it as :

```
? STR(AMOUNT/(10**V),13,V)
```

the output will be :

```
   1000999.00
```

where 13 is the total length including trailing spaces.

NOTE: When V is 2, (10**V) means 10 to the power 2, which is 100. When no decimals are required, the value of V is zero. Then (10**V) means 10 to the power zero, which is according mathematic rules always 1. Then a amount for example of 20312 Italian Lire’s is printed as:

```
? STR(AMOUNT/(10**V),13,V)
```

Be sure that an entered amount of 4510,89 US $ is stored as follows

```
REPLACE <FIELD> WITH <ENTRY> * (10**V).
```

Do this even when v is zero. The same program can be used for other currencies.

An amount or number can be printed with a thousand divider. Make sure that there is a \sera\paramet\thousand.par that contains a dot or a comma.

GLOBAL VARIABLES

CV	Company Valuta (=Currency))
V	Valuta decimals of the local currency

COMPILING RULES

PURPOSE

To provide a standard building of SERA modules and an explanation how to build them.

DESCRIPTION

All standard *.PRG, *.CLP, *.BAT and *.LNK files are kept in the \SOURCES directory.

To create an executable there are 3 steps :

1.create a source file	:	make *.PRG files.
2.compile with C.EXE	:	make *.OBJ files.
3.link with BLINKER.EXE	:	make *.EXE files.

CLP FILES

Most of the PRG's are calling other programs or they are using function that are stored in libraries. In most cases a prg is calling so many other programs. C.EXE detects automatically all the sub-programs and functions involved. In many cases the load of programs is too heavy. Therefore the programs are divided over a number of CLP files.

example **SERA0A.CLP**

```
SERA0310
SERA0311
SERA0320
SERA0321
SERA0330
etc. ......
```

LNK FILES

LNK files are defining the structure of CLP-files and libraries like

Example **SERA2.LNK**

```
OUTPUT SERA2.EXE
FILE SERA2.OBJ
@SERASUB1.LNK
BEGINAREA
  FILE SERA2A.OBJ
  FILE SERA2B.OBJ
  FILE SERA2C.OBJ
  FILE SERA2D.OBJ
  FILE SERA2E.OBJ
  @SERASUB2.LNK
ENDAREA
@SERASUB3.LNK
```

WARNING! Don't change the linking parameters because they are tested very well. If memory problems are occurring you have to split your application instead of changing them.

BATCH FILES

They will do the job for you while they are containing all instructions :

example **SRA2.BAT**

```
C @SERA2.CLP>ERRA2
C @SERA2A.CLP>ERRA2A
C @SERA2B.CLP>ERRA2B
C @SERA2C.CLP>ERRA2C
C @SERA2D.CLP>ERRA2D
C @SERA2E.CLP>ERRA2E
BLINKER @SERA2.LNK>LNKSERA2
```

The result is SERA2.EXE

NOTE: the batch file is called SRA2.BAT and the exe file is calles SERA2.EXE. This naming convention has been chosen to avoid starting the executable instead of the batch file.

SERATOT.BAT

can be used to build all the SERA-programs. It contains all SR*. and UTL*. BAT files. The executables are copied from the \SOURCES directory to the \SERA directory. The object files are deleted after linking them into an executable.

LARGE PROGRAMS

Some programs like A301, enter journals, are very large and may cause memory problems in the regular modules. Then it is better to create a separate SERA3010.EXE
The SERA MENU will always look for a SERA3010.EXE, SERB8610.EXE. If found, this program will be launched. Otherwise SERA3.EXE or SERB8.EXE will be launched.

The batchfiles to create a separate program like SERA3010.EXE are called B_A301, B_B861, B_C304 etc.

SERATOT.BAT is calling beside the regular modules like SRA3.BAT also some modules like B_A301.BAT

NOTE: there is a B_<PRG>.BAT available for most of the programs, but only a few are used by SERATOT.BAT
Create a single program only if there are memory problems. Don't forget to remove or RENEW it when there is a new release.

DATES

PURPOSE

Providing a standard for date storage and presentation.

DESCRIPTION

Dates are stored in the syntax YYYYMMDD as a character field. Several Sera functions are controlling calculations, presentations and user input. In most tables there is also a related <DATE>_ field that is using the date() format

EXAMPLE

```
DO WHILE LABEL = 4
 DAT = SPACE(8)
 @ 10,00 SAY ‘ENTER DATE ‘
 DAT = GETDATE(10,20,DAT)       // GET USER INPUT
 IF LASTKEY() = 27
  LABEL = LABEL - 1
 ELSE
  IF DATEOK(DAT)
   @ 10,20 SAY SHOWDATE(DAT1,3) // SHOW USER INPUT
   LABEL = LABEL + 1
  ELSE
   @ 24,30 SAYMESS ‘ENTER AS ’ + YYMMDD
  ENDIF
 ENDIF
ENDDO
```

SEE ALSO

Functions WEEKDIFF, D2W,C2D, D2C, C2B, DATEOK, PERIOK, WEEKOK, GETDATE, SHOWDATE, P13, D13

DIRECTORY STRUCTURES

PURPOSE

explanation of the directories as used by the SERA program.

DESCRIPTION

\SERA

executables, and some DLL files

\SOURCES

*.PRG, *.CLP, *.BAT and *.LNK files and a number of libraries that are called during linking.

\SERA\DATSTART

*.DBF and *.DBT files with the latest structure. Some of the files are containing data like PRGLIST.DBF that contains the numbers and description of the Sera programs.

MS-SQLSERVER

The actual data of Sera are stored in a SQL database.
The program \SERA\CONVERS.EXE checks the database against the latest structure in \SERA\DATSTART. New tables are added and changed tables are converted.

\SERA\PARAMET

parameter files are located in this directory. Several programs are checking these files during the execution of the program. Layouts for variable forms are also stored in this directory.
A very important file called SQLSERV.PAR contains the login parameters for the database.

\SERA\<USER>

a directory per user is created as soon as a user is added to the user list

SEE ALSO

The manual SERVERINSTALLATION for more information on the database.

FUNCTION KEYS

PURPOSE

a general rule for defining and using function keys in sources.

DESCRIPTION

A user of the SERA program can define his own user interface function keys. The settings are kept in the file \SERA\DAT01\USERS. As soon as an executable is started the key settings are loaded into the memory as variables:

FKA01.. FKA10	The description of the key like F1,F2,..,F10
FKN01.. FKN10	The number of the key like 1,2,..,10
FKV01.. FKV10	The key board value of the key like 28,-2,..,-9

Often used function keys are :

FKN07	- ALL
FKN08	- ONE
FKN10	- BACK / END /ESCAPE

Example:

```
@ 24,00 SAY ‘<‘+FKA10+’> = BACK’
SET FUNC FKN10 TO CHR(27)
IF INKEY() = FKV10
```

SEE ALSO

PROGRAM STRUCTURES

PROGRAM ADDITIONS

PURPOSE

a procedure to describe the additional work for programming.

