



Customizing

SoftOne

The Black Book

ver. 3.5 English

CONTENTS

Use Alt+Left arrow to return from hyperlinks target (Previous View)

Contents	2
1. Screen Forms / UI Controls.....	14
Overview	15
A. Form Design.....	16
A.1 Basic Features	18
A.2 Fields	19
A.3 Form Parameters / Settings	20
A.4 User Defined Tables	21
A.5 String lists.....	29
A.6 Script	31
A.7 Layout	32
A.8 Access Rights	33
B. Layout Controls.....	34
B.1 Tabs / Pages	34
B.2 Panels	35
B.3 Nested Panels	36
B.4 Sub Forms.....	37
C. Data Controls	43
C.1 String Textbox	43
C.2 Numeric Textbox	45
C.3 Memo Textbox	47
C.4 Password Textbox (Hash SHA-2).....	49
C.5 Webpage / Email Textbox.....	50
C.6 Grid	51
C.7 Selector List	55
C.8 DropDown List (Memory Tables)	56
C.9 Text Picker	57
C.10 Checkbox.....	58
C.11 Multiple Checkbox	59
C.12 Image.....	60
C.13 HTML Editor	61
C.14 SoftOne Spreadsheet	62
C.15 Datetime.....	63
C.16 Embedded QuickView.....	66
D. Dialog Controls.....	69
D.1 File Picker	69
D.2 Color Picker	70
D.3 Printers Picker.....	71
D.4 Time Scheduler	72
E. Command Controls	73
E.1 Buttons.....	73
E.2 Hyperlinks	75
F. Attached / Related Files.....	76
G. Editor Commands	82
H. Editor Attributes.....	83

2. Browsers.....	85
Overview	86
A. Modules.....	88
B. Columns.....	89
C. Filters	91
D. Grouping	92
E. Sorting	93
F. Parent / Child Browsers	94
G. User-defined modules.....	96
3. Printout Forms.....	99
Overview	100
A. General Printing Features	100
A.1 Printer Settings.....	100
A.2 Available Printing Methods.....	101
A.3 Printout Form Types.....	101
A.4 Import / Export Printout Forms	102
B. Internal Printout Forms.....	104
B.1 Sections	104
B.2 Toolbar.....	105
B.3 General Form Operations.....	107
B.4 Page Size.....	108
B.5 Draft Printout Fonts	109
B.6 Image Printout Fonts	109
B.7 Recurrent Zone.....	110
B.8 Band Image.....	111
B.9 QR Code	111
C. Ms-Word Printout Forms	112
C.1 General	112
C.2 Form Design	113
C.3 Filters – Sorting – Grouping	114
C.4 Field decimals	114
C.5 Report Footer.....	115
C.6 Item Image.....	116
C.7 Text wrap	117
D. Ms-Excel Printout Forms	118
D.1 General.....	118
D.2 Form Design.....	119
E. Label Printout Forms	120
F. Crystal Reports Printout Forms.....	122
F.1 Basics	122
F.2 Crystal Report Design	123
F.3 Reports using tables	124
F.4 Reports using SQL statements	127
F.5 Import Crystal Report into SoftOne	128
G. Automated Jobs	130
G.1 Auto Email.....	130
G.2 Auto Save to file	132

4. Alerts – EDA Event Driven Actions	133
Overview	134
A. Events & Conditions	134
A.1 Field Rule (On Change)	134
A.2 Table Rule	136
A.3 Module Rule	137
A.4 Browser – Search Condition	139
B. Actions	140
B.1 Send Message	140
B.2 Message Display (Notification)	141
B.3 Error Display (Exception)	141
B.4 Run	142
B.5 Update	142
B.6 Reminder	142
C. Case Studies	143
C.1 Field Rule – Message display	143
C.2 Table Rule – Error Message Display	144
C.3 Object Rule – Send Message	145
C.4 Browser – Search Condition	145
5. User-Defined Fields	146
A. General Features	147
B. Operation Commands	149
B.1 Line Calculation	150
B.2 Calculation of line+totals	153
B.3 SQL command	154
B.4 Question	155
B.5 SQL Filter	157
B.6 Single filter / Multiple filter	158
B.7 Function	159
B.8 Multi Results SQL	159
C. Built-in Functions	163
Abs (Param1: double): double	164
GetConvertSeries (SOSOURCE: number, SERIES: number): string	164
FormatDateTime (Param1:string, Param2: Date): String	165
SetValue (Param1: string, Param2:variant):integer	166
RunSQL (Param1:string):string	167
Copy (Param1: string, Param2, Param3: integer): string	169

6. Database Designer.....	170
A. General Features.....	171
A1. Synchronization	172
A2. Publication – Custom Administration	175
B. Fields.....	177
B.1 Field Data Types.....	178
B.2 Field Properties	178
B.3 Selector Fields.....	179
C. Tables.....	180
C.1 Table properties.....	180
C.2 Memory Tables.....	188
C.3 Child Tables	191
D. String Lists	196
E. Database Views	198
E.1 Design	198
E.2 Advanced Browser Redirection	199
E.3 Examples	201
F. Objects	206
F.1 Design	206
F.2. Object Properties	207
F.3 Table Properties	210
F.4 Field Properties.....	213
F.5 Calculated Fields	214
F.6 Linked Tables.....	215
F.7 Browsers Design.....	216
F.8 Forms Design.....	217
F.9 Display Object in Menu.....	219
F.10 Printout Forms.....	220
F.11 Object Examples	221
G. Virtual Tables.....	225
G.1 Design Virtual Table.....	225
G.2 Virtual Table Example.....	225
H. Report Objects.....	228
H.1 Design Report Object.....	228
H.2 Report Object Example.....	231
7. Extra Tools.....	233
A. Auto Login from Windows Shortcut.....	234
A.1 Create a Windows Shortcut	234
A.2 Configuration of XCO connection file.....	234
A.3 XCO file Commands.....	236
B. Maximum Entries per Module (Select Top).....	237
B.1 Overview	237
B.2 Object Usage.....	237
B.3 Browser Usage.....	238
C. Menu Jobs	240
C.1 Create Job	240
C.2 Job Types.....	241
C.3 Menu Job Parameters – Menu Commands	242
C.4 Examples of Menu Jobs	244

8. Scheduler & Messages	245
A. Remote Server	246
A.1 Activation	247
A.2 Remote Server Commands	248
A.3 Send SMS - Email	249
B. Windows Scheduler	252
B.1. Overview	252
B.2 Job File	253
B.3 XCO Connection File	254
B.3 Windows Scheduler Task	255
C. SoftOne Scheduler	258
C.1 CreateTask	259
C.2 Scheduler Commands	261
D. Messages	262
9. Form Scripts	265
A. General Features	266
A.1 Script Editor	266
A.2 Debug Javascript	269
A.3 Global Variables	270
A.3 S1P & X.SYS	271
B. Object Methods	272
BATCHEXECUTE	272
BEEP	272
CALLPUBLISHED (Library.Function:string, params)	272
CANCELEDITS	272
CLOSEAPPLICATION	272
CLOSEFORM	272
CLOSESUBFORM (SubFormName: string)	273
COPY – PASTE	273
CREATEAPPLPOPUP(ModuleHandle, 0)	274
DBDELETE	274
DBLOCATE (KeyData: variant)	274
DBINSERT	275
DBPOST	275
EXCEPTION (Message: string)	276
FIELDCOLOR (FieldName: string; UserColor: integer)	276
FOCUSFIELD (FieldName: string)	276
FREE	276
INVALIDATEFIELD (FieldName: string)	277
INCLUDE (filename: string)	277
OPENSUBFORM (SubFormName: string)	277
QUICKVIEW (ObjectName: string, ListName: string, Keydata: string)	277
PRINTDOCURL (ObjectName, FormName, RecordID, TemplateCode)	278
PRINTFORM (TemplateCode: integer, PrinterName: string, FileName: string)	279
REFRESH	279
RUNSQL (ASQL: string; AParams: Variant)	279
SETDECIMALS (FieldName: string, Decimals: integer)	279
SETDOCPRINT (PrintNum, Mode, TempID: integer, PrinterName, Caption: string)	279
SETFIELDEDITOR (FieldName: string; Editor: String)	280
SETFIELDVALUE (FieldName: string, Value: Variant)	280
SETPARAM(PrmName, PrmValue);	280
SETPROPERTY ('EDITOPTIONS', 'READONLY=True/False')	280
SETPROPERTY ('MERCHANGELOG', 'True or False')	281

SETPROPERTY ('FIELD', 'FieldName', 'CAPTION', 'NewCaption')	281
SETPROPERTY ('FIELD', 'FieldName', 'VISIBLE', 'True/False or 1/0')	281
SETPROPERTY ('FIELD', 'FieldName', READONLY, 'True/False or 1/0')	281
SETPROPERTY ('PANEL', 'PanelName', 'CAPTION', 'NewCaption')	282
SETPROPERTY ('PANEL', 'PanelName', 'VISIBLE', 'True/False or 1/0')	282
TOFILE (FileName, AMessage: string)	282
WARNING (Message: string)	282
C. Object Functions	283
ASK (ACaption, AMessage: string): integer	283
ASKYESNO (ACaption, AMessage: string): boolean	283
CASE (IfCase, ThenCase, ElseCase: variant): Variant;	283
CHECKACCESS(JobName): Boolean	283
CONNECTIONSTATUS: string;	283
CREATEOBJ (ObjectName: string): OBJECT (IDispatch)	284
CREATEOBJFORM (ObjectName: string): OBJECT (IDispatch)	285
DOUBLE (StrNum: string): real	286
DIR (Name: string): string	286
EVAL (Formula: string): Variant	286
EXEC (Command: string): variant	286
FINDTABLE (TableName: string): variant	289
FILTERSUM (FieldName: string, Filter: string): real	289
FORM: string	289
FORMATFLOAT (number: float, decimalsformat: string): string	290
FORMATDATE (Dateformat: string, Date: TDateTime): string	290
FORMATTEXT (Text: string, parameters: string): string	290
FROMFILE (FileName: string): string	290
GETYEARPERIOD (ADate: TDateTime): Variant	291
GETLASTERROR: string	291
GETPARAM(PrmName)	291
GETPROPERTY('CUSTOMFIELDS, TABLENAME'): string	292
GETPROPERTY('MASTERTABLENAME'): string	293
GETPROPERTY('HASCHANGES'): boolean	294
GETPROPERTY('GRIDSELECTED:GRIDNAME FIELDNAME'): string	295
GETSQLDATASET (ASQL: string; AParams: Variant): TDataset	297
HTTPCALL (URL: string, PostData: string, Headers: string, Method: string): Variant	298
ID (TableName: string, CodeValue: string): integer	298
INPUTBOX (Prompt: string, DefaultValue: string): string	299
INPUTQUERY (Title: string, Prompt: string, DefaultVal:Variant, vPassword:integer) : Variant	299
ISVALIDCONTRACT: boolean	299
LIST: string	299
LOCALE: integer	299
LOCALESTRING (value: string): string	299
LOGINDATE: TDateTime	299
MULTISQL: variant	299
NEWID: integer	302
PASSWORDVALIDATE(stringtoValidate: string, Password: string): boolean	302
PLAY (SoundFileName: string): boolean	302
SHOWOBJFORM: integer	302
SPELL (Num: Real): string	303
SQL (ASQL: string; AParams: Variant): string	303
STRINGS (StringList: string, Key: string): string	304
SUM (FieldName: string): real	304
TEMPDIR: string	304
TIME: string	304
USERVALIDATE(UserName: string, Password: string): boolean	304

WEBREQUEST (JSONRequest: string): string	305
WSCALL (scope: variant, uri: string, postData: string, callbackFunc: variant): string	307
D. Dataset Methods	308
APPEND	308
DELETE	308
DISABLECONTROLS	308
ENABLECONTROLS	308
EDIT	309
FIRST	309
INSERT	309
LAST	309
NAME	309
NEXT	310
POST	310
PRIOR	310
SETREADONLY (FieldName: string, Value: True/False)	310
SETDATASETLINKS(ModuleHandle, DatasetHandle, Value)	311
SETFIELDPROPERTY(FieldHandle, PropertyIndex, PropertyValue);	311
GETFIELDPROPERTY(FieldHandle, PropertyIndex);	311
E. Dataset Functions	313
ACTIVE: boolean	313
BOF: boolean	313
EOF: boolean	313
FIELDBYNAME (FieldName: string): variant	313
FIELDCOUNT: integer	313
FIELDNAME (index: integer): string	314
FIELDTYPE (FieldName: string): integer	314
FIELDS (index: integer): variant	314
FILTER: string	314
FILTERED: boolean	315
GETGRAPH (LabelFieldName: string; ValueFieldName: string): string	315
GETHTML (FieldNames: string): string	315
GETXML (WriteMetadata: Boolean, DisplayTagsOnNullValues: Boolean): string	316
ISNULL (FieldName: string): boolean	316
JSON: string	317
LOCATE (KeyFields: string; KeyValues: variant): boolean	317
LOOKUP (KeyFields: string; KeyValues: Variant; ResultFields: string): Variant;	317
RECNO: integer	317
RECORDCOUNT: integer	318
STATE: integer	318
SUM (FieldName: string, Filter: string): real	318
F. Field Events	319
ON_ObjectName_FieldName_VALIDATE	319
ON_ObjectName_FieldName	319
G. Datagrid Events	320
ON_DatagridName_NEW	320
ON_DatagridName_POST	320
ON_DatagridName_AFTERPOST	320
ON_DatagridName_BEFOREDELETE	321
ON_DatagridName_AFTERDELETE	321
H. Object Events	322
ON_CREATE	322
ON_FORMLOAD	323
ON_DESTROY	323

ON_CANCEL	323
ON_LOCATE	323
ON_POST	324
ON_AFTERPOST	324
ON_DELETE	324
ON_AFTERDELETE	324
ON_INSERT	324
ON_DOCPRINT	325
ON_OPENREPORT	325
EXECCOMMAND (cmd)	325
EXECCOMMAND(cmd) (cmd=-1)	325
EXECCOMMAND(cmd) (cmd=-2)	325
ON_RESTOREEVENTS	325
ON_EDIT	326
ON_AFTEREXECUTE	326
I. Sub Form Events	328
ON_SubFormName_SHOW	328
ON_SubFormName_ACTIVATE	329
ON_SubFormName_ACCEPT	329
ON_SubFormName_CANCEL	329
J. Report Objects Events	330
ON_BANDSTART(BandName)	330
ON_BANDEND(BandName)	330
ON_BANDLINE(BandName)	330
ON_BANDTRANSFER(BandName)	330
K. Advanced JavaScript	331
K.1 JSMain	331
K.2 Library	335
K.3 Debug Functions	336
Case Studies	337
1. Print Form using dialog that asks for number of copies	337
2. Delete related converted documents	339
3. Get Compressed Data	340
4. Run Batch Job	341
5. Custom Screen Filters	343
6. Virtual Table – Item Purchases	345
7. HTTP Request – Google Maps Distance	347
8. Run External Application	351
9. Add Related Jobs in SoftOne Objects	352
10. Run SBSL script without UI (filters screen)	353
11. Auto <i>apply ABC model</i>	354
12. SubMenu in Button	356
13. ASCII Export / Import encoded file	358
14. Get Browser field data	360
15. Add / Retrieve HTML data	363
16. Check / Add / Delete ABC (Activity Based Costing) data	366

10. Data Flows	367
A. Overview	368
B. Data Flow Rules	370
B.1 Source entity	370
B.2 Target entity	372
B.3 Runtime Settings	375
C. Data Flow Scenarios	376
D. Execute Data Flows from Screen Scripts	379
E. Case Studies	380
1. Sales to Purchases Conversion	380
2. Sales to Sales (Company)	382
3. Items to Purchases	384
4. Production to Purchases	387
5. Marketing Campaigns to Tasks	389
11. SoftOne Batch Script Language (SBSL)	391
A. Overview	392
B. Syntax Basics	393
B.1 Case Insensitive	393
B.2 Semicolons	393
B.3 Comments	393
B.4 Variables	393
B.5 Libraries References	393
B.6 Functions	393
B.7 Error Messages	394
B.8 Comparison symbols	394
B.9 Logical operators	394
C. Dialog screen design (Interface)	395
C.1 Section TABLES	395
C.2 Section CACHETABLES	398
C.3 Section PANELS	401
C.4 Section STRINGS	404
C.5 Section FIELDEXEC	405
C.6 Section SCRIPT	408
D. Main Code	410
D.1 Converters	410
D.2 Functions	410
D.3 Connection Types	412
D.4 Variables	413
D.5 Use Libraries	413
D.6 Execute Javascript from SBSL	413
E. SBSL Script Execution	414
E.1 Menu Execution	414
E.2 Browser Execution	415
E.3 Form Execution	417
E.4 Run using Parameters	419
F. Built-in Functions	420
Abs(Number: numeric): numeric;	420
AddMessage(Text: string);	420
CallPublished(FunctionName: string, Params: VarArray);	420
CharToOem (var:string);	420
Commit(Connectionstring: string);	420
Copy(Source: string, Start: integer, Count: integer);	420

DateOfDateTime(DateTime: TDateTime);	420
DayOfDate(DateTime: TDateTime): integer;	421
DayOfWeek(): integer;	421
DaysinMonth(DateTime: TDateTime): integer;	421
Delay(Time: Integer);	421
DeQuotedStr (text:string);	421
EncodeTime(Hours: integer, Minutes: integer, sec: integer, ms: integer): TDateTime;	421
ExecPrg(File: string, Params: string);	421
ExecSQL(Connection: string, Query: string, Params: VarArray);	421
Fetch SQLCursorName	421
FormatSQLText(Connection: string, text: string, Params: VarArray);	421
GetIndexVar (Array: VarArray, index: integer);	422
GetTimeHour(DateTime: TDateTime): integer;	422
GetTimeMin(DateTime: TDateTime): integer;	422
GetTimeSec(DateTime: TDateTime): integer;	422
GetTimems(DateTime: TDateTime): integer;	422
GetQueryDataset(Datasetname: string, Connection: string, Query:string, Params:VarArray);	422
GetQueryResults(Connection: string, Query: string, Params: VarArray);	422
GetWhere(XModule, ParamWhere: string, Param3(0=and,1=where));	422
GrConvert(text: string): string;	422
ImportModule(Module: string);	423
Len(text: string): integer;	423
ModuleCommand (Module, Soft1Command, Params: string);	423
MonthOfDate(DateTime: TDateTime): integer;	423
Now(): TDateTime;	423
OEMToChar (oem: var);	423
QuotedStr (text: string);	423
Pos(Text: string,Source: string): integer;	423
RaiseException(Message: string);	423
RefreshMemoryTable(MemoryTable: string);	424
Resultnum;	424
ReplaceStr (text: string, String1: string, String2: string);	424
RollBack(Connectionstring: string);	424
SafeCallPublished(FunctionName: string, Params: VarArray);	424
SafeExecSQL(Connection: string, Query: string, Params: VarArray);	424
SendResponse(Values, Fields: string);	424
Space (num: integer);	425
StartTrans(Connectionstring: string);	425
StrToDate(text: string, format: string): TDateTime;	425
StrToFloat (var: string);	425
StrToInt (var: string);	425
TableExists(Connection: string,TableName: string);	426
Trim(text: string): string;	426
Time():TDateTime;	426
VarArray(Var1, Var2, ..., Varn, n);	426
VarArrayDimCount (Array: VarArray);	426
VarArrayHighBound(Array: VarArray, Dim: integer);	426
VarToStr(Param: Variant);	426
XModule();	427
XSupport();	427
YearOfDate(DateTime: TDateTime): integer;	427
G. SoftOne Libraries	428
G.1 ModuleIntf	428
G.2 PiLib	437
G.3 SysRequest	442