DESCRIPTION

Before a program can be taken into SERA, several steps are required:

FIILELIST

In a database called \SERA\DATSTART\FILELIST.DBF are the names, paths and index structures of the databases kept. This file has to be updated with new created databases. The installation or conversion program will take care of installation of the new databases.

WARNING ! Never change the type of an existing field of already installed databases.

DATSTART

The empty new or changed databases has to be copied to the directory \SERA\DATSTART. Only in very special occasions the database is filled with records.

ADDITIONAL FIELD INFORMATION

Extra field information is stored in a database called \SERA\DATSTART\FCOMMENT.
Util5400 is the tool to fill the database with the field names.
After running this tool the new records have to be completed. The additional information is used in the summary of record layouts.

RELATIONS BETWEEN TABLES

The relation ships between tables are stored in \SERA\DATSTART\SQLLINK.

PROGRAM LISTS AND USER MENU’S

The new programs have to be added to \SERA\DATSTART\PRGLIST and to \SERA \DATSTART\USERMENU.

PROGRAM STRUCTURES

PURPOSE

general rules for building Sera styled programs.

DESCRIPTION

To ensure a standard way of programming and a standard user interface, all Sera programs have to meet a set of standard rules.

COMMENTS

Parts of a program that are difficult to understand have to contain information for other programmers. Information like

explanation of calculations
references to other programs
warnings

DON'T comments in the program where it's not required to understand the program like

```
* ask user to enter data
NAME = SPACE(30)
@ 10,20 GET NAME
```

In this case the program itself is clear enough.

HEADER FILE

A header file is called on top of every program by the command :

```
#INCLUDE 'SERA.CH'
```

The header files are containing instructions for the compiler to redirect regular command and functions to other or user defined commands and functions. The programmer doesn't have to worry about it.

SCREEN LAYOUT

A screen is always build in the program itself. Don't use *.FMT files or other external sources. Follow existing programs.

example

```
*BEGIN OF PROGRAM
#INCLUDE 'SERA.CH'
DO PROCTITL      && CLEAR SCREEN AND FILL ROW 0 AND ROW 1
@ 03,00 SAY 'Part number '
@ 04,00 SAY 'Description '
....
....
@ 24,00 SAY '<'+FKA10+'> = BACK'
```

SEMI COLONS ':'

Semi colons are not allowed. The input area's are already high lighted, so there is no need for it. Semi colons are also causing a problem in translating texts into another language:

example

```
@ 10,01 SAYBOLD S('NAME  :') // NOT CORRECT
@ 10,01 SAYBOLD S('NAME')    // CORRECT
```

TEXT STRINGS / TRANSLATIONS

The basic programming language is English. Put a text in the function S() like S('This is an English text')
The function S() collects the translation from the SCRTEXT table according the user language.

QUESTIONS

Avoid as much as possible the usage of question marks. The user knows already that he has to enter something into the system. Never use additions like (Y/N) , y/n.
Never use a space before a question mark, in case one is really required.

```
@ 09,02 SAY 'NUMBER '          // No question marks
@ 11,02 SAY 'Name '
@ 12,02 SAY 'Address '
```

A special question is put at the end of a lot of programs. Only one syntax is allowed:

CONTINUE?   **/NOTE: No Y/N addition, no space !**

UPPER AND LOWER CHARACTERS

Use upper characters only in case information is really important. Use lower characters in all other occasions.

ABBREVIATIONS

Avoid abbreviations as long as there is enough space.

LABELS

Every step in a program is defined by a so called label. A variable like LABEL or STEP etc. is filled with a start value. This value is changed during every step in the program. The new value is representing the next step in the program.

Example

```
LABEL = 1
DO WHILE .T.

 DO WHILE LABEL = 1
  CUST = SPACE(10)
  @ 02,00 SAY ‘CUSTOMER NUMBER ‘
  @ 02,30 GET CUST
  READ
  IF LASTKEY() = 27
   RETURN
  ELSE
   ..
   LABEL = LABEL + 1
  ENDIF
 ENDDO

 DO WHILE LABEL = 2
  NAME = SPACE(30)
  @ 03,00 SAY ‘CUSTOMER NAME  ‘
  @ 03,30 GET NAME
  READ
  IF LASTKEY() = 27
   @ 03,30 SAY SPACE(30)
   LABEL = LABEL - 1
  ELSE
   ..
   LABEL = LABEL + 1
  ENDIF
 ENDDO
 ..

ENDDO
```

MARGINS

Always start with a margin of 0 while writing a program. Increase the margin with only 1 position after a DO WHILE, IF or DO CASE. Decrease the margin with 1 position at a ENDDO, ENDIF or ENDCASE.

SCREEN POSITION

Screen positions must always have the same syntax.

Example

```
@ 03,00
@ 05,40
@ 10,20
@ 04,10
```

instead of

```
@ 03,0
@ 5,40
@10,20
@4,10
```

VARIABLES

The usage of variables is restricted. Watch carefully the predefined ones in SERA-TEST.PRG.
Some variables have a long history. There were times were the number of variables was limited. Therefore names like A6 , A2 were reused. Today's variables should be chosen as meaningful as possible like:

NNUMBER
CNAME
ARRAY

PARAMETER FILES / CUSTOMIZING PROGRAMS

To keep the programs maintainable in the future, it is NOT allowed to make customized versions of existing programs. Customizing facilities are offered by the usage of parameters:

example

```
IF FILE('\SERA\PARAMET\GRN.PAR') .OR. PAINSP = 'y'

 * Customer wants a printed note of all goods receipts or (default) the
 * received parts have to be inspected, for which purpose a note is
 * printed.

 DO GRNPRINT

ENDIF
```

Note : be sure that the parameter is documented in the parameter chapter of the user manual.

In very exceptional cases and ONLY with approval of Sera Europe, the following example is applicable in customizing programs:

example

```
IF ‘INA’$NAM              // NAM IS A PUBLIC VARIABLE CONTAINING THE NAME
                                  // OF THE COMPANY
 SPEC_FUNC()
ELSE
 STD_FUNC()
ENDIF
```

PRINTING / SCREEN OUTPUT

```
N2 = 66                      //       PAGE LENGTH COLLECTED FROM PRINTE/R
N3 = 0                                //       LINE COUNT
N4 = 0                                //       PAGE COUNT
I = 0                                 //       VARIABLE FOR INKEY() VALUE
DO SERAPRIN                  //       INITIALIZE THE PRINTER OR THE SCREEN
N4 = 1                                //       FIRST PAGE
? 'DATE(),TIME() ....                 //       PAGE HEADER.
?
? ‘NUMBER    DESCRIPTION    //       TITEL
?
N3 = 4
SELECT MAIN                                          //       COLLECT DATA TO PRINT
DO WHILE NOT.EOF()
  IF N2 <= N3                               //       NEW PAGE
   EJECT
   N4 = N4 + 1
   ? 'DATE(),TIME() ............
   ?
   ? ‘NUMBER    DESCRIPTION                 ‘
   ?
   N3 = 4
  ENDIF
  ? NUMBER,DESCR
  N3 = N3 + 1
  SET CONSOLE ON                                     //       SHOW PROGRESS
  SET PRINT OFF
  @ 24,30 SAY ‘KEY              PAGE ‘
  @ 24,35 SAY NUMBER
  @ 24,60 SAY STR(N4,4)
  SET CONSOLE OFF
  SET PRINT ON
  I = INKEY()
  IF I = FKV10
   EXIT
  ENDIF
  SKIP
ENDDO
EJECT
SET PRINT TO
SET PRINT OFF
DO SERALP               // END THE PRINT JOB.  DON’T FORGET!!
SET CONSOLE ON
```

PRINTING AND UPDATING

Printing is one of the most critical moments during a program execution. To avoid half update files. a program has to **UPDATE ALL** files and finish all other jobs, **BEFORE PRINTING** is started. See also BEGINTRANS() / ENDTRANS()

COLUMNS

To give a screen a nice look, try to use only a few starting columns for questions, input and output on the screen.

```
CUSTOMER            1000          Enterprise Ltd.
Address                           1000 Lincoln street
City                              Ottawa 23512 ON

Part number         a354521       Electric device a.12

↑                   ↑             ↑
```

The arrows are column positions. Try to use no more than 5 different positions in 1 program.

BOXES AND LINES

Avoid too many boxes and lines.

USER INPUT

After entering data into an input area, the area has to be filled immediately with the validated output.

example

```
DO WHILE LABEL = 4
 DATE = SPACE(6)
 @ 10,00 SAY 'ENTER REFERENCE '
 @ 10,20 GET REF PICTURE '!!!!!!'
 READ
 IF LASTKEY() = 27
  LABEL = LABEL - 1
 ELSE
  IF .NOT. EMPTY(REF)
   @ 10,20 SAY REF
   LABEL = LABEL + 1
  ELSE
   @ 24,30 SAYMESS 'ENTER A REFERENCE'
  ENDIF
 ENDIF
ENDDO
```

Notice that the screen area at row 24 , pos. 30 is cleared after the user input. In case the input is correct, nothing (as it should be) is displayed. In case the input is not correct row 24 is used.

SAY / GET INSTRUCTIONS

Never put a GET instruction in the same line as the SAY instruction. Text strings can have different sizes in different languages and the get positions are not equal anymore after translation.

FUNCTION KEYS

Define the function key values and descriptions just before the GET command.
The function key values have to be switched off directly after the read command. Only the FKN10 key is assumed to be available at all time and switching on/off is therefore not required.

SEE ALSO

FUNCTION KEYS, COLOR SETTINGS, WINDOWS COMPATIBILITY, TRANSLATION, VERSIONS, PROGRAM ADDITIONS, TESTING

SERA FUNCTIONS

PURPOSE

A list of SERA functions and procedures and the syntax how to use them.

DESCRIPTION

Most of the functions and procedures are contained in the CTS1/2/3.LIB
The LIB files are packed in LIBS.ZIP. The unpacked programs are store in V:\SERA\QSLIB.
The LIB files are created by running V:\SERA\QSLIB\MAKELIB.BAT

BEGINTRANS()   / ENDTRANS()

purpose	Make sure that critical processes are stored correctly in the database, even when there is a system or power failure,
description	the functions are marking the begin and end of the process. All updates AFTER BEGINTRANS() is called are stored in the transaction (log) file beside the database. Only after the ENDTRANS() instruction is called, the database is updated. Other users will see the temporarily updates as real, but when there is a failure before the end is reached, all transactions are turned back to the situation as it was at BEGINTRANS()
Note	Make sure the process is as short as possible. So NEVER ask for user input during the transaction.

PROCTITL

purpose	procedure to show the basic screen
description	see program structures
syntax	DO PROCTITL

PROCWIN   (WINWIN)

purpose	procedure to present a windows to look up for a number or a code. Before starting the actual procedure the size, heading and fields have to be defined.
syntax	See example
example	USE \SERA\DAT00\CUST INDEX \SERA\DAT00\CUABC

```
NAME = SPACE(10)
GET NAME PICTURE '!!!!!!!!!!'
READ
SEEK NAME
BOXSIZE = '05|10|20|75'
BOXFIELD = 'CUNUMBER|CUNAME|CUADRESS1|CUADRESS2'
BXTITLE  = 'NUMBER|NAME|ADDRESS|TOWN'
DO PROCWIN
IF LASTKEY() = 13
 CUST = CUNUMBER
ENDIF
```

parameters DO PROCWIN WITH <top , <bott>, <start>, <func>, <inkey>, <prefunc>, <skipfunc>

These parameters are all optional:

<top>	the lowest value of the window to look up. Default is the first record according the index file.
<bott>	the highest value. Default is the last record according the index file.
<start>	the record where to start with. The value has to be in between the <top> and <bott> value. Default is the actual record pointer.
<func>	a user defined function. **NOTE:** If an other function or program is called, where a screen save / restore is required, the saving and restoring has to be done in the calling function and NOT in the function or program that is going to be called. In the windows functions for saving and restoring are also private variables used that are overwritten in a lower program.
<inkey>	the default is .T. In case a .F. is given as parameter. The program assumes that the inkey-handling is done in the user defined function. This can be useful while use a GET statement in this function. The variable PW_INKEY (already available from the main procedure) has to be filled with the lastkey() value or any required other user defined value.
<prefunc>	This function is executed BEFORE an inkey() or get. This option can be used in case f.i. the function keys are depending on the type of line that is displayed. See SERA5115 for an example.
<skipfunc>	The skip function controls special forward or backward scroll instructions. Make sure that the <top> and <bot> values are correct in case this function is used.