H. Case Studies	451
1. ReUpdate Sales Documents	451
2. Import Customers from Excel	452
3. Add Customer Collections (UI)	455
4. Get String List Data	456
5. Read ASCII file using specific Codepage	458
12. Web Services	460
A. Overview	461
B. Open Enterprise Engine	462
B.1 Prerequisites & Setup	462
B.2 Custom Web & Mobile Apps	463
B.3 Web Accounts	464
C. Methods / API Calls	466
Ping (Get Method)	466
Refresh(Get Method)	466
Login(Post Method)	467
Authenticate(Post Method)	469
ChangePassword (Post Method)	470
GetObjects (Post Method)	471
GetObjectTables (Post Method)	472
GetTableFields (Post Method)	473
GetDialog (Post Method)	474
GetFormDesign (Post Method)	475
GetBrowserInfo(Post Method)	476
GetBrowserData(Post Method)	478
GetReportInfo(Post Method)	479
GetReportData(Post Method)	480
GetData(Post Method)	481
FileName(Get Method)	482
SetData(Post Method)	483
Calculate(Post Method)	485
DelData(Post Method)	486
GetSelectorData(Post Method)	487
SelectorFields(Post Method)	488
D. Case Studies	489
1. Get Items	489
2. Add Customer	491
3. Update Items	492
4. Add Customer Branch	493
5. Get Sales Documents	494
6. Add Sales Document	495
7. Add HTML data	496
8. Get HTML data	498
9. Add Related Files	500

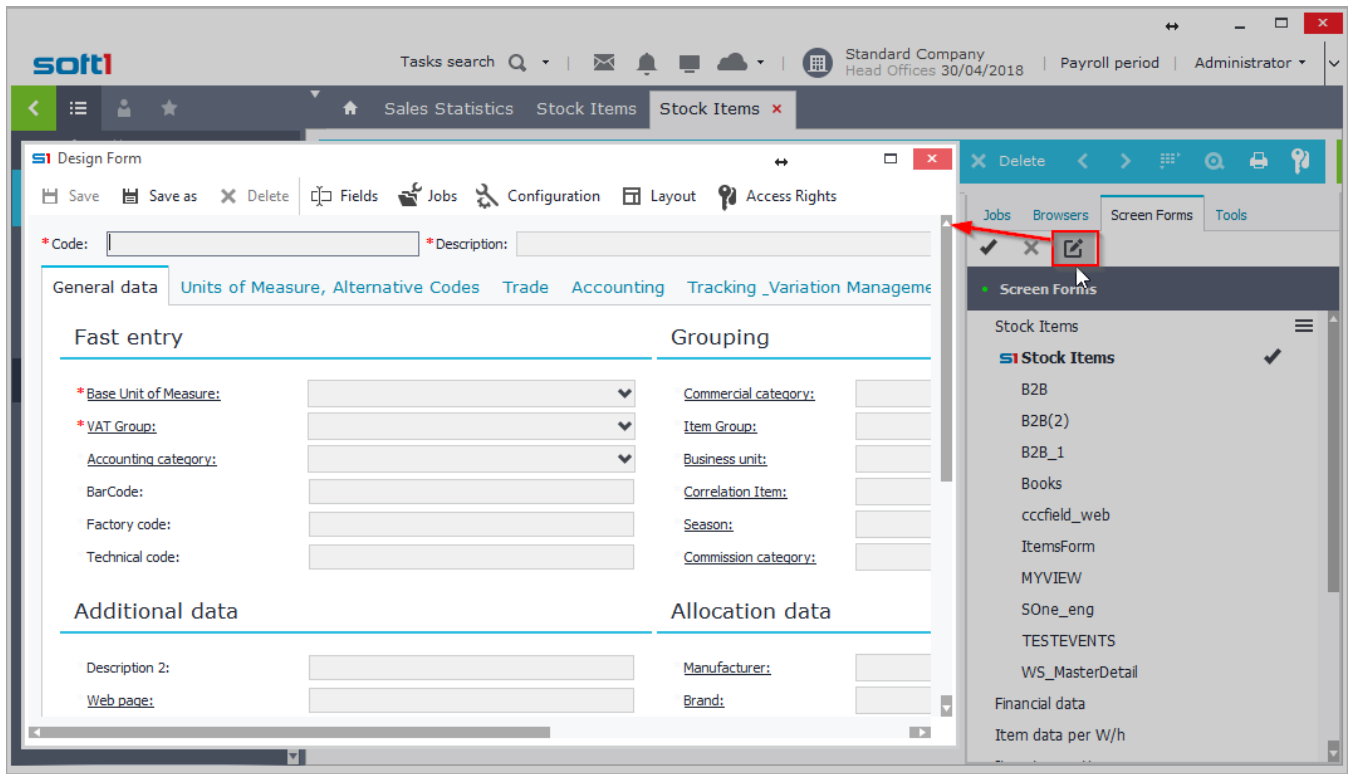
13. Business Automation Machine	502
A. Overview	503
B. Prerequisites & Setup.....	504
C. Design & Mode	506
C1. Available Tools	506
C2. Mode.....	507
C3. Process Testing	508
D. Variables.....	509
D.1 Calculation.....	510
D.2 SQL Command.....	512
D.3 Soft1 360.....	513
E. Process Steps	514
E.1 Start.....	514
E.2 Decision.....	518
E.3 Standby.....	520
E.4 User Action.....	521
E.5 Job	523
E.6 Sub Process	536
F. Runtime.....	537
G. Audit & Reports.....	539
G.1 Business Process Management	539
G.2 Business Process Analysis	540
Appendix System Parameters & Commands	541
A. SODTYPE – Entity Values.....	542
B. SOSOURCE – Module Values.....	543
C. ORIGIN – Transaction Source Values	544
D. CSTTYPE (CSTINFO Table)	545
E. X.SYS – System Parameters	546
F. ACMD Commands – System Tools.....	548
G. Editor Commands	551
H. Editor Attributes.....	552
I. Built-in Editors	554
J. Related Job Commands	557
K. Browser Job Commands	560
L. Command Line Switches.....	564
M. XCO: On-Premise Connection Settings	567
N. PARAMS.cfg: Cloud Connection Settings	571
O. Internal Object Parameters	573

1. SCREEN FORMS / UI CONTROLS

- A. [Form Design](#)
- B. [Layout Controls](#)
- C. [Data Controls](#)
- D. [Dialog Controls](#)
- E. [Command Controls](#)
- F. [Attached Files](#)
- G. [Editor Commands](#)
- H. [Editor Attributes](#)

Overview

Forms are visual surfaces of SoftOne objects (modules) that contain controls, such as Textboxes, Datagrids, EditLists, Checkboxes, etc. and offer a graphical environment to display and process data from database objects.



Form interface can be designed in any way that it meets your needs and can be exported to file (.cst) so as to use it in any other Softone installation. The tool for importing and exporting forms is "Custom Administration".

Note that every SoftOne object has at least one form (internal) that can not be updated. Instead, you can create as many custom forms as you like and save them with different names inside the object.

Form structure is saved inside the blob field SODATA of the database table CSTINFO (CSTTYPE=1), which makes it easy to transfer through different installations as .cst or .auv files using the "[Custom Administration](#)" tool.

Most of the actions users do inside form controls generate events. Forms respond to these events using custom script code (Javascript) and process them when they occur. For more information, see [Chapter Form Scripts](#).

SoftOne application provides many form controls with a rich feature set. Additionally, SoftOne provides a fully functional API which allows you to create DLL applications in .NET or Delphi, that can be displayed or interact with SoftOne object forms.

The controls that can be used inside SoftOne Forms are discussed in the following sections.

A. Form Design

Forms are designed either through the “Design” icon or through right click (Figure A1). The option “Export to (.cst)” exports the form as file, that can be later imported through “Custom Administration” tool.

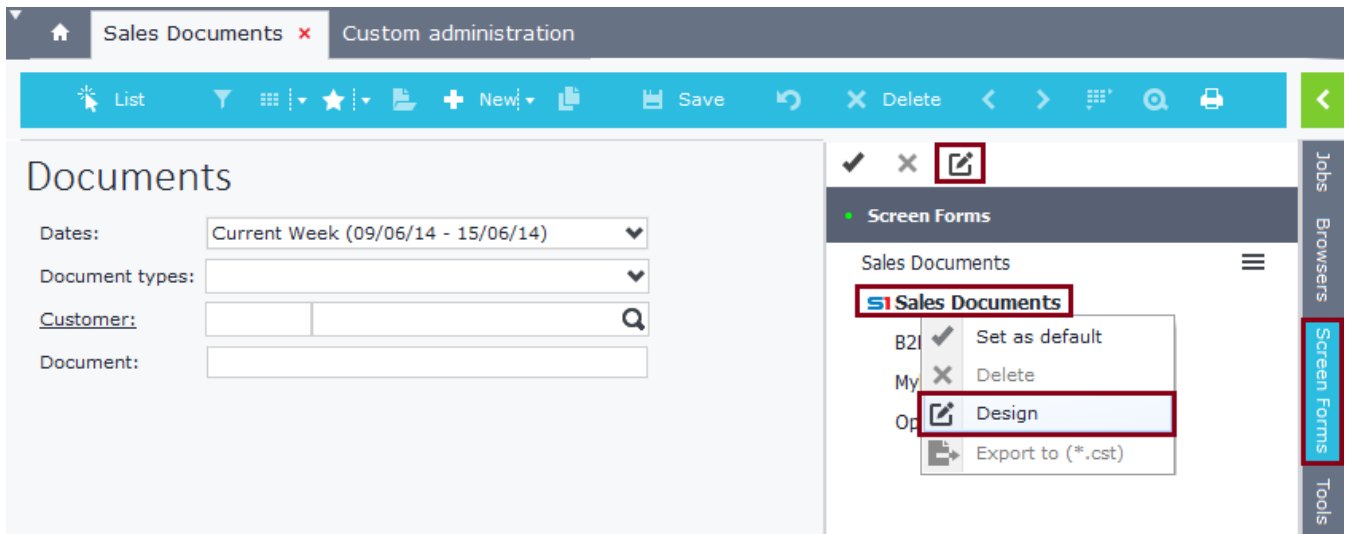


Figure A1

The window that is displayed is a WYSIWYG tool that allows you to alter the form layout. Use buttons “Save” and “Save as” to save the changes you have made (Figure A2).

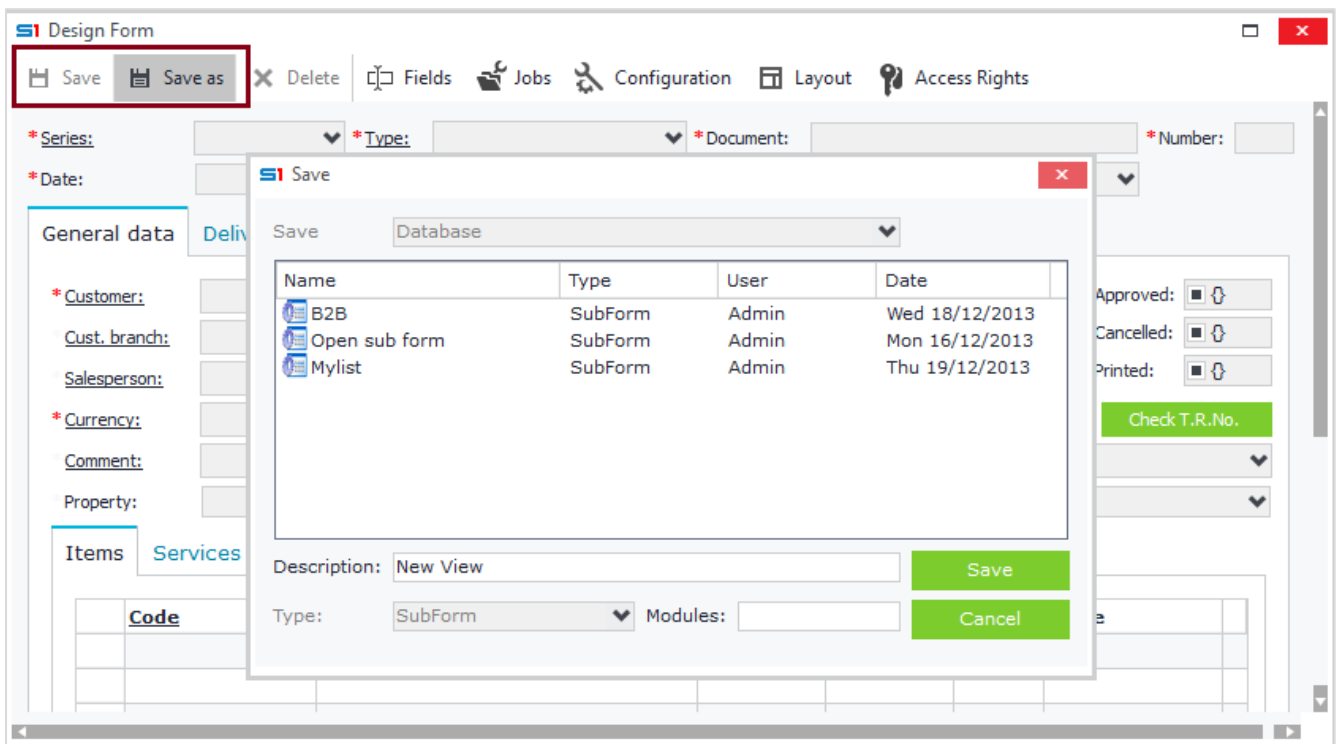


Figure A2

Notice that the design screen form is similar to runtime. The only difference is a toolbar that stays on top of the form and contains the tools to design it (Figure A3).



Figure A3

Form design is internally saved in a text format that can be displayed and copied to other forms through the shortcut keys: **CTRL + ALT + SHIFT + F12** (Figure A4).

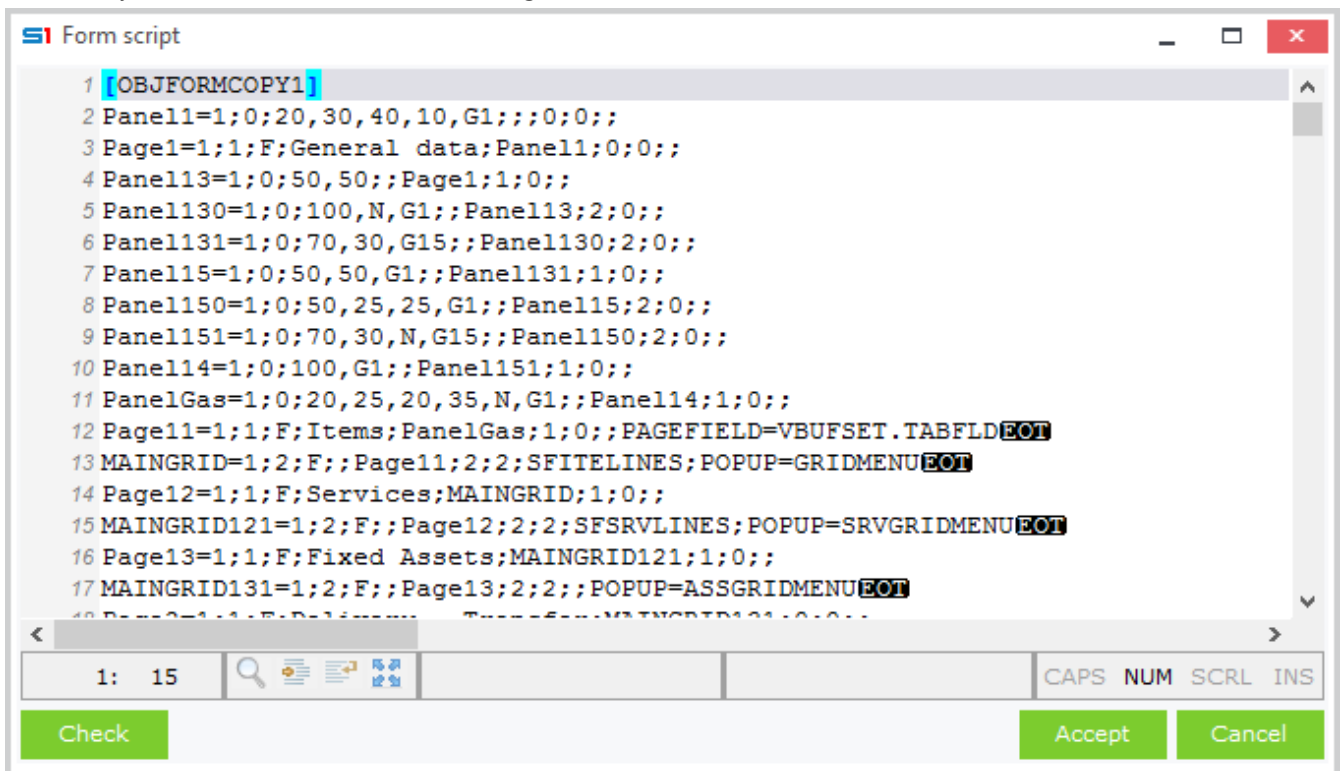


Figure A4

You can also view the object Info using the shortcut keys: **CTRL + ALT + SHIFT + F11** (Figure A5).

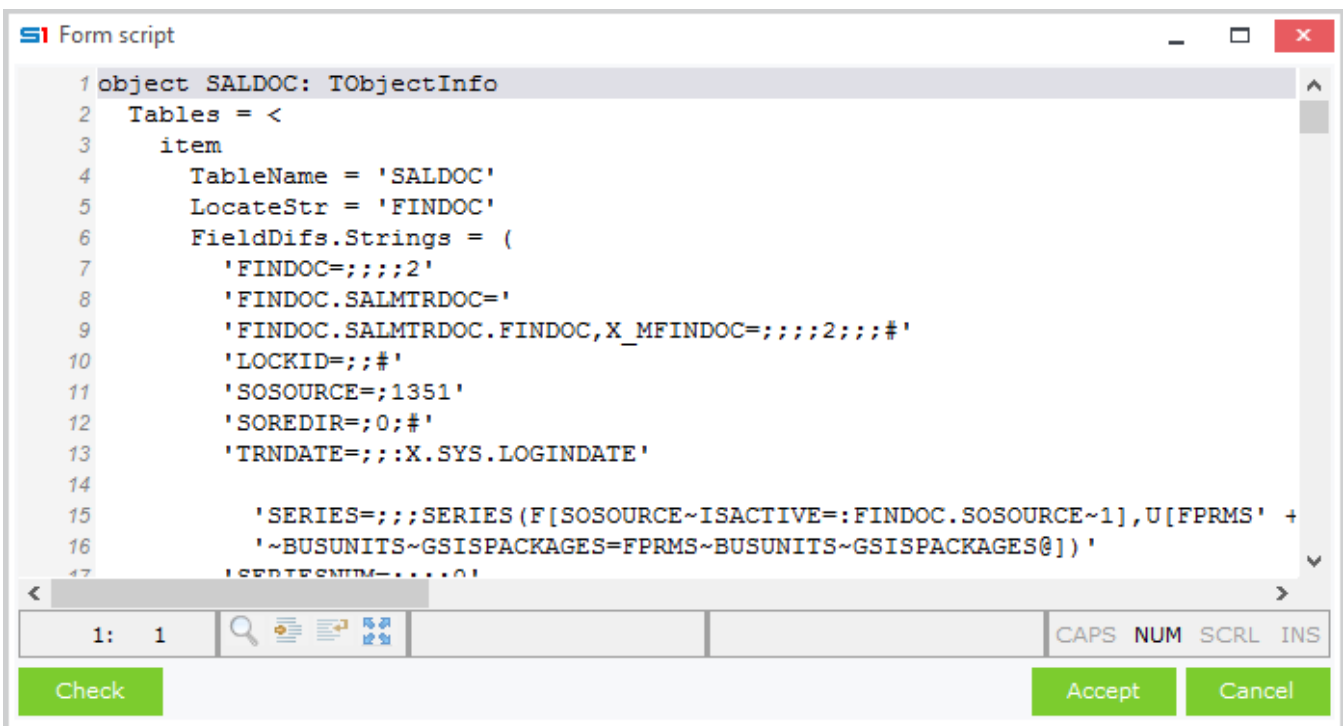


Figure A5

A.1 Basic Features

The design tools are displayed through the right click pop up menu, which displays the available options that can be used to create controls (panels, datagrids, tabs, memo textboxes and images) (Figure A1.1). The controls that can be added to a form are discussed in detail below.