PROCPOPU

purpose procedure to present a popup menu. Before starting the actual procedure the size, the items and the start value have to be defined.

syntax See example

example

```
BOXSIZE = ‘10|20|15|40’
BOXITEM = ‘MENU 1|MENU 2|MENU 3|QUIT’
BOXFIRST = 1
BOXCHOICE = 0
DO PROCPOPU
IF LASTKEY() = 13
 CHOICE = BOXCHOICE
ENDIF
```

The top/left position is at @ 10,20.
The procedure will calculate the right lower corner depending on the number and the length of the items

PROCBOX

purpose procedure to draw a box according the Sera standard.

syntax See example

example

```
BOXSIZE = ‘10|10|20|50’
DO PROCBOX
```

provides a box with shadow from @ 10,20 to 12,50

PROCDIR

purpose Collecting a file name from a directory.

syntax BOXDIR = <wildcard>. All standard DOS wild cards are allowed.
.
returns The name of the selected file in the variable BOXDIR

example

```
BOXDIR = ‘\SOURCES\*.CLP’
BOXSIZE = ‘06|40|20|60’
DO PROCDIR
IF LASTKEY() =13
 FILE_NAME = BOXDIR
ENDIF
```

SERAFILL()

purpose function to fill user defined forms

syntax SERAFILL(<txt>,<var>,<n1>,<n2>)

<txt>	user defined text with codes to be replaced by actual values.
<var>	a variable with the actual value to place into the user defined text. **See NOTE at the examples for further details.**
<n1>	number of decimals to be used for amount fields. If N1 has the value 9, then the user can define the format in his formula’s.
<n2>	If 0 : zero values are not printed, else : zero values are printed.

In the user defined <txt> a variable is presented as

-<var>,<len>,<dec>- for numeric fields

<len>	The total length of the field including decimals, sign and dot.

<dec>	The number of decimals. This field makes only sense if the program allows it by the parameter <N1> in SERAFILL().

-<var>,<pos>,<len>**-** for character fields

<pos>	The start position of the field to print. (Normally 1)
<len>	The length of the string to print.

returns The replaced value of the formula.

example 1 Variable form -CUNUMBER- -CUNAME-

Program
```
UNUMBER = CUNUMBER
UNAME = CUNAME
TXT = MEMOREAD('\SERA\DAT02\INVHEAD1.ENG')
SERAFILL(TXT,'CUNUMBER',9,1)
SERAFILL(TXT,'CUNAME,9,1)
PRINTFILE(TXT)
```

example 2 Variable form INVOICE : -IVNUMBER-   AMOUNT :-IVVALUTA,11-

Program
```
VNUMBER = IVNUMBER
CURR = VALUTA->VADECIMALS
VSALES = IVVALUTA / ( 10**CURR)
TXT = MEMOREAD('\SERA\DAT02\INVHEAD1.ENG')
SERAFILL(TXT,'IVNUMBER',9,1)
SERAFILL(TXT,'IVVALUTA,CURR,1)
PRINTFILE(TXT)
```

In this example has the user no influence on the number of decimals. The invoice can have another currency with other decimals.

NOTE
The values must be stored into variables with the same name as the code that the user has to put into his text EXCEPT for the first character. The function will recognize automatically the correct variable.

SERAQNT()

purpose function to transform quantities or amounts to a character string

syntax seraqnt(<n1>,<n2>,<n3>,<l>,<c>)

<n1>	input amount or quantity			
<n2>	length of the amount			
<n3>	number of decimals			
<l>	.T. = USA presentation			1,000,000.00
	.F. = European			1.000.000,00
<c>	'C '	-99	will be	99 CR
	'D'	99	will be	99 DT
	'('	-99	will be	(99)

‘-‘	-99	will be	99 -
‘B’	0	will be	blank
‘R’	-99	will be	99 (Reversed)
‘*’	9	will be	*9
‘A’	-99	will be	99 (Absolute)

returns the transformed value as a character string

example seraqnt(-1234.5,10,2,.F.,’DC’) returns 1.234,50 CR
seraqnt(91234.5,12,2,.T.,’*DC’) returns ***91,234.50 DT

C2D()

purpose function to convert a YYYYMMDD string into a date variable

syntax C2D(<c1>)

<c1> character string in format YYYYMMDD

returns Date as YYYY.MM.DD in the Clipper DATE format

D2C()

purpose function to convert a Clipper date field into a YYYYMMDD string

syntax D2C(<d1>)

<d1> date in the format YYYY.MM.DD

returns a character string in the format YYYYMMDD

D2W()

purpose function to convert a YYYYMMDD in character format or in date format into character string YYYYWW

syntax D2W(<c1>) or D2W(<d1>)

<c1> character string in format YYYYMMDD
<d1> date in format YYYY.MM.DD

returns Character string YYYYWW

WEEKDIFF()

purpose	function to calculate the number of week 1 compared to week 2
syntax	WEEKDIFF(<d1>,<d2>,[<max>])

<d1>	date or character string in any format
<d2>	date or character string in any format
<max>	the max. difference. Default is 10.

returns the difference in weeks + 1

NOTE The return value is increased with 1.
In case 2 dates are in the same week the result will be 1.
Negative values and zero are rounded to 1.
Differences > than the <max> parameter are rounded to the <max> value.

D2B()

purpose function to convert a YYYYMMDD in character format or in date format into character string STR(999999999 – VAL(YYYYMMDD),8). This function can be used to search an index on a YYYYMMDD field that is descending sorted.

syntax D2B(<c1>)   or D2B(<d1>)

<c1>	character string in format YYYYMMDD
<d1>	date in format YYYY.MM.DD

returns Character string STR(99999999-VAL(YYYYMMDD),8)

DATEOK()

purpose function to check if a date or character variable is in a correct YYYYMMDD format.

syntax DATEOK(<c1>)   or DATEOK(<d1>)

<c1>	character string in format YYYYMMDD
<d1>	date in format YYYY.MM.DD

returns .T. or .F.

WEEKOK()

purpose function to check if a week variable is in a correct YYYYWW format.

syntax WEEKOK(<c1>)

<c1>	character string in format YYYYWW

returns .T. or .F.

PERIOK()

purpose function to check if a period variable is in a correct YYYYPP format. This function
is not checking if the period exists or is opened in \SERA\DAT00\PERIOD

syntax PERIOK(<c1>)

<c1>	character string in format YYYYPP

returns .T. or .F.

GETDATE()

purpose formatted user date iput can be created by using GETDATE instead of the
` standard GET.

syntax <datvar> = GETDATE(<row>,<col>,<datvar>)

<datvar>	the name of the variable to be filled.

returns A character string in the format YYYYMMDD.

SHOWDATE()

purpose This function presents the standard YYYYMMDD format as user definded format.

syntax SHOWDATE(<datvar>[,<type>])

<datvar>	the variable containing the YYYYMMDD character string	
<type>	0 or empty	6 digits no century, no separators
	1	8 digits no century, with separators
	2	10 digits with century and separators

3	8 with century, no separators

returns The converted date format.