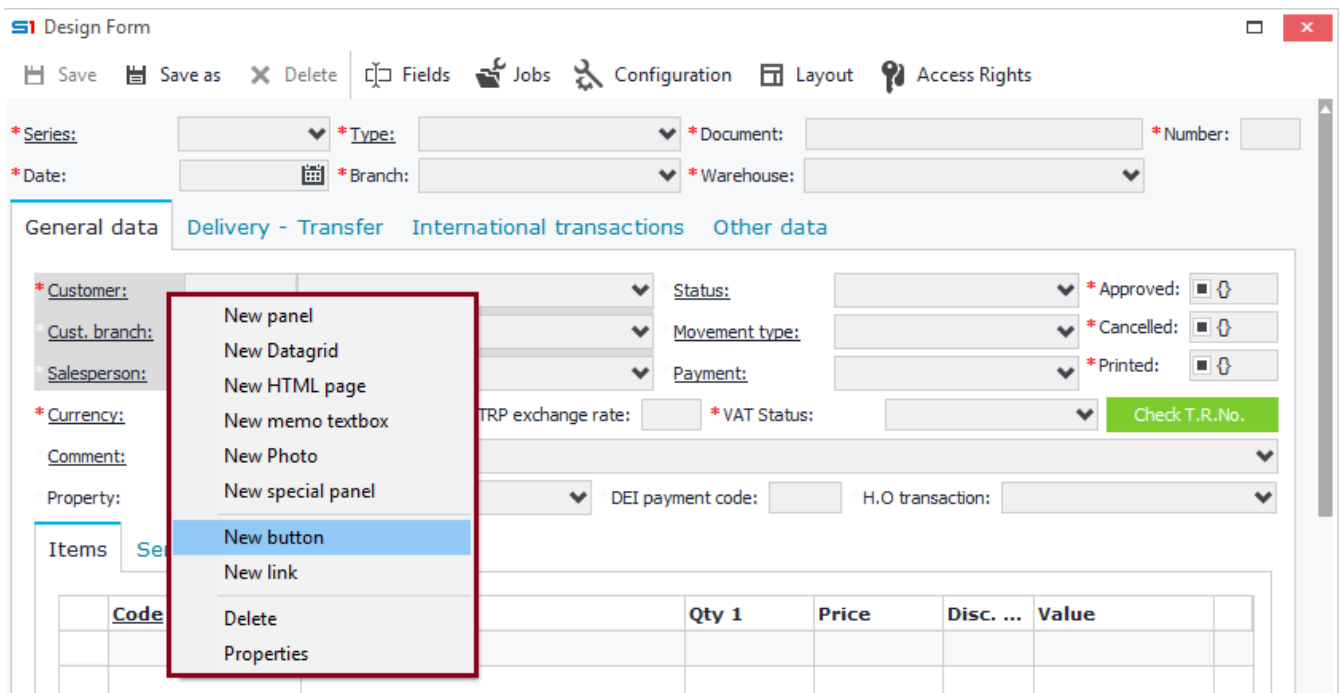


Figure A1.1

Form Design Options			
	Tab / Page	Panel / Area	Datagrid
New panel	Inserts a new panel in the selected tab.	Inserts a new panel under the selected panel.	Not applicable
New datagrid	Inserts a new datagrid in the selected tab.	Inserts a new datagrid under the selected panel.	Not applicable
New tab	Inserts a new tab on the left of the selected tab.	Inserts a new tab under the selected panel.	Inserts a new tab under the selected datagrid.
New memo textbox	Inserts a new memo textbox in the selected tab.	Inserts a new memo textbox under the selected panel.	Inserts a new memo textbox under the selected datagrid.
New Photo	Inserts a new image in the selected tab.	Inserts a new image textbox under the selected panel.	Inserts a new image textbox under the selected datagrid.
New button	Not applicable	Inserts a new button under the selected panel.	Not applicable
New link	Not applicable	Inserts a new link under the selected panel.	Not applicable
Delete	Deletes the selected tab.	Deletes the selected panel.	Deletes the selected datagrid.
Properties	Not applicable	Opens the panel properties window.	Opens the datagrid properties window

A.2 Fields

The toolbar on the top of the form window provides the tools for designing a form.

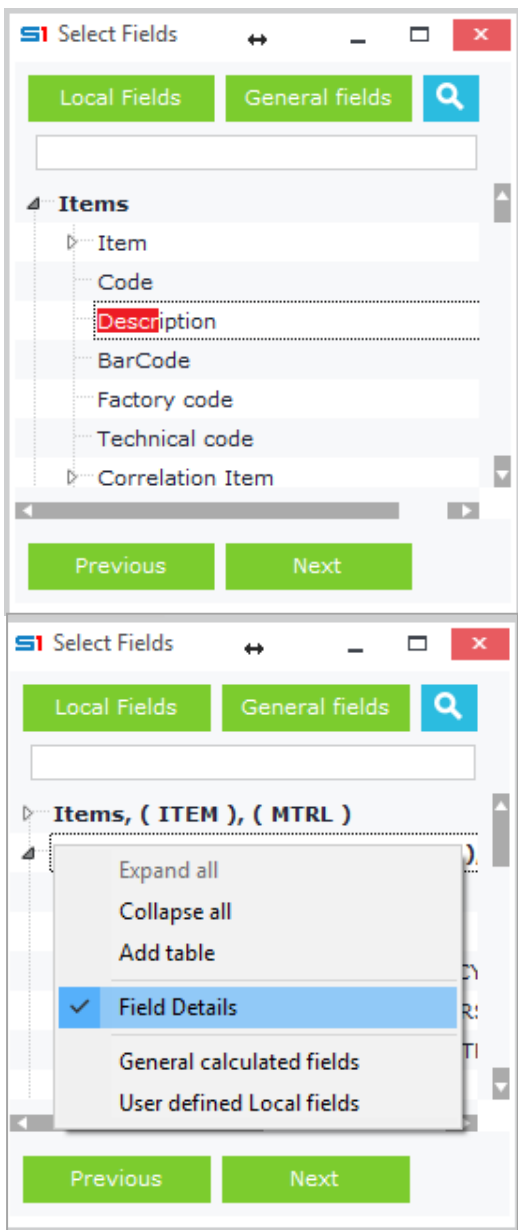
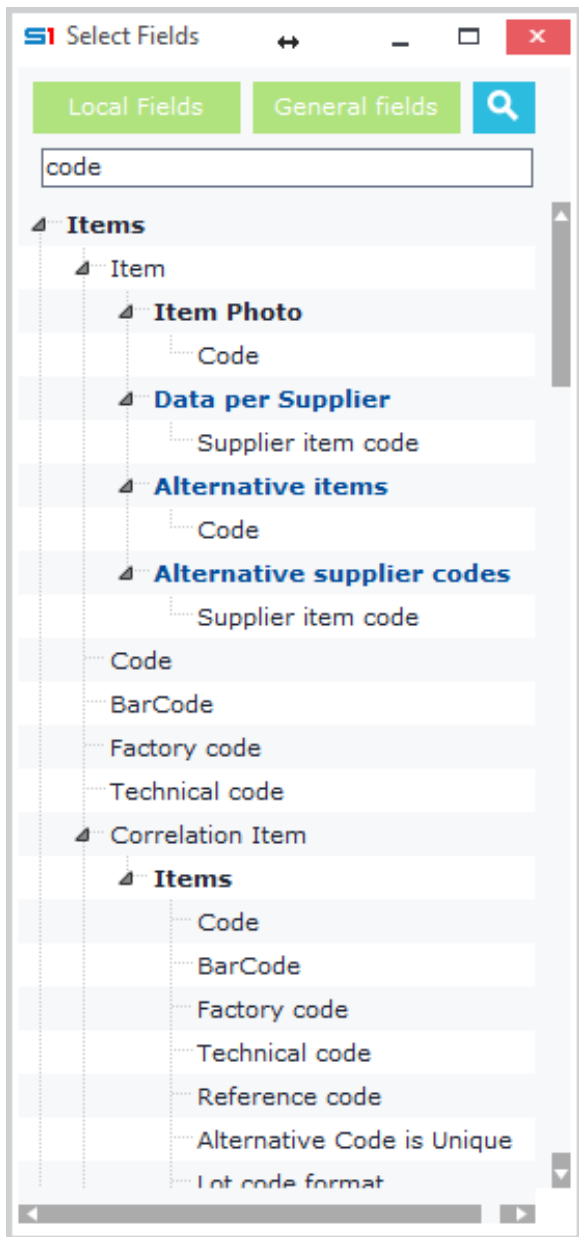


Button **"Fields"** displays a window that contains all the available tables and fields defined inside the object. Fields are inserted in panels or datagrids of the form through drag and drop.

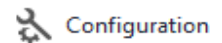
The search icon lets you search for fields names or captions that contain the search word.

Another way to search for a field is by typing inside the window. This searches for the field captions and displays the buttons *"Previous"* and *"Next"* at the bottom of the window.

Table names and field types are visible when you select the option *"Field Details"* from the right click menu.



A.3 Form Parameters / Settings



"**Configuration**" button displays a window where you can change the database objects (tables, views, virtual tables) that will be displayed in the form. The same tool allows you also to change field captions, default values, decimal places, and also the way the form will interact with users, by entering your own code in JavaScript.



The first tab "**General settings**" allows you to add module parameter commands. These commands are the same as the object properties and are explained fully in the section [Object Properties](#).

Figure A3.1 shows an example of the parameter command *BUFEXCLUDE*, which is used to exclude data from copy. In the example the field "NAME" from the table "ITEM" is excluded when users click on the toolbar button "Copy".

The command *CUSTOMBUFEXCLUDE* is the same as *BUFEXCLUDE*, but works for user defined fields. This command is automatically inserted when you enable the property "Exclude from Copy Buffer" in a user defined field. The third command *EDITOPTIONS=NOINSERT,NODELETE* hides the toolbar buttons "New" and "Delete" from the current form.

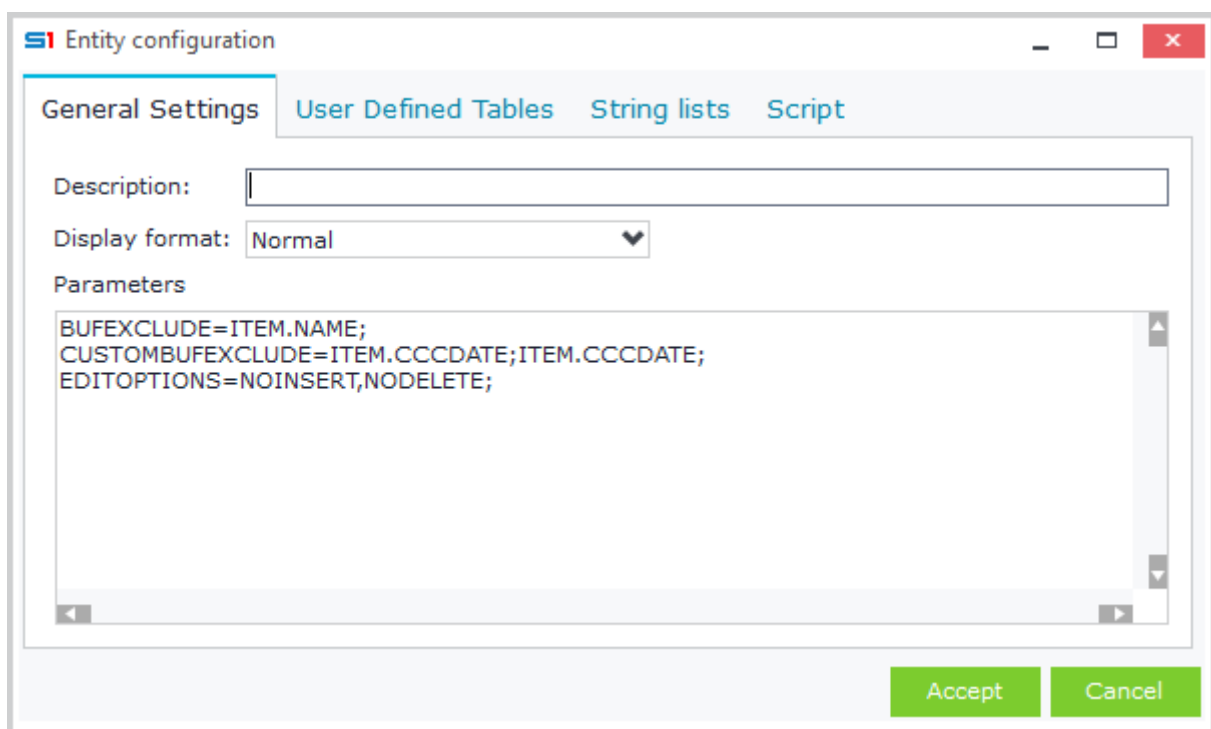


Figure A3.1 – Form Parameters

A.4 User Defined Tables

The second tab “User Defined Tables” contains the tables and fields of the current object. Use this tab to add or edit tables and also alter their field properties.

The screenshot shows the 'Entity configuration' window with the 'User Defined Tables' tab selected. On the left, there is a list of tables including 'Number 2', 'Number 3', 'Number 4', 'Yes/No 1', 'Table 1', 'Table 2', 'MY TRDR' (selected), 'Not processed', 'Document line value (no discount)', 'Altern.package', 'Item SN', 'Section of F.P.', 'Services: Lines', 'Fixed Asset lines', and 'Serial number per material line'. On the right, the field properties for 'MY TRDR' are displayed: Field name: CCCMYTRDR, Caption: MY TRDR, Display size: 12, Editor: CUSTOMER, Forced value: (empty), Default value: (empty), Decimals: (empty), Visible: checked, ReadOnly: unchecked, Required: unchecked, and No Copy Buffer: unchecked. At the bottom right, there are 'Accept' and 'Cancel' buttons.

Figure A4.1 – Field Properties

A4.1 Field Properties

Field properties are summarized in the table below.

PROPERTY	DESCRIPTION
Field name	Name of the field (Database Name) – ReadOnly
Caption	Title of the field, as it appears inside a form.
Display size	Width of the field. Applicable only in Datagrids.
Forced Value	Indicates the value that will be applied to the field when the record is submitted.
Default Value	Indicates the default value of the field, that applies to new entries. Users can change it at runtime.
Decimals	<ul style="list-style-type: none"> Indicates the number of the decimal places. Specific commands (P,V,Q,%,C) can be used in order to define the use of the company defaults for decimal places. When used in fields that are added to memo textboxes, then use the numbers 1, 2 or 3 to display the horizontal, vertical or both scrollbars. When used in fields that have memory tables as editor, then use number 0, 1, 2 to display the code, description or both values in datagrids.
Visible	Indicates whether the field is visible or not, when inserted to panel or datagrid in form.
ReadOnly	Specifies whether the contents of the field can change at runtime by users.
Required	Specifies whether a value for the field is required in order to submit a record (e.g. sales lines with null items are not permitted).
No Copy Buffer	Visible only in custom fields. If selected then the value of this field is not copied, when a record is copied (toolbar button “Copy”).

A4.2 Add / Delete Tables

The combo box “Tables” lists all the custom tables, virtual tables and views that can be added in the form.

In order to add a SoftOne table in this list, click on the button “Add table” (Figure A4.2) and double click on the desired table.

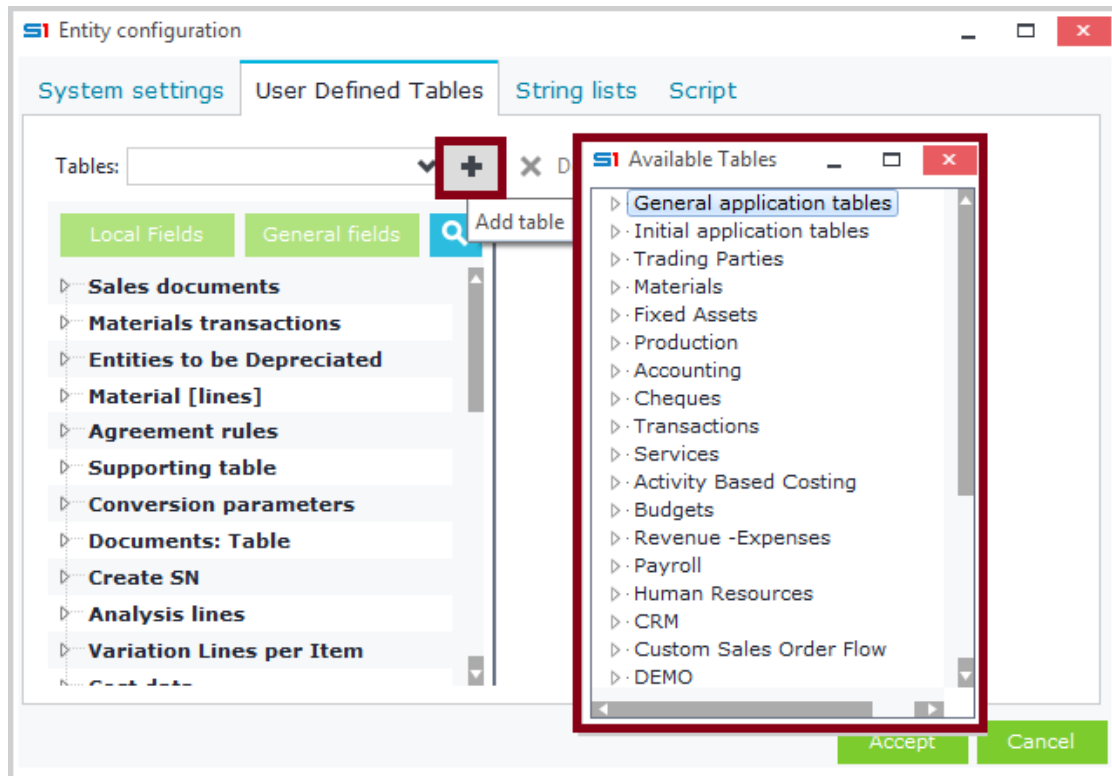


Figure A4.2

Any custom or internal table that is added can be deleted using the button “Delete table”.