DISCOUNT

purpose procedure to calculate customer discounts

description the customer discount on parts and part groups is calculated by using the discount file.

syntax do discount with <line>,<head>,<part>,<group>,<deldate>,<retail>,<qnt>

<line>	the line code of the customer, used to find if any discount is given to a general and dummy customer PL<line>
<head>	the number of the head-office of the customer
<part>	the part for which the discount is searched. If no part is required, but a part group (cost line) , the input has to be replicate('-',10).
<group>	the part group of the part or the group in case of cost lines.
<deldate>	the delivery date. The discount file is using date limits for the validation of a discount.
<retail>	the retail price of a part of a cost line
<qnt>	the quantity of the items for which the discount has to be calculated.

returns The following variables have to be defined before calling the discount procedure. The procedure will fill them except for D-VALUTA.

F_NETT	the net amount per item in the customer currency
D_NETT	the net amount per item in the local currency
D_NETTMARK	space in case the net amount is based on a discount percentage, '*' if the net amount is based on a net price from the discount table, or no discount at all was found.
D_DISC	discount percentage. The percentage is also calculated when a net price is found.
D_VALUTA	the currency of the customer.

example

```
SELECT CUST
LINE          = CUBRANCH
HEAD          = CUHEAD
SELECT PARTS
PART          = PANUMBER
PRTGROUP      = PAGROUP
PRICE         = PARETAIL
QNT = 0
@ 10,20 GET QNT
READ
DATE = SPACE(6)
@ 11,20 GET DATE
READ
```

```
F_NETT         = 0
D_NETT         = 0
D_NETTMARK  = ' '
D_DISC         = 0
D_VALUTA      = VS
DO DISCOUNT WITH LINE,HEAD,PART,PRTGROUP,DATE,PRICE,QNT
```

PROCBUTT()

purpose function to draw a button, pressed or non pressed on the screen. The function can also be used to draw boxes without shadow.

syntax PROCBUTT(<row1>,<col1>,<row2>,<col2>,<txt>,<on/off>)

<row1>	upper row
<col1>	left column
<row2>	lower row
<col2>	right column
<txt>	text to be displayed. In case only a box is required, this parameter has to be SPACE(1).
<on/off>	.T. is a pressed button, .F. a non-pressed button

returns nil

SERADR()

purpose function to locate the drive where the data bases can be found

description Sera offers the possibility to locate executables on another disk than the disk where the data are located.

syntax SERADR()

returns A different drive character or NIL.

P13()

purpose function to calculate the financial booking period of a date.

description The default period is equal to the month (YYYYMM) of the date. Some organizations have other periods, like 13 periods of 4weeks per year. Instructions for non-default calculations are collected from a parameter file called \SERA\PARAMET\PERIOD13.PAR.

syntax P13(<date>)

<date>	a character string in the format YYYYMMDD

returns a character string of the format YYYYPP.

D13()

purpose	function to calculate the start and finish dates of a period.
syntax	D13(<numtype><period>)

<type>	1 = start date 2 = finish date
<period>	a character string in the format YYYYPP

returns	a character string of the format YYYYMMDD

SERASTAT

purpose	procedure to fill the order and sales statistics.
syntax	DO SERASTAT WITH <peri>,<area>,<group>,<cust>,<part>,<head>, <inv_q>,<inv_a>,<kit_q>,<cst_a>,<int_q>

<peri>	book period
<area>	area or replicate('-',4)
<group>	part group
<cust>	customer number or replicate('-',10)
<part>	part number or replicate('-',10)
<head>	head office of the customer or replicate('-',10)
<inv_q>	invoice quantity or 0
<inv_a>	invoice amount or 0
<kit_q>	quantity invoices as part of a kit or 0
<cst_a>	cost price or 0
<int_q>	quantity of non-sales stock movement or 0

PART_SEEK()

purpose	function to seek parts and related information
description	a part can be found by entering a partial code or description. While seeking, the user can get information about available stock and the stock forecast.
syntax	PART_SEEK(<input>,<info>)

<input>	the initial code to seek with. A dot + code will open the parts file with the index on part number. Any other entry will open the parts file with the index on the part description.

returns the selected part number or CHR(174) in case the user didn't make a selection.

CUSTSEEK()

purpose function to seek customers

description a customer can be found by entering a partial code or description.

syntax CUSTSEEK(<input>,[<fromnr>])

<input>	the initial code to seek with. A dot + code will open the customer with the index on the customer number. Any other entry will open the customer file with the index on the name.
<fromnr>	An optional parameter that indicates the first customer to start with. Used in an From – Till query program.

returns the selected customer number or CHR(174) in case the user didn't make a selection.

SUPPSEEK() PROSSEEK() PERSSEEK() ACCO_SEEK()

Same as CUSTSEEK for SUPPLIERS, PROSPECTS, PERSONS and LEDGER ACCOUNTS.

CREATETEMPFILE()

purpose create a temporarily file with an index

description a temporarily file and index will be created at the location \SERA\TEMP with the names A<time>.DBF and A<time>.NTX

syntax CREATETEMPFILE(<field array>,<data file>,<memo file>,<index file>,<index key expression>)

returns Nothing

example

```
// Define a multi dimensional array with the field name,type, length and
number of decimals
AFIELDS = {}
AADD(AFIELDS,{'NUMBER','c',10,0})
AADD(AFIELDS,{'DESCRIPTION','c',30,0})
AADD(AFIELDS,{'QUANTITY','N',10,2})
```

```
CREATETEMPFILE(AFIELDS,'TEMPFILE','TEMPMEMO','TEMPINDEX','NUMBER')
USE &TEMPFILE INDEX &TEMPINDEX
// operation
…
// end
CLOSE DATA
DELE FILE &TEMPFILE
DELE FILE &TEMPMEMO
DELE FILE &TEMPINDEX
```

NOTE The memo file is only created in case a memo field was defined in the initial array.

TEMPTEXT()

purpose create a temporarily text file

description a temporarily file and index will be created at the location \SERA\TEMP with the names T<time>.TXT

syntax TEMPTEXT(<text file>)

returns Nothing

example

```
TEMPFILE('TEMPFILE')
SET ALTE TO &TEMPFILE
// operation
…
// end
CLOSE ALTE
DELE FILE &TEMPFILE
```

EXTRAFIELD()

purpose a set of functions to handle user defined fields

description users can defined extra fields for files like customers, suppliers in EXTFIELD.
This set of functions makes the entry, change and display of fixed data more easy.

syntax EXTRAFIELD(<data file>,<type>,[<extra parameterl>)
// extra parameter for type 4 and 7

returns Depending on <type>

TYPE = 1 **Check Occurence And Load Basics**

In case of changing existing records: Place after the definition of the CHANGEVARS
variable. The user defined fields are added to this variable.

A public variable EXTRAFIELD is created, indicating if there are userdefinedfields: A public variable SCR_NR is set to 1

Returns .T. or .F. (= variable EXTRAFIELD)

TYPE = 2 **Get new data**

The user is offered a screen to enter the user defined fields.