Tables that are linked to fields (through editor property) can be deleted using the right click option “Delete link...” (Figure A4.3).

Notice that linked tables will not be deleted if they have fields inside the form layout.

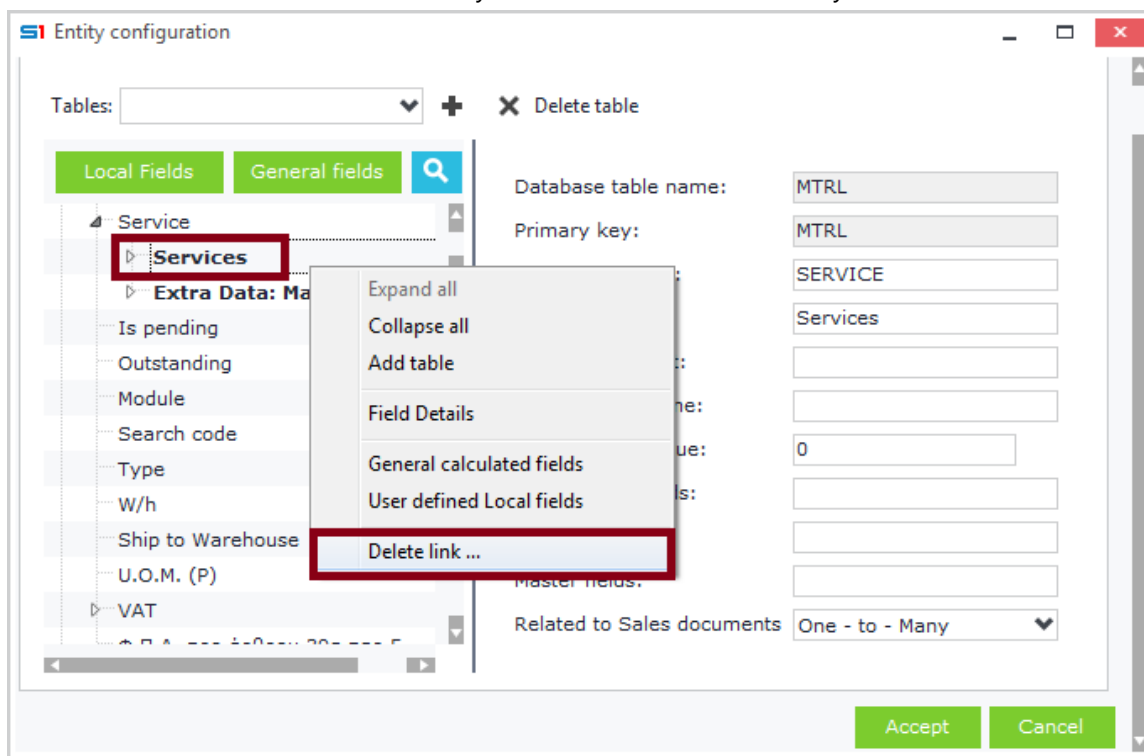


Figure A4.3

A4.3 Table Properties

The available properties of the selected table (left pane) are displayed in the right pane of the window (Figure A4.4).

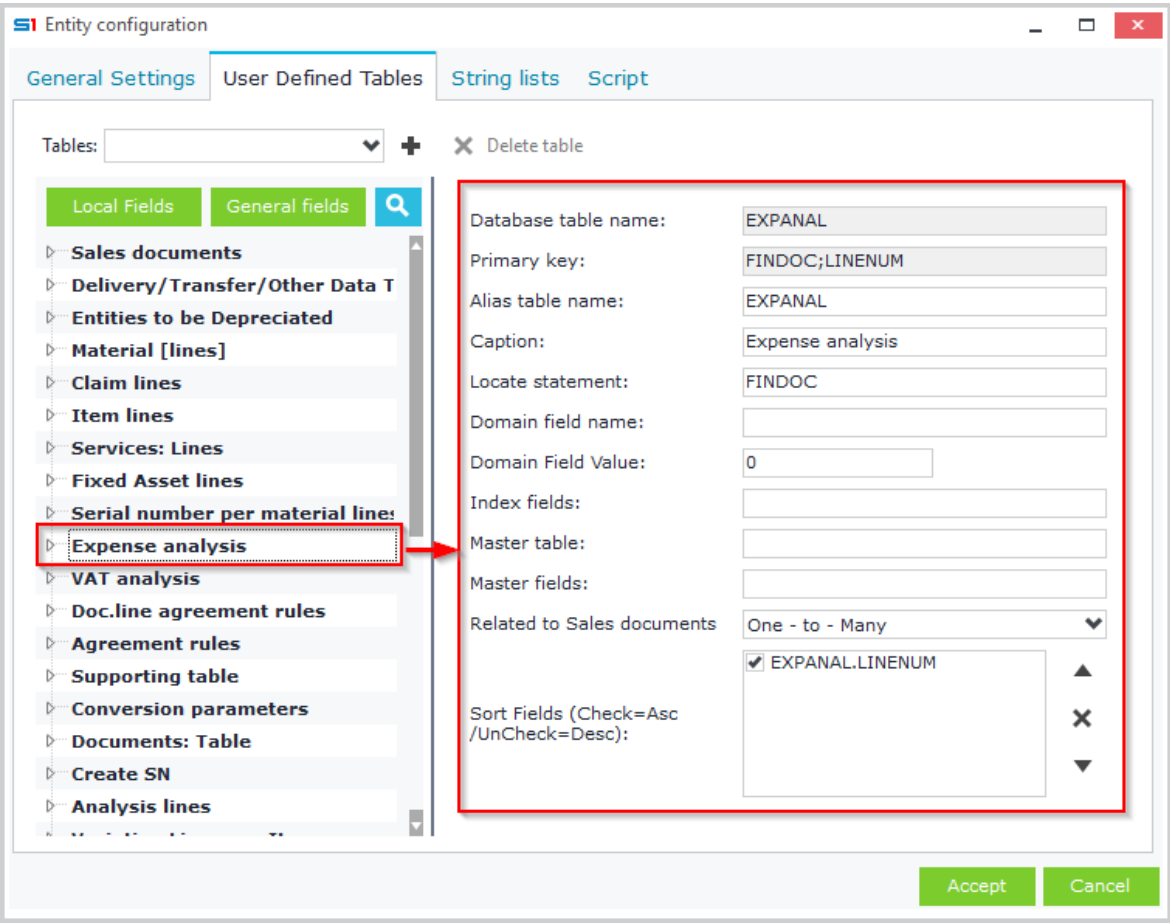


Figure A4.4

Database table name

Displays the name of the selected table or view as it exists in the database. It also displays the name of custom virtual tables or view tables as defined in SoftOne database designer.

Primary key

Displays the primary key of the selected table (Figure A4.5).

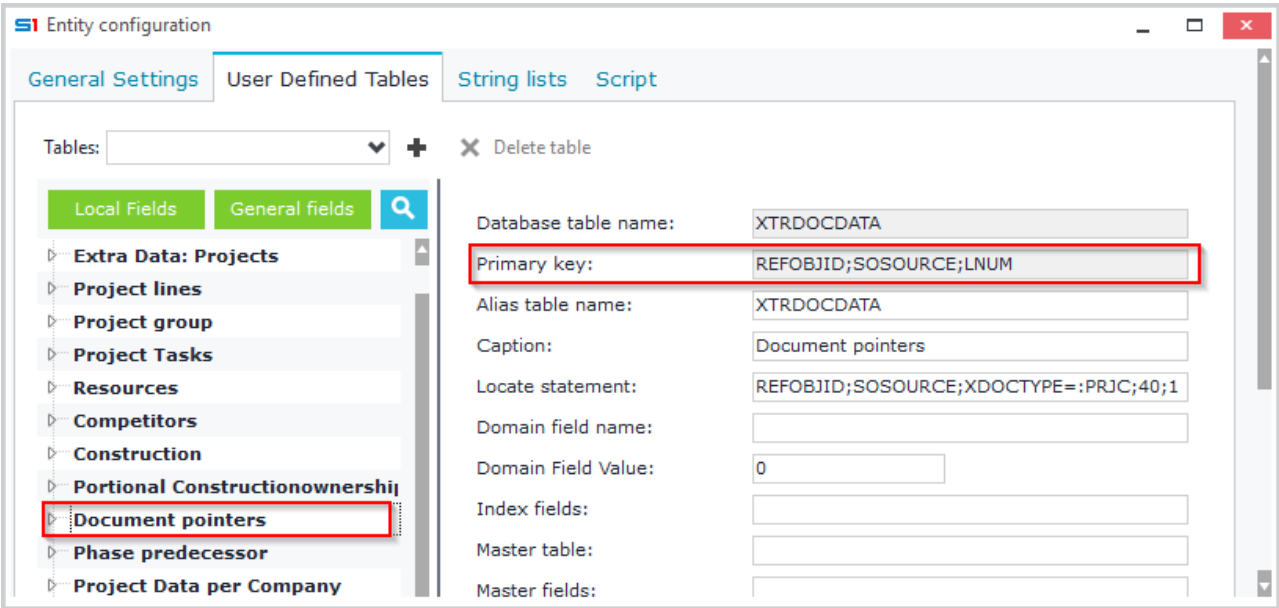


Figure A4.5

Alias table name

Defines the alias name of the table. Alias names are used to refer to tables in form scripts. It is possible to use the same table more than once inside a form as long as you have different alias names. Figure A4.6 shows that the database table PRJLINES is inserted many times inside form PRJC, given different alias table names (PRSNLINES, STGLINES, etc).

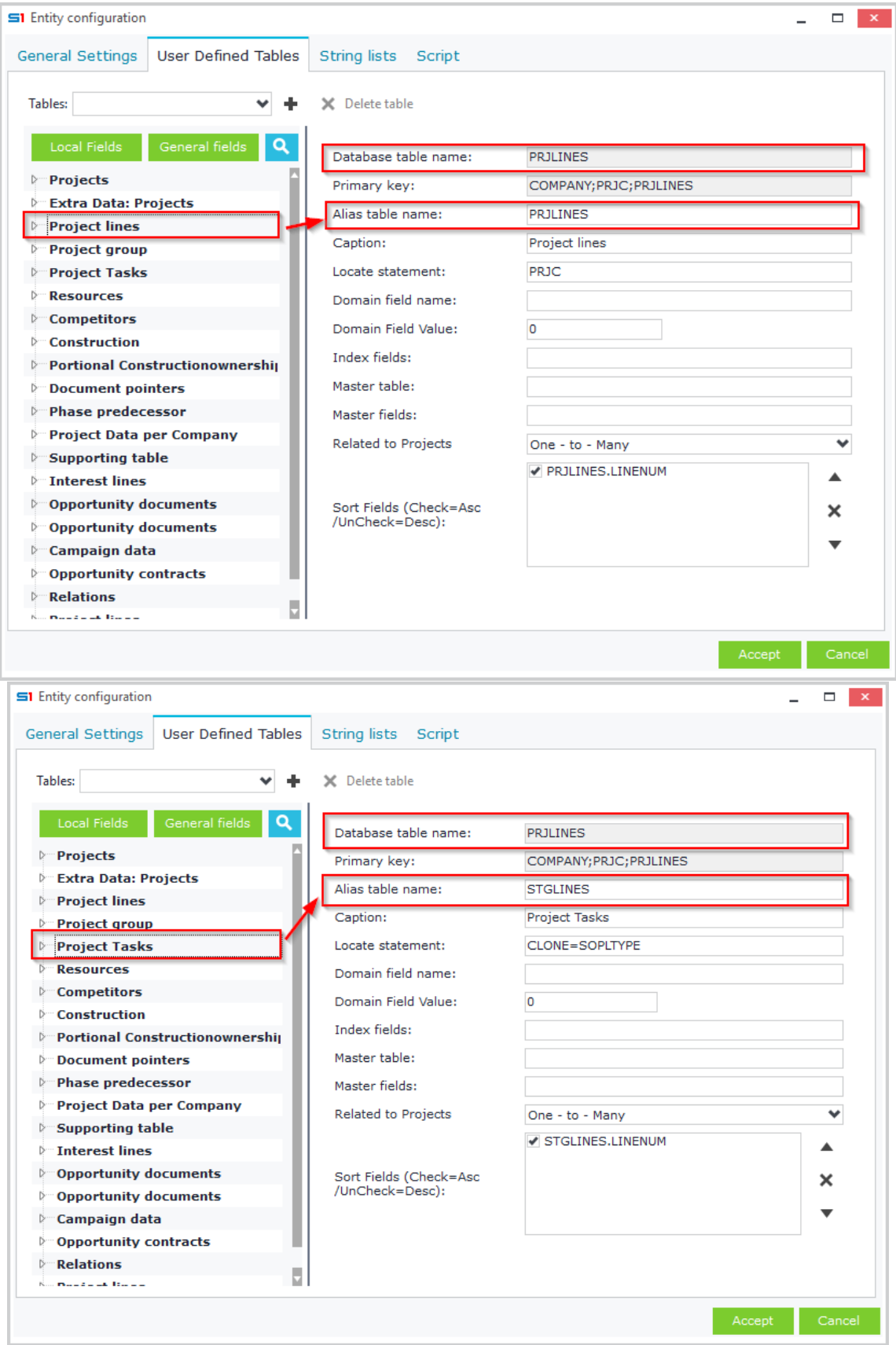


Figure A4.6

Locate statement

Defines the fields that will be used to locate the data of the selected table according to the parent table.
If the the primary key of the detail table has the same name with the parent one (e.g. FINDOC), then you only need to enter its name inside the locate statement (e.g. FINDOC). The application will auto create the relation **ChildTable.ID = :HeaderTable.ID** (e.g. MTRLINES.FINDOC=:FINDOC.FINDOC) (Figure A4.7).

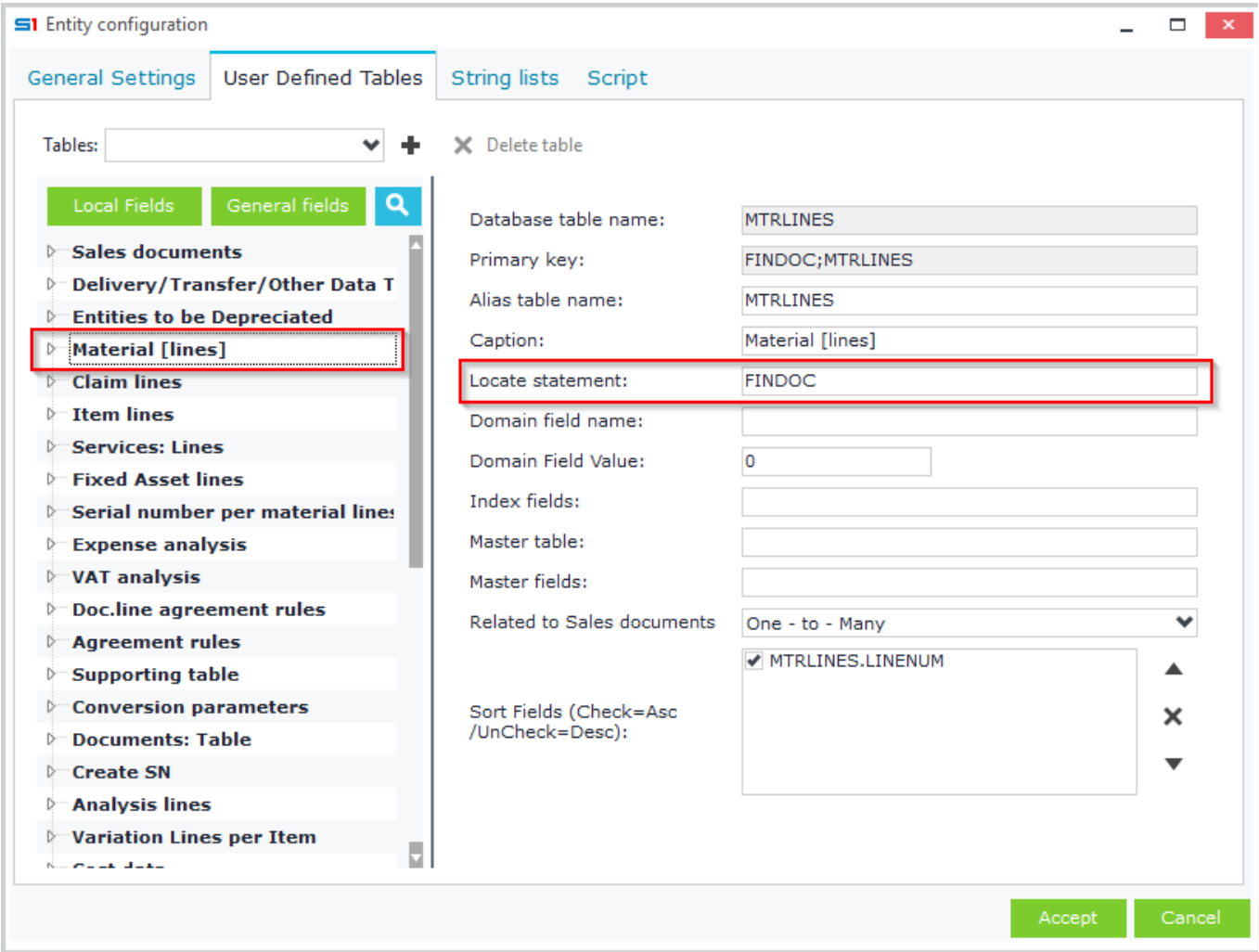


Figure A4.7

More that one keys are defined using a key-value pair expression divided by semicolons as follows:

```
MyTableField1;MyTableField2;MyTableField3=1000;2000;3000
```

References to parent table field values are made using colons followed by fields (without table names) (Figure A4.8):

```
MyTableField1;MyTableField2;MyTableField3=:ParentField1::ParentField2;1000
```

SoftOne X.SYS parameters can also be used as values inside locate statements (Figure A4.9)

```
MyTableField1;MyTableField2;MyTableField3=:ParentField1::X.SYS.COMPANY;1000
```

The screenshot shows the 'Entity configuration' window with the 'User Defined Tables' tab selected. On the left, a tree view lists various entities, with 'Document pointers' highlighted. The main area displays the configuration for the 'XTRDOCDATA' table. The 'Locate statement' field is highlighted with a red box and contains the text: `REFOBJID;SOSOURCE;XDOCTYPE=:PRJC;40;1`.

Field	Value
Database table name:	XTRDOCDATA
Primary key:	REFOBJID;SOSOURCE;LNUM
Alias table name:	XTRDOCDATA
Caption:	Document pointers
Locate statement:	REFOBJID;SOSOURCE;XDOCTYPE=:PRJC;40;1
Domain field name:	
Domain Field Value:	0
Index fields:	
Master table:	
Master fields:	

Figure A4.8

The screenshot shows the 'Entity configuration' window with the 'User Defined Tables' tab selected. On the left, a tree view lists various entities, with 'Data per Supplier' highlighted. The main area displays the configuration for the 'MTRSUPPRCS' table. The 'Locate statement' field is highlighted with a red box and contains the text: `COMPANY;MTRL=:X.SYS.COMPANY;:MTRL`.