Returns .T. in case all entries are finished
Returns .F. in case the back function was used

TYPE = 3 **Fill existing data into variables**

Used after an existing record was selected for change

Returns NIL

TYPE = 4 **Change an existing field**

The user is offered a screen to enter the user defined fields.
The extra parameter indicates the field number.
If the value is zero, all fields are only displayed.

Returns NIL

TYPE = 5 **Update file with user defined fields**

The main file is update by the new or changed user defined fields.

Returns NIL

TYPE = 6 **Show user defined fields**

Used for showing existing data

Returns NIL

TYPE = 7 **Print user defined fields**

Used for printing existing data
The extra parameter indicates the position of the contents to be printed:

Extrafield('Suppl',7,20):
Contact Person Sam Stoker

Contact Phone	012 5 43443

Extrafield('Suppl',7,30):

Contact Person	Sam Stoker
Contact Phone	012 5 43443

Returns NIL

Examples

ENTER NEW RECORD

```
#INCLUDE 'SERA.CH'
EXTRAFIELD('SUPPL',1)    // CHECK OCCURENCE AND LOAD BASICS
                         // CREATES PUBLIC VARIABLE EXTRAFIELD .T./.F.
.....
DO WHILE .T.
 .....
 .....
 IF LABEL = 10
  NAME = SPACE(30)
  @ 10,30 GET NAME
  READ
  IF LASTKEY() = 27
   LABEL = LABEL -1
  ELSE
   // END OF INPUT OF STANDARD FIELDS
   IF .NOT. EXTRAFIELD          // PUBLIC VARIABLE
    LABEL = LABEL + 1
   ELSE
    IF EXTRAFIELD('SUPPL',2)    // INPUT OF USER DEFINED FIELDS
     LABEL = LABEL + 1
    ENDIF
   ENDIF
  ENDIF
 ENDIF
 IF LABEL = 11
  OK = ' '
  @ 12,02 SAY 'CONTINUE?       '
  @ 12,30 GET OK PICT "!"
  READ
  IF OK $'JYOS'
   APPEND BLANK
   DO WHILE .NOT.RLOCK()
    APPE BLANK
   ENDDO
   REPLACE SUNUMBER WITH NR
   REPLACE SUNAME WITH NAME
   EXTRAFIELD('SUPPL',5)       // UPDATE USER DEFINED FIELDS
   UNLOCK
   LABEL = 0
  ELSE
   .....
  ENDIF
 ENDIF
ENDDO
```

CHANGE EXISTINGRECORD

```
#INCLUDE 'SERA.CH'
CHANGEVARS = '|NAME|CITY|'
EXTRAFIELD('SUPPL',1)
LABEL = 0
DO WHILE .T.
 IF LABEL = 0
  .....
  @ 03,00 SAY '  NUMBER'
  @ 05,00 SAY '2.NAME'
  @ 06,00 SAY '3.CITY'
  NR = SPACE(10)
  @ 03,30 GET NR
  READ
  .....
  IF FOUND()
   NR = SUNUMBER
   NAME = SUNAME
   CITY = SUCITY
   EXTRAFIELD('SUPPL',3)    //  FILL EXISTING USER DEFINED FIELDS
   LABEL = 50
  ENDIF
 ENDIF
 .....
 .....
 IF LABEL = 50
  FIELD = SPACE(1)
  @ 24,00 SAY '<'+FKA10+'> = END CHANGES        '
  IF EXTRAFIELD
   IF SCR_NR <= 1
    @ 24,20 SAY '<'+FKA02+'> page+'
    SET FUNC FKN02 TO '>'+CHR(13)
   ENDIF
   IF SCR_NR = 2
    @ 24,20 SAY '<'+FKA02+'> page-'
    SET FUNC FKN02 TO '>'+CHR(13)
   ENDIF
  ENDIF
  CHANGESTAT = .T.
  @ 24,30 SAY 'FIELD TO BE CHANGED ' GET FIELD PICT "!"
  READ
  CHANGESTAT = .F.
  IF "."+ALLTRIM(STR(VAL(FIELD),2))+"."$"."+TRIM(CHANGELOCK)+"." .AND. VAL(FIELD) > 0
   @ 24,30 SAYMESS "No authorisation."
   FIELD = SPACE(LEN(FIELD))
  ENDIF
  IF LASTKEY() = 27
   LABEL = 51
  ELSE
   IF '>'$FIELD
    IF SCR_NR = 2
     RESTORE SCREEN FROM EXTRASCR
     SCR_NR = 1
    ELSE
     SAVE SCREEN TO EXTRASCR
     EXTRAFIELD('SUPPL',4,0)
    ENDIF
    FIELD = ' '
   ENDIF
   IF VAL(FIELD) > 1 .AND. VAL(FIELD) < 4
    LABEL = VAL(FIELD)
   ENDIF
   IF VAL(FIELD) >= 51 .AND. VAL(FIELD) <= 50+LEN(ITEMS)
    IF SCR_NR <= 1
     SAVE SCREEN TO EXTRASCR
    ENDIF
    EXTRAFIELD('SERIAL',4,VAL(FIELD) - 50)    // CHANGE FIELD 51 OR 52 ETC.
   ELSE
    IF LABEL <> 50
     IF SCR_NR = 2
      RESTORE SCREEN FROM EXTRASCR
      SCR_NR = 1
     ENDIF
    ENDIF
   ENDIF
  ENDIF
 ENDIF
 IF LABEL = 31
  OK = SPACE(1)
  @ 24,00 SAY '<'+FKA10+'> = BACK TO CHANGE'
  @ 24,30 SAY 'CONTINUE?            '
  @ 24,67 GET OK PICT "!"
  READ
  IF OK $'JYOS'
   DO WHILE .NOT.RLOCK()
   ENDDO
   REPLACE SUNAME WITH NAME
   REPLACE SUCITY WITH CITY
   EXTRAFIELD('SUPPL',5)       // UPDATE USER DEFINED FIELDS
   UNLOCK
   LABEL = 0
  ELSE
   .....
  ENDIF
 ENDIF
ENDDO
```