Field	Value
Database table name:	MTRSUPPRCS
Primary key:	MTRL;COMPANY;TRDR;SOCURRENCY
Alias table name:	MTRSUPPRCS
Caption:	Data per Supplier
Locate statement:	COMPANY;MTRL=:X.SYS.COMPANY;:MTRL
Domain field name:	
Domain Field Value:	0
Index / Sort fields:	
Master table:	
Master fields:	
Related to Items	One - to - Many

Buttons: Accept, Cancel

Figure A4.9

Index Fields

Used in second child tables (child tables linked to child tables) in order to indicate the fields that define the child to child relationship. Figure A4.10 shows the table SNLINES which is child table to MTRLINES, which is also child table of the parent table FINDOC. Index field of the SNLINES table is the field MTRLINES.

Master Table

Used in second child tables to indicate the name of the master child table that will be used for locating the records of the second child table.

Master Fields

Used in second child tables to indicate the fields of the master child table that define the child to child relationship.

Entity configuration

General Settings | User Defined Tables | String lists | Script

Tables: + X Delete table

Local Fields | General fields

Sales documents
Delivery/Transfer/Other Data T
Entities to be Depreciated
Material [lines]
Claim lines
Item lines
Services: Lines
Fixed Asset lines
Serial number per material lines
Expense analysis
VAT analysis
Doc.line agreement rules
Agreement rules
Supporting table
Conversion parameters
Documents: Table
Create SN
Analysis lines
Variation Lines per Item

Database table name: SNLINES
Primary key: FINDOC;MTRLINES;SNLINES
Alias table name: SNLINES
Caption: Serial number per material lines
Locate statement: FINDOC
Domain field name:
Domain Field Value: 0
Index fields: MTRLINES
Master table: ITELINES
Master fields: MTRLINES
Related to Sales documents: One - to - Many
Sort Fields (Check=Asc /UnCheck=Desc):
SNLINES.SNLINES
Accept Cancel

Figure A4.10

Sort Fields

Defines the default sort fields of the selected table (when used in grid).

Fields are added in the “**Sort Fields**” list using drag and drop from the “**Fields**” list (Figure A4.11).

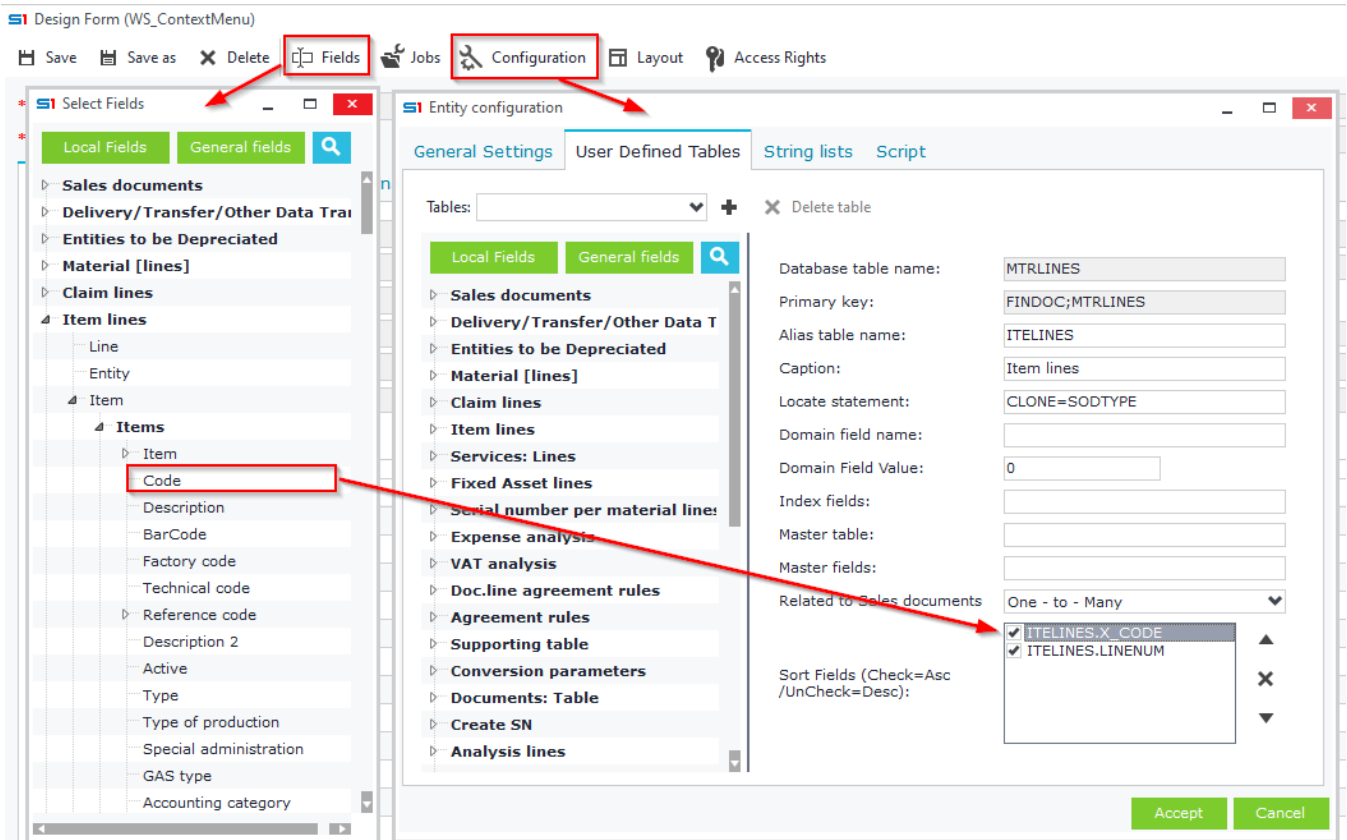


Figure A4.11

Uncheck a field if you need to sort it in descending order (Figure A4.12)

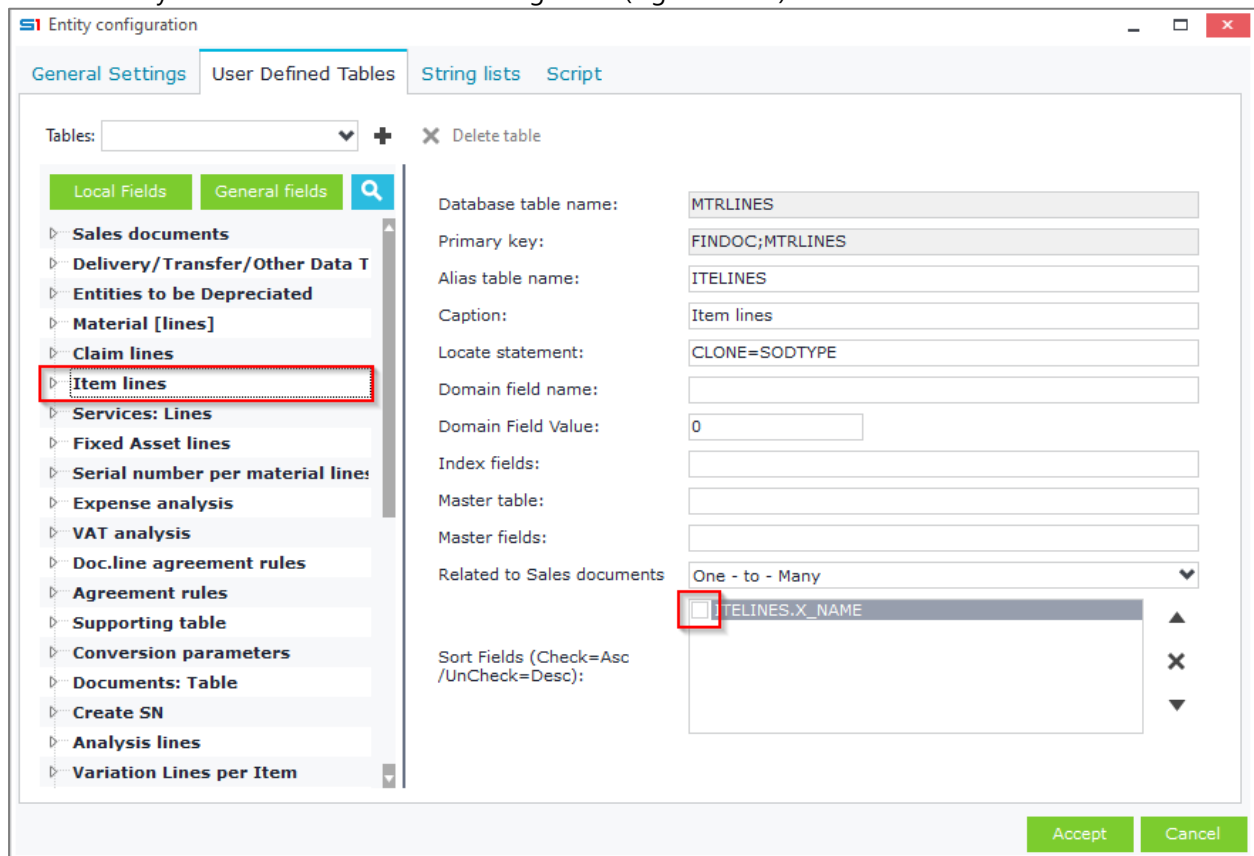


Figure A4.12

A.5 String lists

This tab gives you the option of creating string lists that can be used inside the form. String Lists display a collection of keys and values, that can be used as editors to numeric fields and display data in combo box controls with fast lookups (Figure A5.1).

The screenshot shows a 'New entry' form with various fields and tabs. A dropdown menu is open for the 'Integer 01' field, displaying three options: 'Value 1', 'Value 2', and 'Value 3'. The 'Value 2' option is currently selected. The form includes fields for Series, Type, Document, Number, Date, Branch, Warehouse, Customer, Cust. branch, Salesperson, Payment, Currency, Exchange rate, TRP exchange rate, Exemption note, VAT Status, and Comment. There are also tabs for General data, Delivery transfer, International transactions, and Other data. At the bottom, there are tabs for Items, Services, and Fixed assets.

Figure A5.1

A string list is created using the right click option “New” on the left panel and its values are inserted through the grid in the right panel (Figure A5.2).

The screenshot shows the 'Entity configuration' window with the 'String lists' tab selected. On the left, there is a tree view showing 'MyList'. A context menu is open over 'MyList' with options 'New', 'Modify', and 'Delete'. On the right, there is a table with two columns: 'Key' and 'Value'. The table contains three rows of data: Key 1 with Value 1, Key 2 with Value 2, and Key 3 with Value 3. At the bottom of the window, there are 'Accept' and 'Cancel' buttons.

Key	Value
1	Value 1
2	Value 2
3	Value 3

Figure 5.2

A string list can be used as lookup list of numeric fields by adding its name in the editor property of the field (Figure A5.3).

Note that string lists created through SoftOne Designer are referenced with a \$ prefix (e.g. **\$DesignerStringListName**).

Analysis of String Lists design can be found in section [String Lists](#) (Chapter Form Scripts).

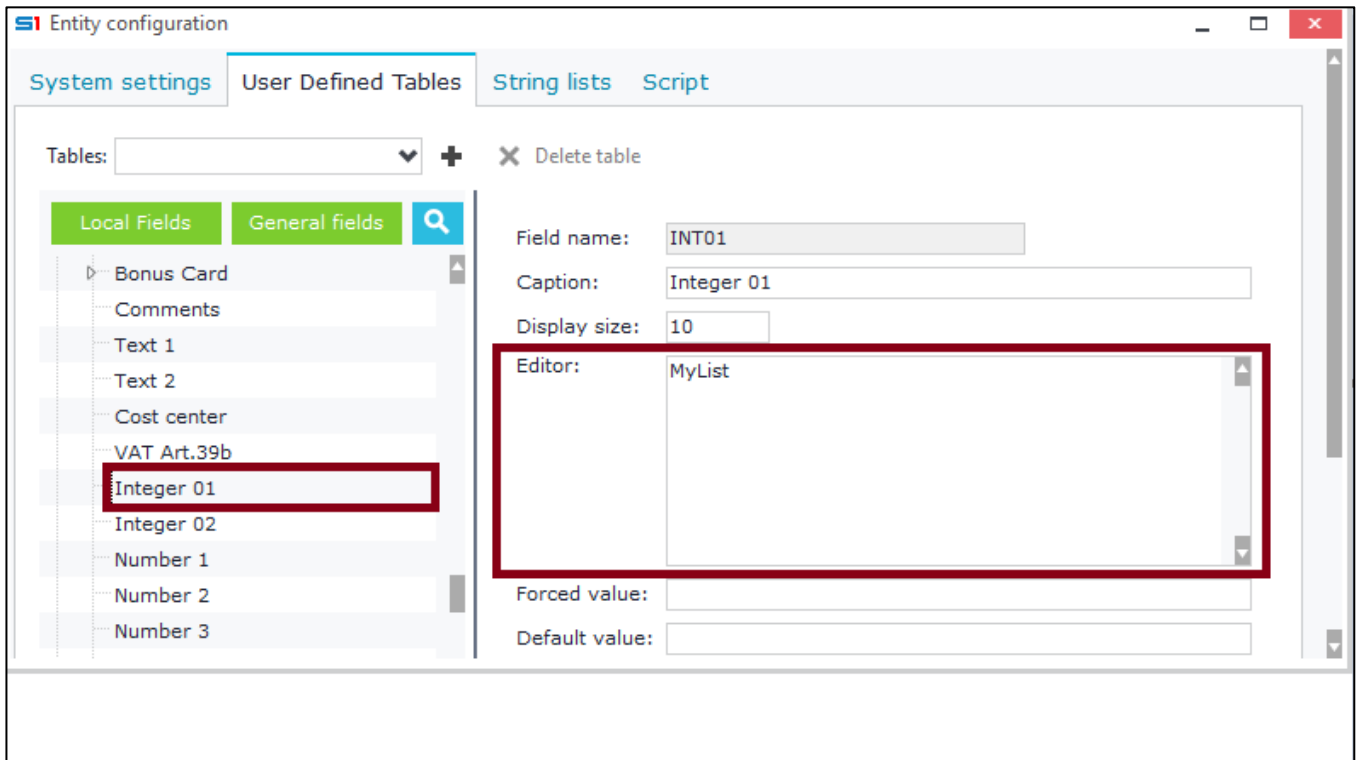


Figure A5.3

A.6 Script

Tab “Script” is used for adding extra code in JavaScript or VBscript that implements functionality in the form. The default script Language is Javascript. Button “Check” (bottom-left) can be used to check Javascript code for syntax errors.

The available commands, functions and events that can be used inside a form script are covered in Chapter [Form Scripts](#).

Note: When you press **CTRL+Spacebar** after entering “X.”, “function”, “DatasetName.” (e.g. ITELINES.) then a list of the available commands is displayed as in Figure A6.1.

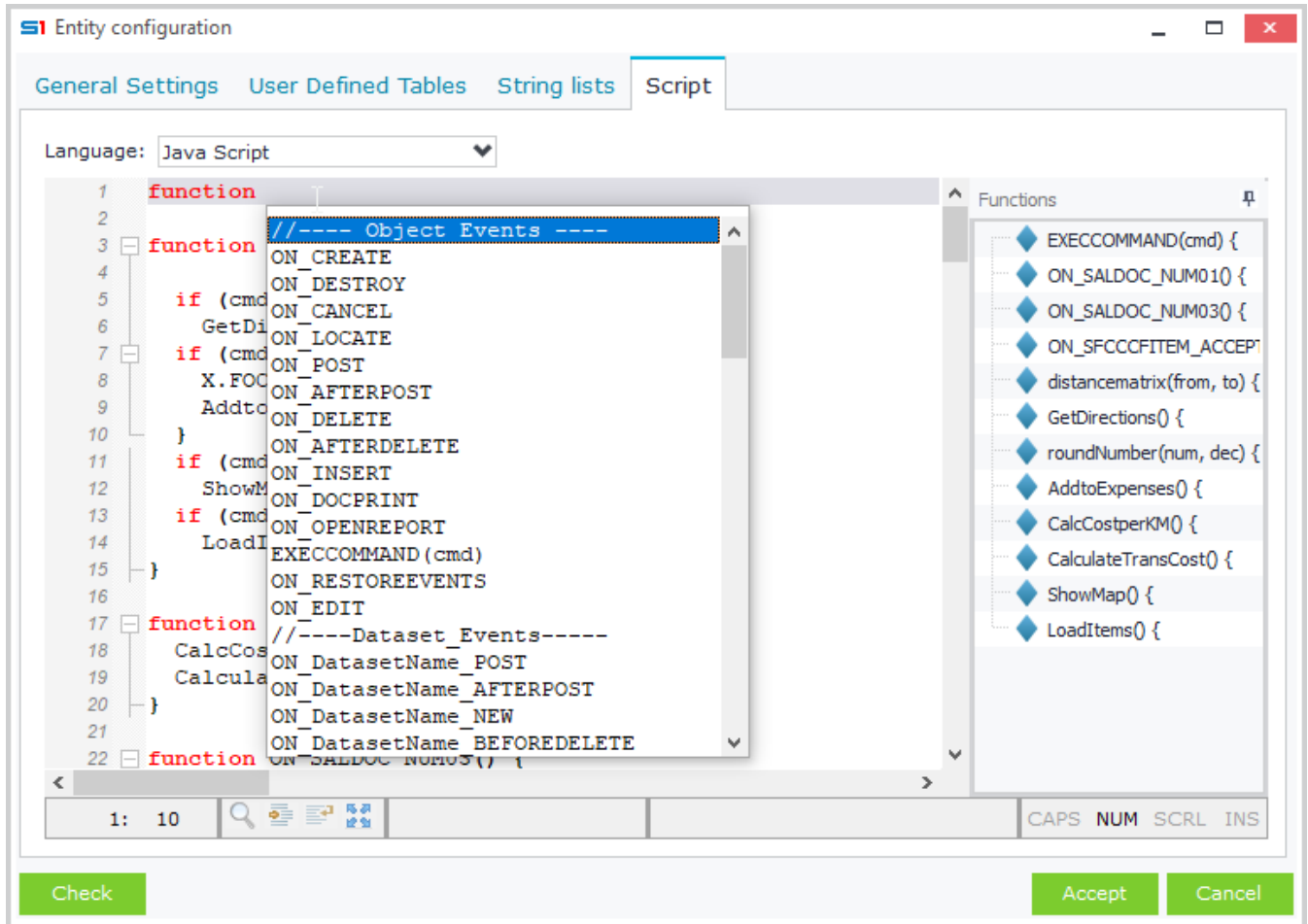


Figure A6.1

A.7 Layout

The toolbar button “Layout” displays all the form controls structure in a tree view. This tool has all the controls that are used inside the form as well as the sub forms. It also provides the same tools as the designer window, which means that you can design the whole form through this tool. Use the right click context menu to display all the available commands for designing the form through this tool (Figure A7.1). The hierarchical arrangement of the controls in the form is done through this tool. Simply drag a control (tab, panel, e.t.c) from a lower node and drop it to the preferred higher node. This way your control is moved to a new position inside the form (Figure A7.2).