SHOW / PRINT RECORDS

```
#INCLUDE 'SERA.CH'
IF EXTRAFIELD('SERIAL',1)
 MAXPAGE = 2
ELSE
 MAXPAGE = 1
ENDIF
....
DO WHILE .T.
 ....
 IF OP = '1'
  DO PROCTITL
  SELE SERIAL
  SEEK A2
  SCR_NR = 1
  DO WHILE SENUMBER <= A3 .AND. .NOT. EOF()
   IF SCR_NR = 1
    DO PROCTITL
    @ 02,03 SAY 'NUMBER '
    @ 04,03 SAY 'Description      '
    ......
    @ 02,26 SAY SENUMBER
    @ 02,26 SAY SEDESCR
   ENDIF
   IF SCR_NR = 2
    EXTRAFIELD('SERIAL',6)
   ENDIF
   END = .F.
   IF SCR_NR = MAXPAGE
    SKIP
    IF SENUMBER > A3 .OR. EOF()
     END = .T.
    ENDIF
    SKIP - 1
   ENDIF
   IF END
    OK = ' '
    @ 24,30 SAY “ENTER = END” GET OK PICT "!"
    READ
    EXIT
   ELSE
    OK = ' '
    IF SCR_NR < MAXPAGE
     @ 24,30 SAY “NEXT PAGE?  “ GET OK PICT "!"
    ELSE
     @ 24,30 SAY “NEXT NUMBER?“ GET OK PICT "!"
    ENDIF
    READ
    IF LASTKEY() = 27
     SCR_NR = SCR_NR - 1
     IF SCR_NR = 0
      SCR_NR = 1
     ENDIF
    ELSE
     IF OK = "N"
      EXIT
     ELSE
      SCR_NR = SCR_NR + 1
      IF SCR_NR > MAXPAGE
       SCR_NR = 1
       SKIP
      ENDIF
     ENDIF
    ENDIF
   ENDIF
  ENDDO
  ......
 ENDIF
 IF OP $'23'
  DO PROCTITL
  I = 0
  DO SERAPRIN
  SET COLOR TO &CR1/&CR2,&CR7/&CR8
  IF I = FKV10
   CLOSE DATA
   RETURN
  ENDIF
  SET PRIN ON
  SET CONSOLE OFF
  SELE SERIAL
  SEEK A2
  I = 0
```

```
  .....
  DO WHILE SENUMBER <= A3 .AND. .NOT.EOF()
   .....
   ? 'NUMBER                  ',SENUMBER
   ?
   ? 'Description             ',SEDESCR
   ....
   EXTRAFIELD('SERIAL',7,26)      // PRINT USER DEFINED FIELDS
   ......
   SKIP
  ENDDO
  ....
 ENDIF
ENDDO
```

HOT_AREA

purpose check if the user has clicked with the mouse on special screen-area's .

description an array containing the area's is filled.
The return-value of the INKEY() function will give the element number of the array in case an area is clicked on.

syntax HOT_AREA ={ {<Y1>,<X1>,<Y2>,<X2>} [,{<NY1>,<NX1>,<NY2>,<NX2>}],{….}}

returns In case one of the areas is clicked on, the INKEY() function will return the element number of HOT_AREA that contains the coordinates of that area. To prevent confusion with other return values, the value is increased with 500000.

example

```
HOT_AREA = {}
AADD(HOT_AREA,{10,10,12,12})                    // button one
AADD(HOT_AREA,{13,10,15,13})                    // button two
PROCBUTT(10,10,12,12,'off',.F.)
PROCBUTT(13,10,15,12,'off',.F.)
I = INKEY(0)
IF I = 500001
 PROCBUTT(10,10,12,12,'on',.T.)
ENDIF
IF I = 500002
 PROCBUTT(10,13,12,15,'on',.T.)
ENDIF
HOT_AREA = ()// remove hot-spots
```

WINTAB()

purpose create typical Windows tab's.

description tab's are shown and mouse clicks are captured.

Mouse capturing can be switched off and back to on by changing the public variable TABSTAT from the default .T. into .F. and back into .T.

syntax WINTAB(<type>,<acttab>)

<type>	1 - the tabs are build from left to right
	2 - the tabs are build from right to left
<acttab>	the tab-number that is selected.

Note: BOXSIZE and BOXTITLE are variables that have to be filled before. See example.

returns In case one of the tab areas is clicked on, the GET or INKEY() function will return a lastkey() value of 13 (enter).
A public variable called LST_TAB is filled with the tab number.
This value is also the return value.

example

```
BOXSIZE  = '04|02|20|75'                  // the screen area
BOXTITLE = 'Tab one|Tab two|Tab three'    // the tab titles
WINTAB(1,1)                               // show tab with the
                                          // 1st tab selected.
DO WHILE .T.
 TABSTAT = .T.                            // set to active
 I = INKEY(0)
 TABSTAT = .F.                            // prevent tab
                                          // change
 IF LASTKEY() = 13
  IF LST_TAB = 1
   DO PROCTITL
   WINTAB(1,1)
   PAGE(1)                                // do page 1
  ENDIF
  IF LST_TAB = 2
   DO PROCTITL
   WINTAB(1,2)
   PAGE(2)                                // do page 2
  ENDIF
  …
 ENDIF
ENDDO
```

PROGBAR()

purpose show a progression bar.

description The progression bar function offers a basic bar on the screen that can be updated

during the progression. The bar is divided in 10ths of 100%.
The total width is 65 characters. The height is 4 rows.

syntax PROGBAR(<type>,<actval>,<maxval>,<row><column>)

<type>	1 - initial setup and set cursor off

	2 - progression update
	3 - close bar with 100 fill and set cursor on
<actval>	the progression value or percentage
<maxval>	the maximum value or 100 (%)
<row>	the upper row of the bar
<column>	the left corner of the bar

returns No return is given.

example

```
PROGBAR(1,0,1000,17,7)
X = 0
DO WHILE X < 1000
  X = X + 1
  PROGBAR(2,X,1000,17,7)
ENDDO
PROGBAR(3,1000,1000,17,7)
```

NOTE: The screen is not saved and restored.

See also: SERAINDEX()

SAYTEXT

purpose a special form of the classic @ SAY function.

description it's used to prevent a character field or variable to be handled and displayed as a number.

syntax @ <row>,<column> SAYTEXT <TXT>

example

```
@ 10,10 SAY “10000”        will be displayed as      1000
@ 11,10 SAY “1000A”        will be displayed as 1000A
@ 12,10 SAYTEXT “10000”    will be displayed as 10000
```

SAYMARK

purpose a special form of the classic @ SAY function.

description it's used to present a Y(es) as a √ and a N(o) as a '**-**' .

syntax @ <row><column> SAYMARK <FIELD>

example @ 10,10 SAYMARK “Y”

GETMARK

purpose a special form of the classic @ GET function.

description it’s used to present a Y(es) as a √ and a N(o) as a ‘-‘ .
After clicking the input box with the mouse a √ (yes)
is changed into a “-“ (no) and visa versa.

syntax @ <row>,<column> GETMARK <FIELD>

example

```
@ 10,10 GETMARK “Y”
READ
@ 10,10 SAYMARK “Y”
```

SAYMESS

purpose a special form of the classic @ SAY function.

description it’s used to present a special message box. The windows version will ignore the line and the row position. The message is always displayed at the right lower corner of the screen. A message can have only 1 line.

syntax @ <row><column> SAYMESS <TXT>

example

```
@ 24,30 SAYMESS ‘ENTER ONLY A VALUA > 1’
```

SAYCONT

purpose a special form of the classic @ SAY function.

description it’s used to present a special message box with a ‘”continue” button.

syntax @ <row>,<column> SAYCONT <TXT>

example

```
@ 24,30 SAYCONT ‘NO DATA FOUND’
```

GETYNC()

purpose a special form of the classic @ GET function.

description A box appears with a question with 3 possible answers: Ýes, No, Cancel.

returns Depending on the user action a Y, N or chr(174).
The variable LASTKEY is set to 13 in case the anwer is Yes or No and to 27 in case the answer was cancelled.

syntax GETYNC <TXT>.
The text can have several lines, separated by a chr(13) + chr(10).

example

```
ANSWER = GETYNC('DO YOU WANT TO CONTINUE?')
IF ANSWER = [Y]
 …
ELSE
 IF ANSWER = [N]
…
```

PROCMARK()

purpose a function to collect Y/N answers from a list of options.

description an array is filled with option texts and the default answers.
The user gets an option list that can be changed.
After finishing it, the array is filled with the user input.