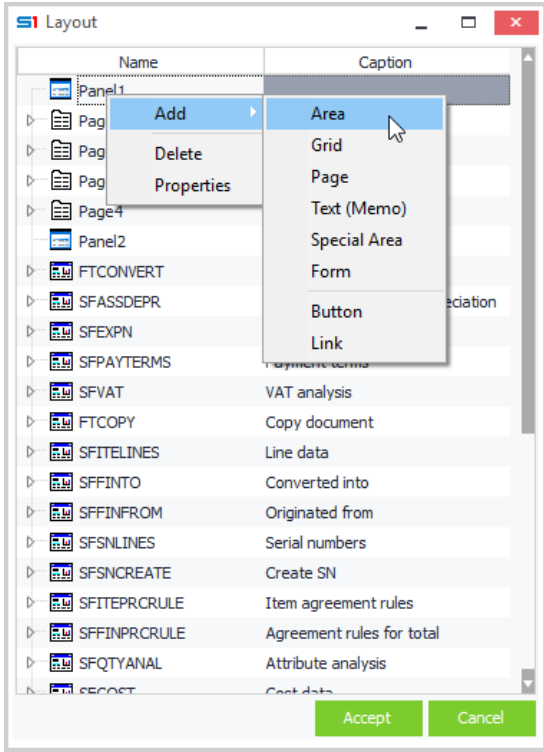


Figure A7.1

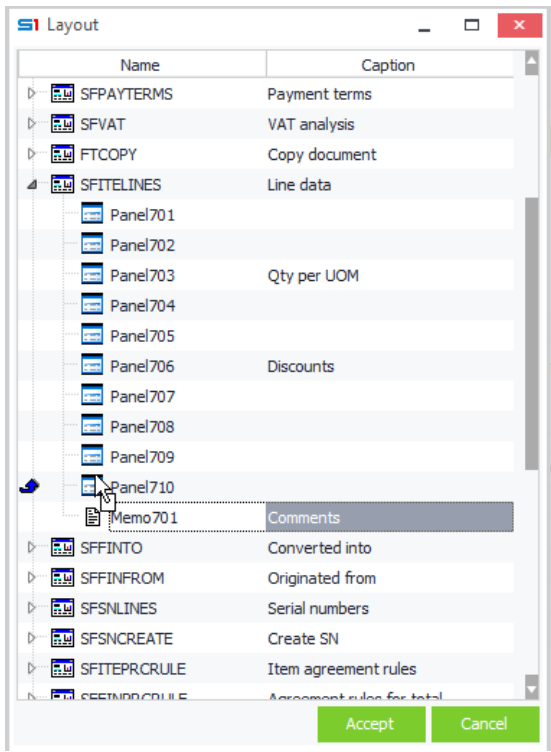


Figure A7.2

Nested / horizontal panels (Figure A7.3) are also created through this tool, by creating a parent panel and then creating the child (horizontal) panels inside it (See [Nested Panels](#)).

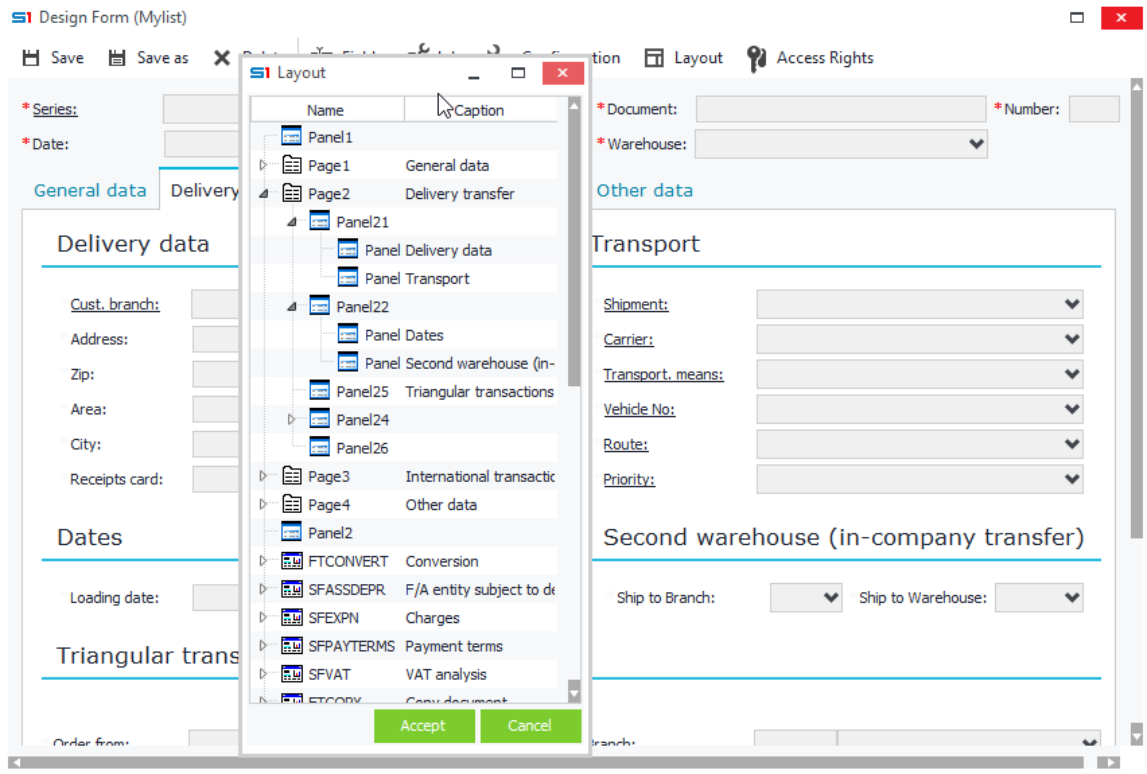


Figure A7.3

A.8 Access Rights

Users or groups of users that can access the form are defined through this window (Figure A8.1)

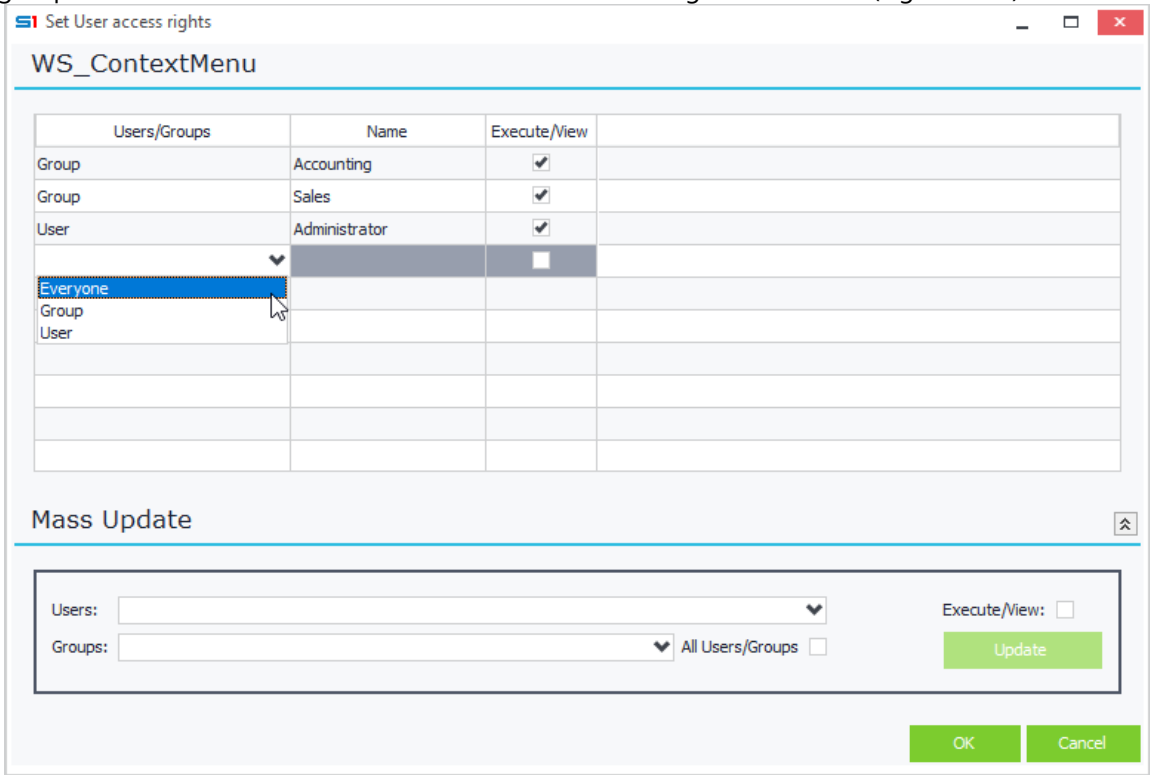


Figure A8.1

Record access rights are defined through object button “Define User Access Rights”. The available options are (A8.2):

- **Execute / View:** Access (Open) current object
- **New:** Insert new records
- **Edit:** Edit records
- **Delete:** Delete records

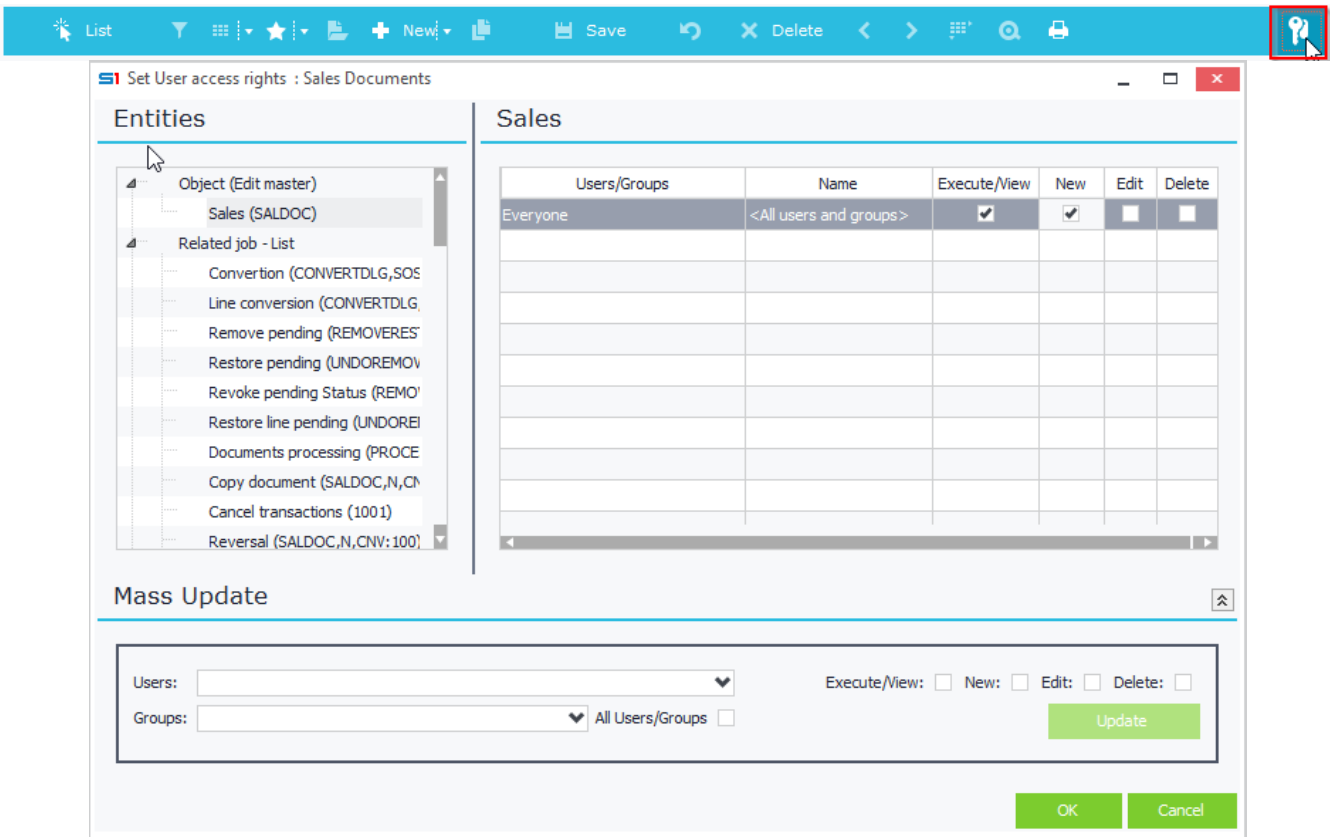


Figure A8.2

B. Layout Controls

B.1 Tabs / Pages

Tab control provides a way to present related information on separate labeled tabs (pages) within the form. By using the Tab control, you can define multiple tabs in a SoftOne object form. Each page consists of a group of controls that an object view displays when the user selects the corresponding tab. The following image displays an example of tabs inside the Items object form (Figure B1.1).

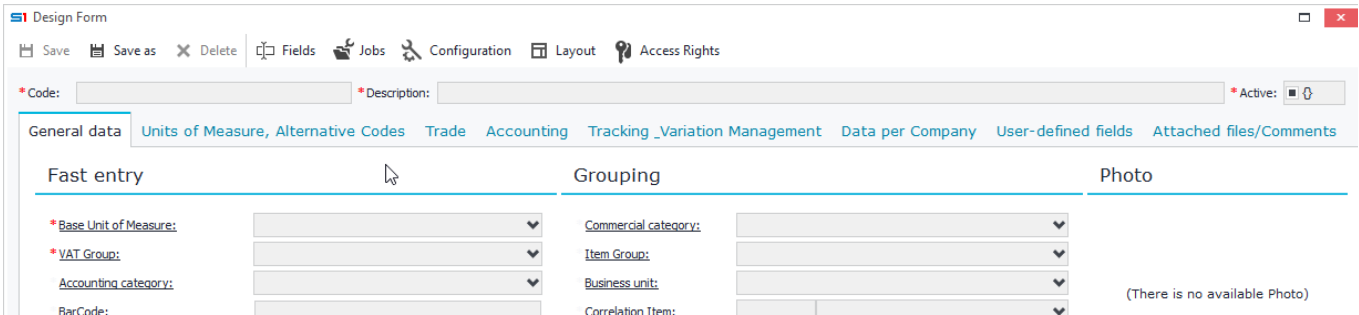


Figure B1.1

Tabs can be created in forms design mode using the option “New page” of the right click pop up menu or from the layout window (Figure B1.2). The order of tabs in a form can be altered using the “Layout” button. Alter the position of the tabs by performing a drag and drop action.

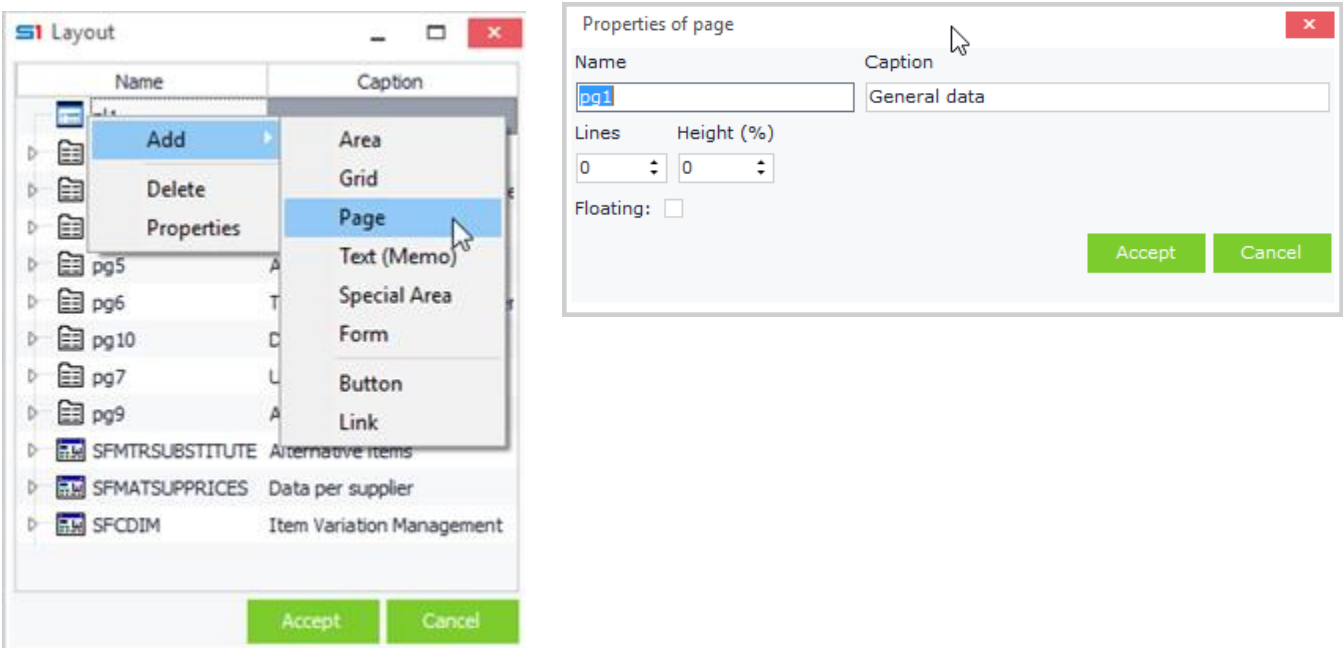


Figure B1.2

PROPERTY	DESCRIPTION
Name	Sets the name of the tab control (used in Form Scripts).
Caption	Sets the text that appears as title of the tab.
Lines	Sets the height of the tab counted in number of lines. Zero value auto increases the height of the tab in order to display all the panel controls.
Height	Sets the percentage height of the panel, according to its main layout control.
Floating	Auto fits the tab to fill the available space of the form. Works for tabs inserted at the bottom of forms or tabs.

B.2 Panels

Forms and tabs consist of panels that contain controls, which are either designed for data entry such as textboxes, combo boxes and checkboxes or for invoking commands within the application, such as buttons and links. Figure B2.1 displays the Items form in design mode.

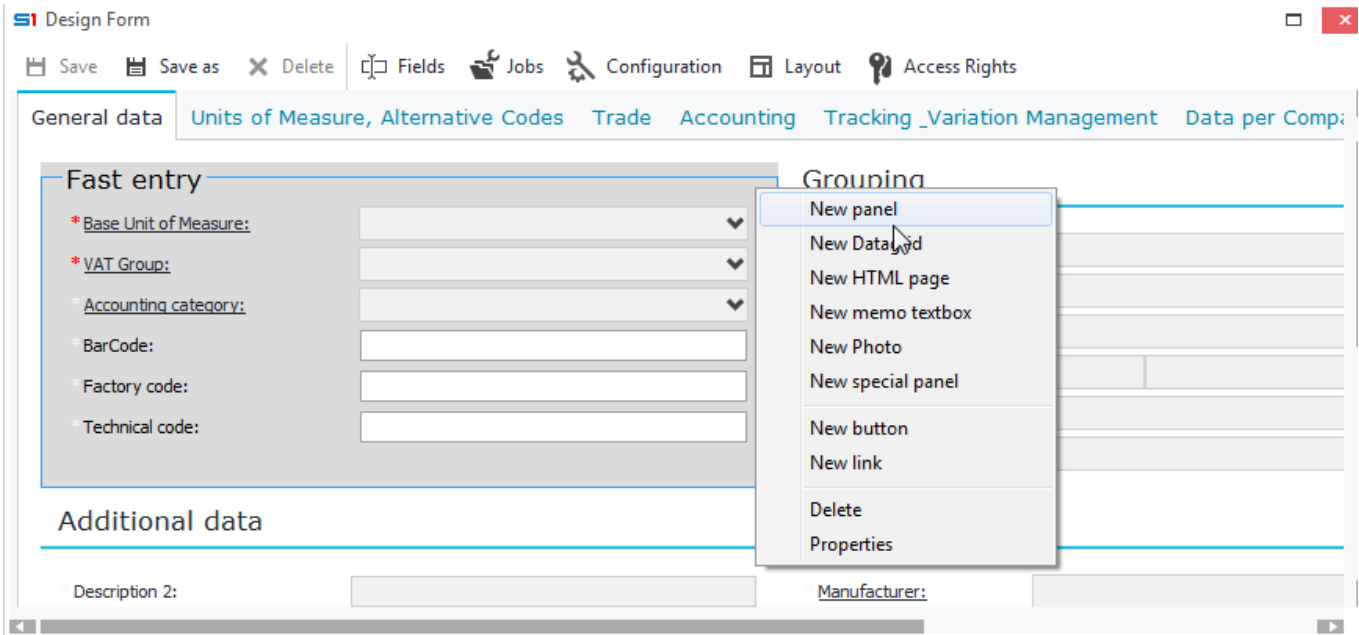


Figure B2.1

Design

Panels are created by right clicking on a form or tab and selecting “New Panel”. Fields can be added into panels through drag and drop action either from the “Fields” window (Figure B2.2) or through “Configuration” window. Panels can display up to four fields (columns) in a row, but you can use more than one panels (nested panels) in order to display more than four fields in a row.