Returns 13 in case the user confirmed the input. In case the user cancelled the input a 27 is returned.

syntax PROCMARK(<upper row>,<left column>,<lower row>,<right column>)
The array with a fixed name AMARK is a multi dimensional array.
Each option has the syntax {<row>,<column>,<text>,<answer>}

example

```
AMARK = {}
AADD(AMARK,{4,4,'OPTION 1','Y'})
AADD(AMARK,{5,4,'OPTION 2','Y'})
AADD(AMARK,{6,4,'OPTION 3','N'})
IF PROCMARK(3,3,10,50) = 13
 ANSWER1 = AMARK[1,4]
 ANSWER2 = AMARK[2,4]
 ANSWER3 = AMARK[3,4]
ENDIF
```

TESTING

PURPOSE

Procedure for testing new or modified programs.

DESCRIPTION

The paragraphs marked with an asterix need to be documented to make a audit afterwards possible. A hand written document is preferred above a word processed one.

Discussion

Every program change has to be discussed with at least one other college to make an audit afterwards possible. A hand written document is preferred above a word processed one.

Integrity check *

Check the interactions with other programs:

1. What other programs are using the data ?

2. What other programs need to be updated to maintain the new functionality all over the Sera system?

example

A new function is required to exclude some customers from any service, because of bad payments.

1. Use Util0220 to find what programs are using the cust.dbf. Check where a calculation, quotation, order, invoice etc. is made for a customer.
 . The located programs need to get a warning function.

2. An Edi-program in the CCAT-range is transferring customer data to another location. This program needs to transfer the new information also.
.

build a case *

Write down from the program requirements

1. The starting values, quantities, description etc.
2. All the changes on the starting position.
3. The expected results of the operation.

From the databases :

4. The starting values, quantities, descriptions etc.
5. The contents of the mutation records
6. The final result

NOTE: Be sure that all indexes are reliable after some older tests

example

The program has to change the standard cost price of a part.

Description	**manual data**	**databases / programs**	**computer**
stock quantity	10	stock.dbf / sera6870	10
old price	2.5	parts.dbf / sera2540	2.5
stock value	25	- / sera6900	25
new price	3	parts.dbf / sera2540	3
new value	30	- / sera6900	30
change off stock value	5	moves.dbf / sera7310	5
		genledg.dbf / sera3300	5
		finstat.dbf / sera3310	5

execute the program

Do all steps. Try to backwards from the last step to the first step. ('The F10-test')

menu

Check if the menu is starting from within the menu system. Check also if the program is returning properly. Do this at least 4 times and start also programs from other modules.

check original requirements with the final result

Try to think as the user. Test if everything is working like the original design was meant.

use the check list on programs

The check list will guide you through all the steps, required to create a quality product.

SERATEST

A program called SERATEST.PRG can be used to test single programs. Just change the program to be tested in seratest.prg at the end by changing the instruction DO <prg> where <prg> has to be changed into the program name you want to test.

Make seratest.exe by running BLINK.BAT
Rename SERATEST.EXE into another name in case you want to keep this program for future usage.

NOTE: renamed seratest.exe programs can be confusing after some time, especially when there is not a proper documentation or when there is no source code.

SERATST.PRG together with BLINK1.BAT are an alternative

USER MANUALS

PURPOSE :

instruction how to create user manuals and an on-line help file.

DESCRIPTION

User manuals have to contain several paragraphs depending on the importance of the program.

purpose block

A one-liner showing the basic function of the program.

description

An explanation of the program and it's functionality

references

References to other programs.

environment

Where, when, how and by who is the program used?
What's is the final goal of using this program ?
What are the links with other modules ?
What preparations are required from the user ?
This paragraph has to give answers on these questions, before explaining the actual program.

case

Examples of entries, calculations, results, procedures etc..

TABLE OF PARAGRAPHS

type of program	purpose	description	references	environment	case	layout
listing without updates	-					
change of existing data	-					
create simple master data	√	√				
create complex master data	√	√	√	√		
main entry programs	√	√	√	√	√	√

Note :	An important part of the information is already stored in SERA\DATSTART\FILEUSE. The usage of databases per program can be printed by Util0210 or Util0220.

VERSIONS

PURPOSE

Procedure for version control and release management.

DESCRIPTION

SERA has 4 versions of the basic, English sources. The versions are called after the disk names where they are located.

OLD VERSION (not shown on download.serasoft.com)

general	This version is the oldest version. It has to be replaced by the actual J-version.
databases	No changes allowed at all times.
sources	Changes are not allowed except for bugs that are causing errors in databases. Any other change is not allowed, even when it's an improvement.
customizing	No customizing is allowed.
distribution	It is not allowed to do new installations of this version.
location	zip file at download server

ACTUAL VERSION (shown as OLD version on download.serasoft.com)

general	This version is an old version, not recommended
databases	Changes in the existing database structures are not allowed.
sources	Allowed is : 1.Solving all kind of bugs. Not allowed is : 1.Any change that updates the existing database. 2.Any change in existing programs that requires a new database. 3.Any functionality change in existing programs.

	4.Customized changes in existing programs.
distribution	No distribution
location	zip file a download server

BETA VERSION (shown as ACTUAL on download.serasoft.com)

sources	V: d:\drives\serasoft
libs	V:\sera\vslib
general	the beta version is stable version for customers that don't want the latest developments. It's not allowed to place unfinished or untested programs on this version.
databases	No changes allowed
customizing	Preferably no changes
distribution	Only existing customers

TEST VERSION (shown as LATEST on download.serasoft.com)

sources	Q: / T: d:\drives\seratest
libs	V:\sera\qslib
general	This is the version where new developments are done. It is special for isolated servers, where only remote desktop / remote app access is allowed.
databases	No restrictions
sources	No restrictions
distribution	Existing and new customers.