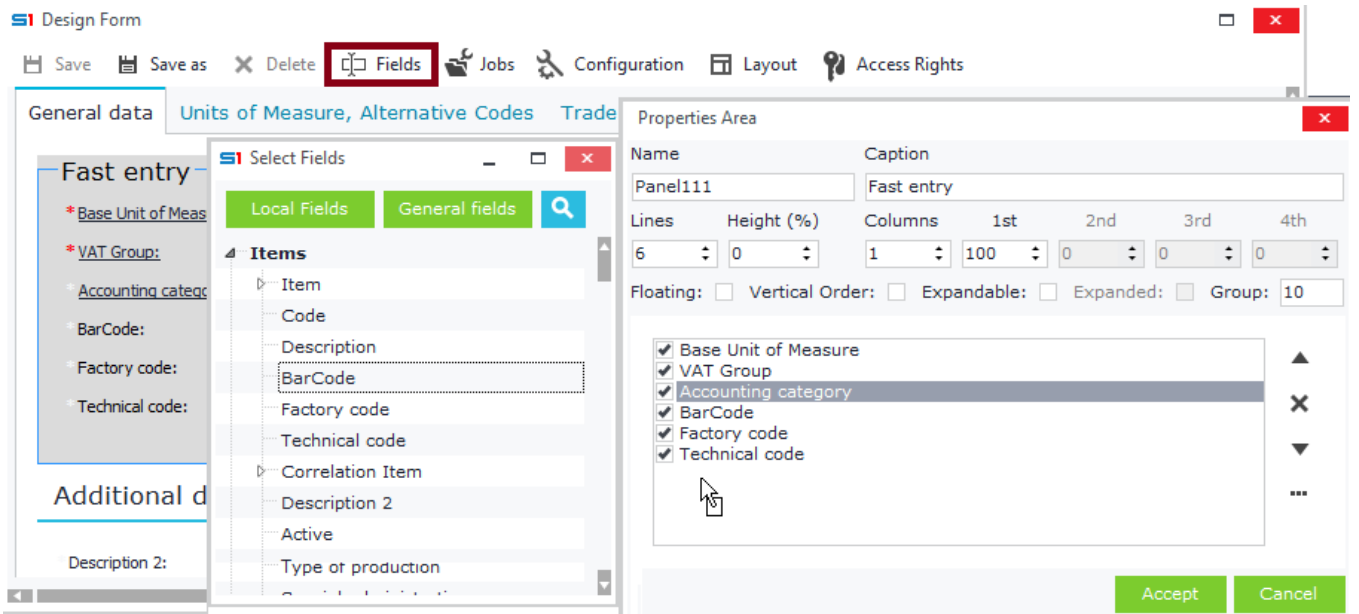


Figure B2.2

Properties

The properties of a panel enable you to alter its appearance in the form. They are almost the same as other form controls, since most of them share similar properties. The table below lists panel properties which are the same for nested panels.

PROPERTY	DESCRIPTION
Name	Sets the name of the panel control.
Caption	Sets the text that appears as title of the panel.
Lines	Sets the height of the panel counted in number of lines. Zero value auto increases the height in order to display all the panel controls.
Height	Sets the percentage height of the panel, according to its main layout control.
Columns	Sets the number of displayed columns (Max=4). Splits data controls into horizontal columns. It is also used in nested panels and defines the number of horizontal panels contained in a master panel.
Columns percentage	Percentage of displayed columns, which is relevant to the width of the panel
Floating	Auto fits the panel to fill the available space of a form or tab. Works for panels inserted at the bottom of forms or tabs.
No bottom line	Hides bottom border line of panel.
Vertical order	Vertical order of the panel controls. It also sets the control tab order to vertical instead of horizontal
Group	Indicates a number that defines the selected panel. All panels that have the same number align their controls in the same way.
Fields	Fields, buttons and links that will be displayed in the selected panel.

B.3 Nested Panels

Object forms and tabs can display panels horizontally, which means that you can more than one panels, containing up to four fields each, in a horizontal layout. A typical example of nested panels is the items form (Figure B3.1).

The screenshot shows the 'Design Form' application interface. The main window has a title bar 'Design Form' and a menu bar with 'Save', 'Save as', 'Delete', 'Fields', 'Jobs', 'Configuration', 'Layout', and 'Access Rights'. The form is divided into several sections:

- Fast entry**: Contains fields for 'Base Unit of Measure', 'VAT Group', 'Accounting category', 'BarCode', 'Factory code', and 'Technical code'.
- Additional data**: Contains fields for 'Description 2', 'Web page', 'Invoice Warning', 'Substituted by', and 'Sold only to'.
- Grouping**: Contains fields for 'Commercial category', 'Item Group', 'Business unit', 'Correlation Item', 'Season', and 'Commission category'.
- Allocation data**: Contains fields for 'Manufacturer', 'Brand', 'Model', 'Country of origin', and 'Made in'.

A context menu is open over the 'Grouping' panel, showing the following options: 'New panel', 'New Datagrid', 'New HTML page', 'New memo textbox', 'New Photo', 'New special panel', 'New button', 'New link', 'Delete', and 'Properties'.

Figure B3.1

Design

Nested panels are created as sub panels of a main panel. First, you have to create a main panel that will contain the sub panels. In order to achieve that, you have to click the button "Layout" in form design mode and then create the main panel inside the window that appears.

Right click inside the Layout window and create a new panel, which will be used as "header" panel. Enter the number of the nested panels inside the Columns box and then define their width as percentage per column. Then, select the "header" panel and create new panels inside it, by right clicking and choosing "New Panel". Now, you will have one "header" panel with sub panels, that will be displayed horizontally inside the form.

Properties

The properties of nested panels are the same as Panel properties, except for the "master" panel, whose columns define the width of the sub panels (Figure B3.2).

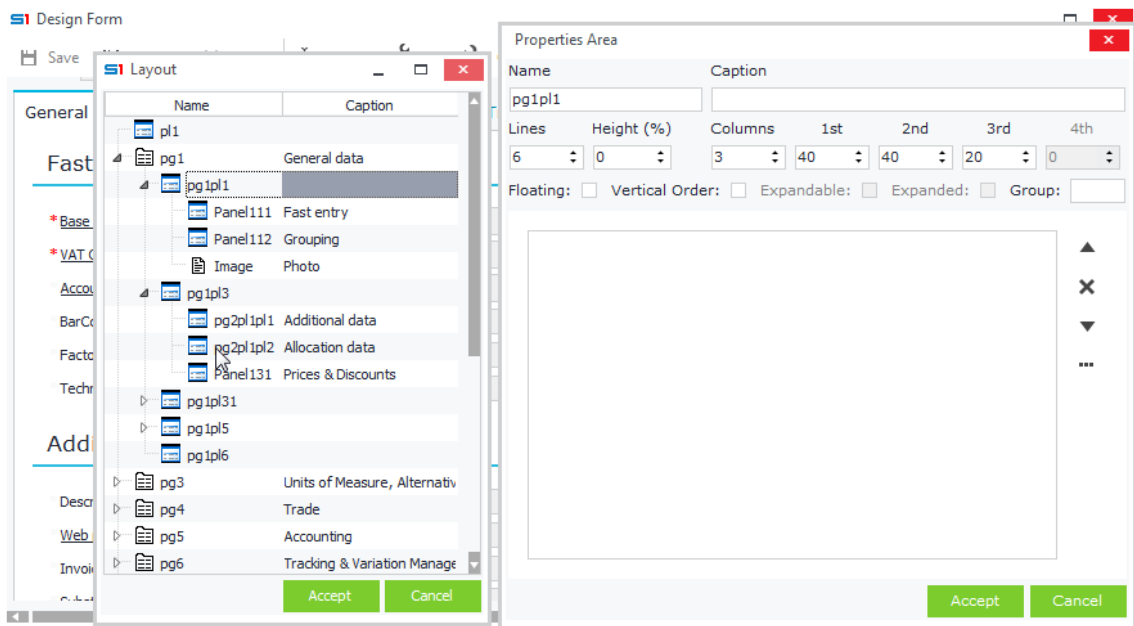


Figure B3.2

B.4 Sub Forms

Sub forms are pop up forms that can be displayed inside the main object form, through user’s interaction. They have many uses, for example you can use a pop up form to display a more detailed view of a datagrid record that contains many fields (Figure B4.1). Another example would be to use a sub form to display additional data associated with a record, such as calculated values, datagrids with financial data, etc. when user clicks on a button of a form or when a form event occurs (using form script).

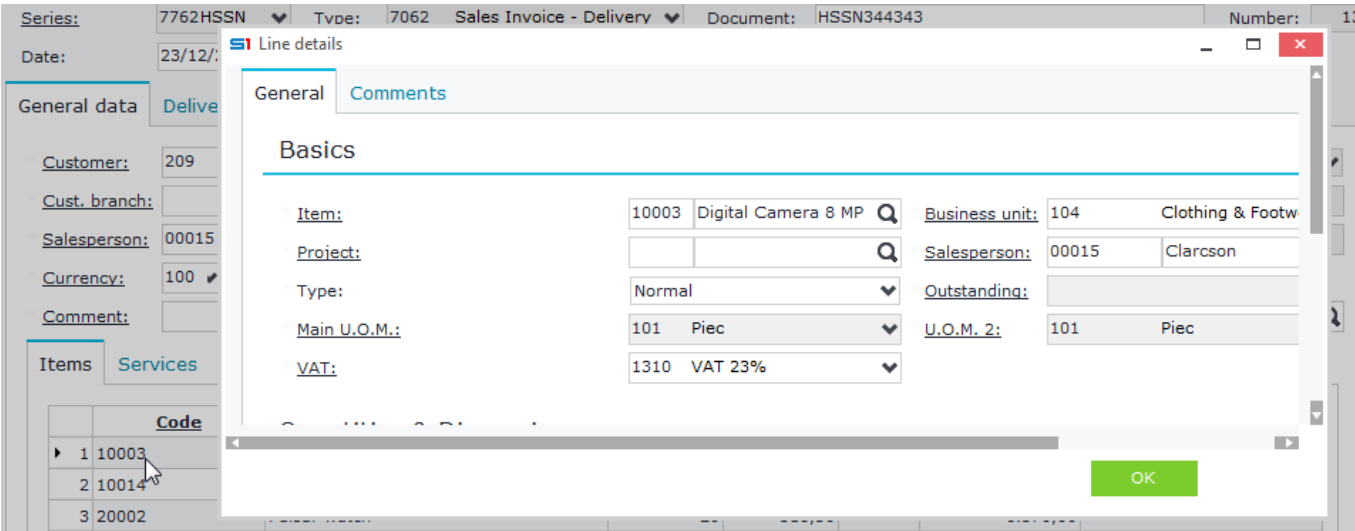


Figure B4.1 – Sub Form in sales documents datagrid

Design

In design mode of an object form, select the “Layout” button and then right-click to create a new “Sub form”. In the properties window of the newly created form enter its name, which can be later used to access the form in datagrids or in form script (Figure B4.2).

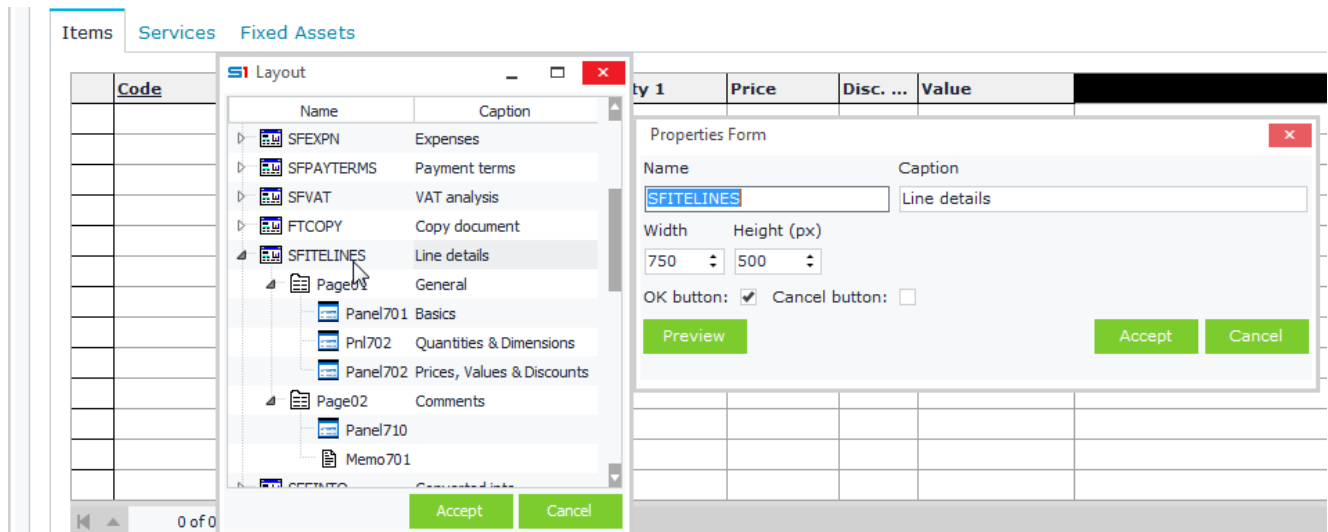


Figure B4.2 – Sub Form SFITELINES in sales documents

Any control can be used inside a sub form as long as it is created under the sub form’s node in the layout window. This means that when you right click a sub form (using the layout tree window) you can add panels, tabs, images, datagrids, etc.

One way to display a sub form (e.g. “MySubForm”) in runtime is through button, that can be created using the command **RUNB_SubFormName=Open Sub Form** (Figure B4.3). Alternative ways of creating buttons are discussed in section [Buttons](#).

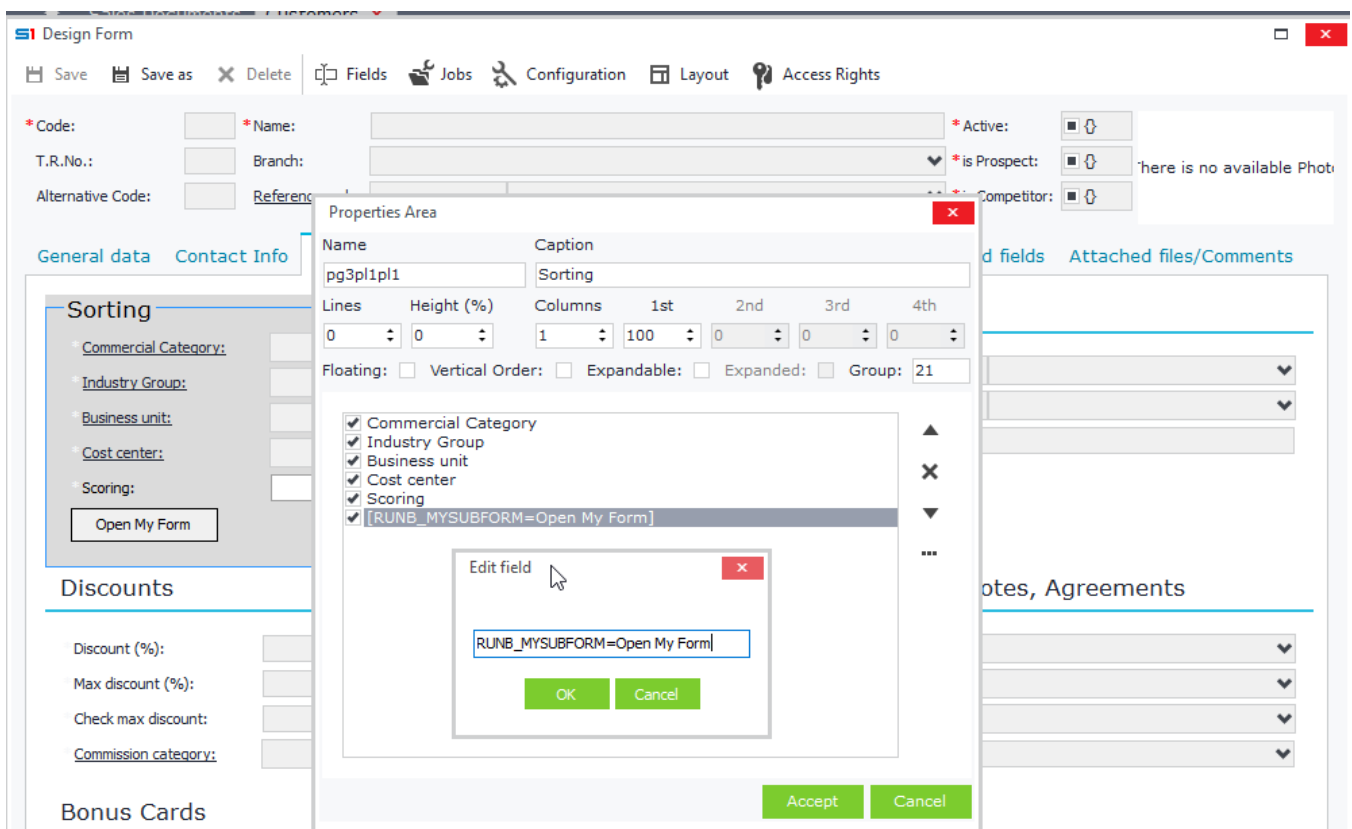


Figure B4.3

Sub forms can also be displayed when users double click on datagrid rows. This can be done by entering the name of the sub form in the “**Popup form**” property of the datagrid (see [Datagrid Properties](#)). Figure B4.4 displays the properties of item lines datagrid in sales documents form.

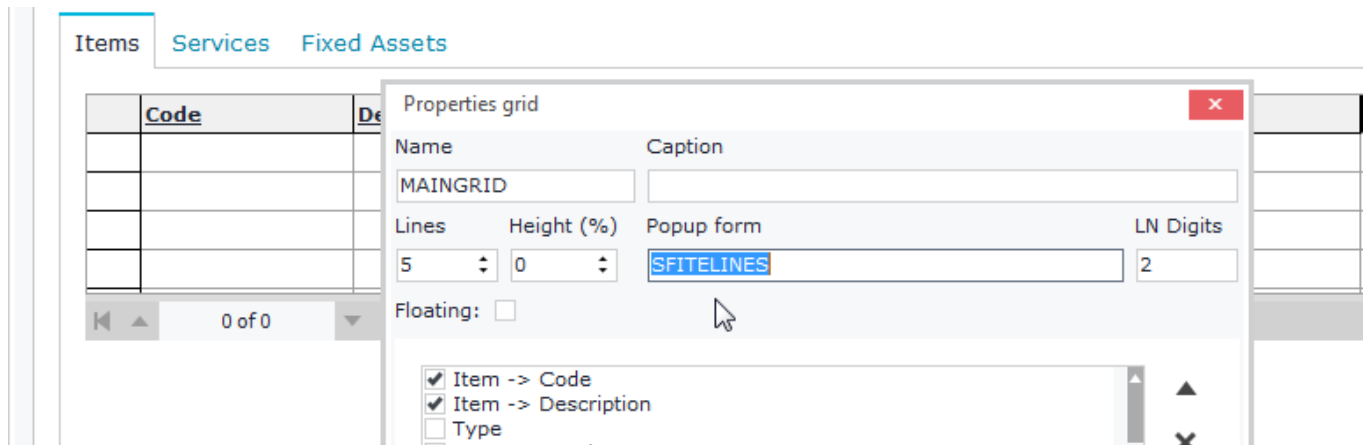


Figure B4.4 – Sales documents – Item lines datagrid properties

Properties

Sub form properties are listed in the table below.

PROPERTY	DESCRIPTION
Name	Sets the name of the pop up form. This name can be used to access the sub form from datagrid records or in form script.
Caption	Sets the text that appears as title of the sub form.
Width	Sets the width of the sub form.
Height	Sets the of the sub form.
OK button	Indicates that the OK button will be displayed on the bottom side of the sub form
Cancel button	Indicates that the Cancel button will be displayed on the bottom side of the sub form

PopUp Form Examples

Example 1

Display a pop-up form from Datagrid records

This example demonstrates the design of a sub form that is displayed when users double click on a row of the Items datagrid, which is inside the object “Sales documents”.

- In design mode of a “Sales documents” form, click on the button “Layout”, right click and select “New Form” (Figure B4.5).

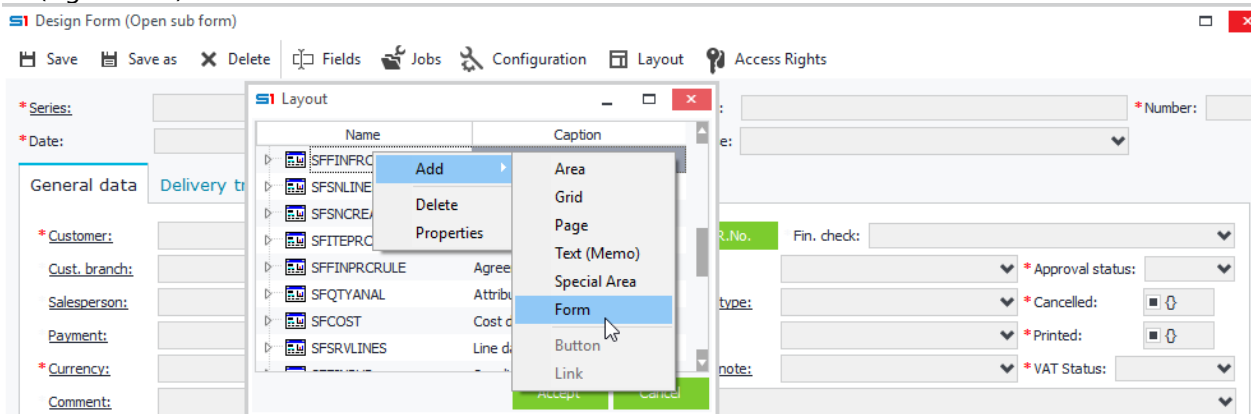


Figure B4.5

- Enter a name for your new sub form (e.g. MyPopUpform) in the Form Properties (Figure B4.6).

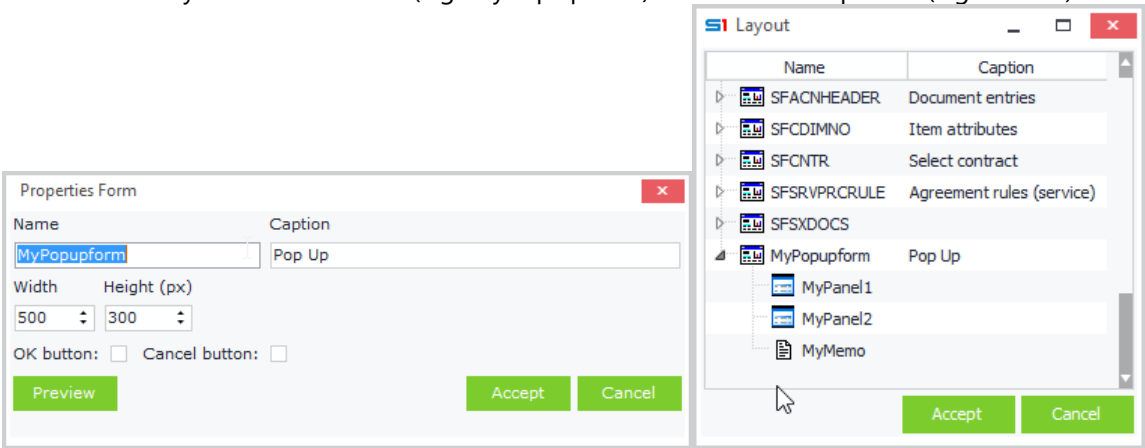


Figure B4.6

- Create panels, datagrids, tabs and any other control by right-clicking and selecting the appropriate control (Figure B4.6). Fields can be added through the properties of controls.

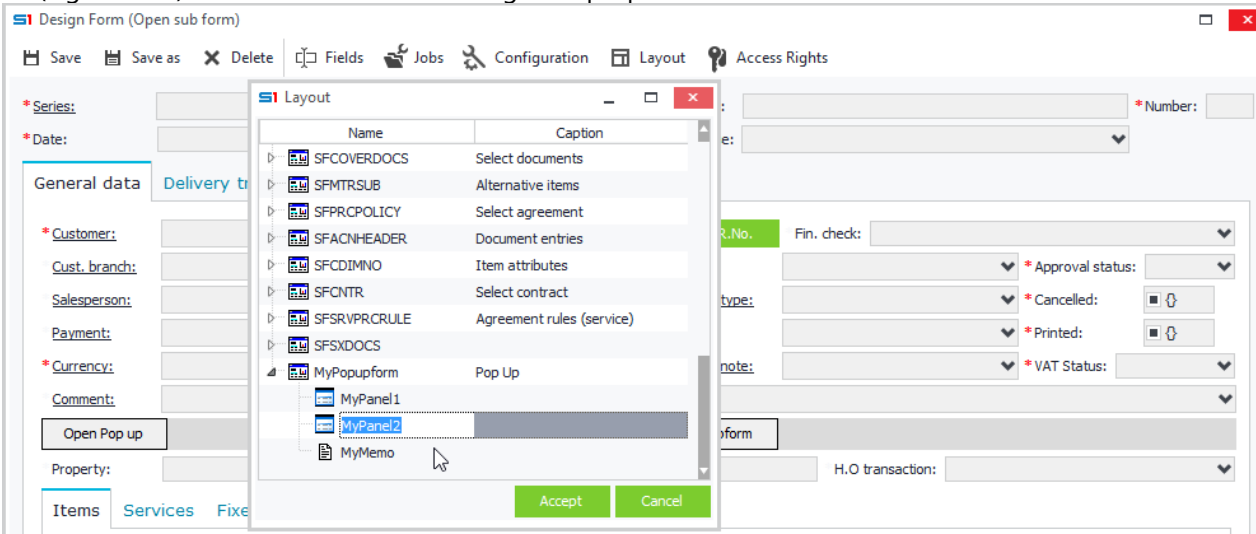


Figure B4.6

- Finally, add the name of your sub form inside the “pop up form” property of the Items datagrid. This way your sub form will be displayed when users double click on a record line of the Items datagrid (Figure B4.7).

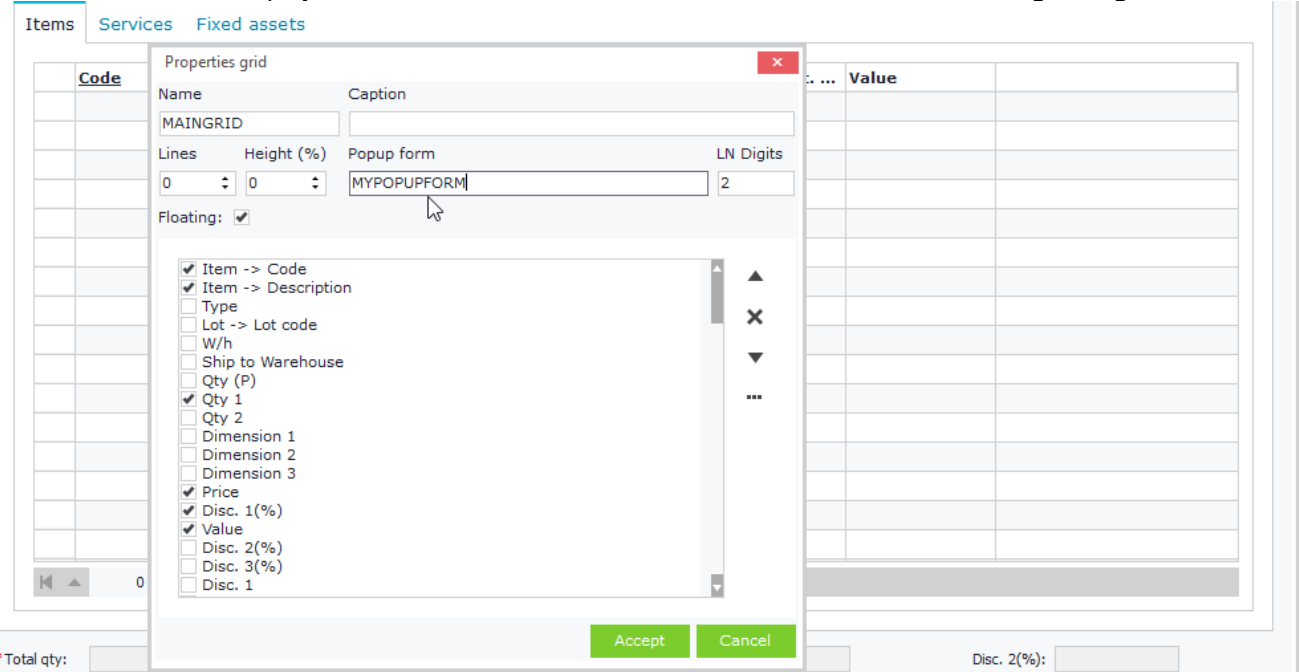


Figure B4.7

Example 2**Display a pop-up form from Button click**

Create a new button inside the “Fields” property of a panel, by entering a new field (right click) with the command RUNB_ButtonID=ButtonCaption (e.g. RUNB_20140101=Open Pop up).

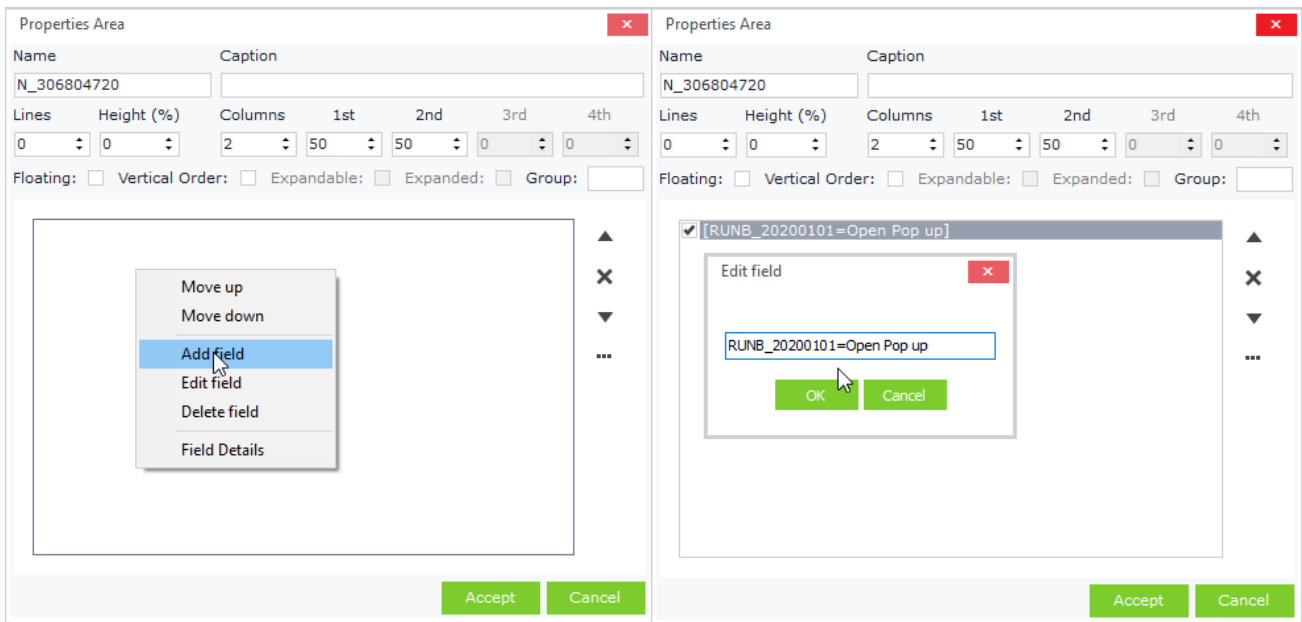


Figure B4.8

In the tab “Script” of the configuration window add the following JavaScript code (Figure B4.9):

```
function EXECCOMMAND(cmd) {
  if(cmd == 20140101)
    X.OPENSUBFORM('MYPOPUPFORM');
}
```

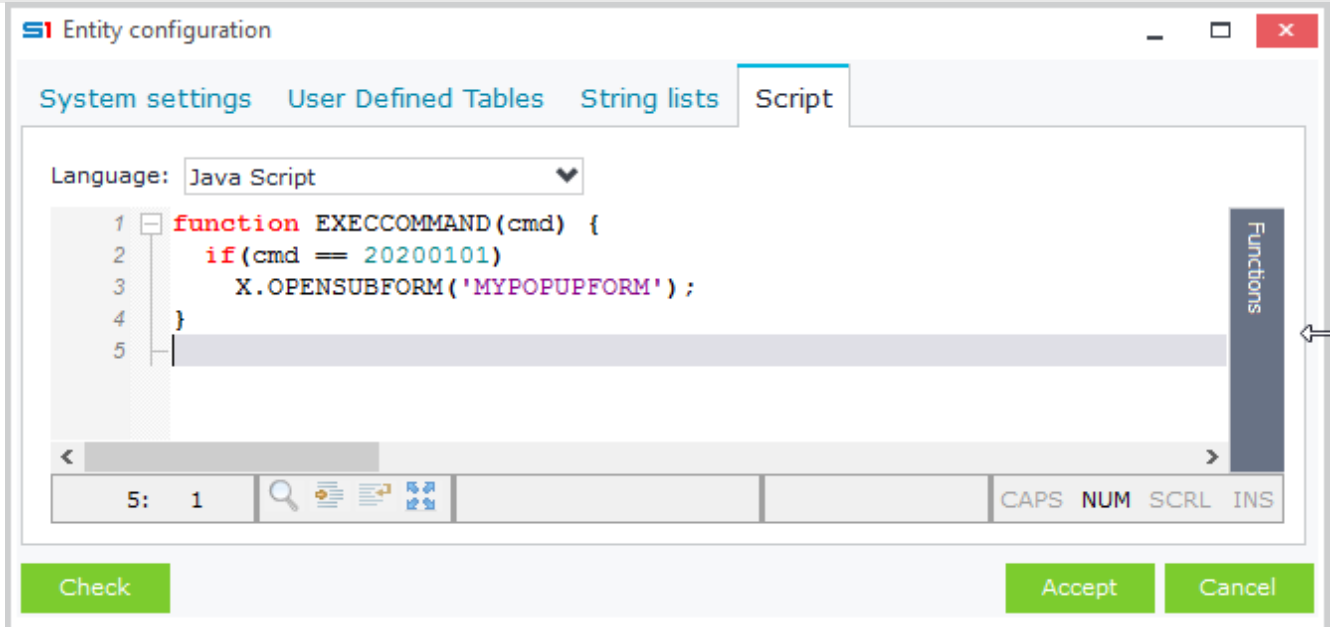


Figure B4.9

Alternatively, you can simply use the name of the sub form as the id of the button. In this case you don't need to write any code at all (Figure 4.10).

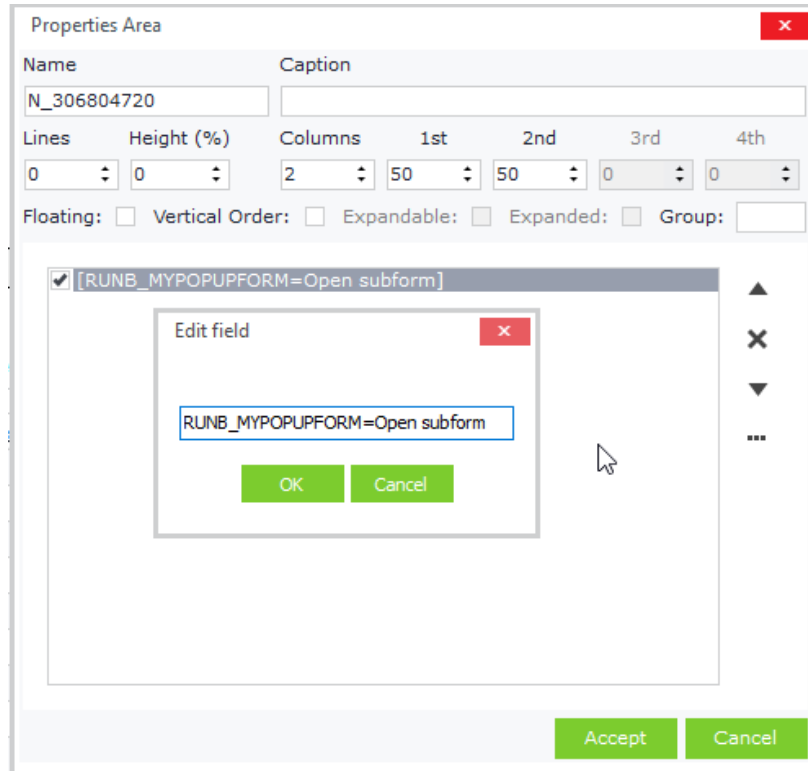


Figure B4.10

Another way to open a sub form without writing script code would be to use a (virtual) numeric field for creating a button. Then, inside the property "Editor" of the field you would have to enter the following command:

XCMD:MYPOPUPFORM (Figure B4.11).

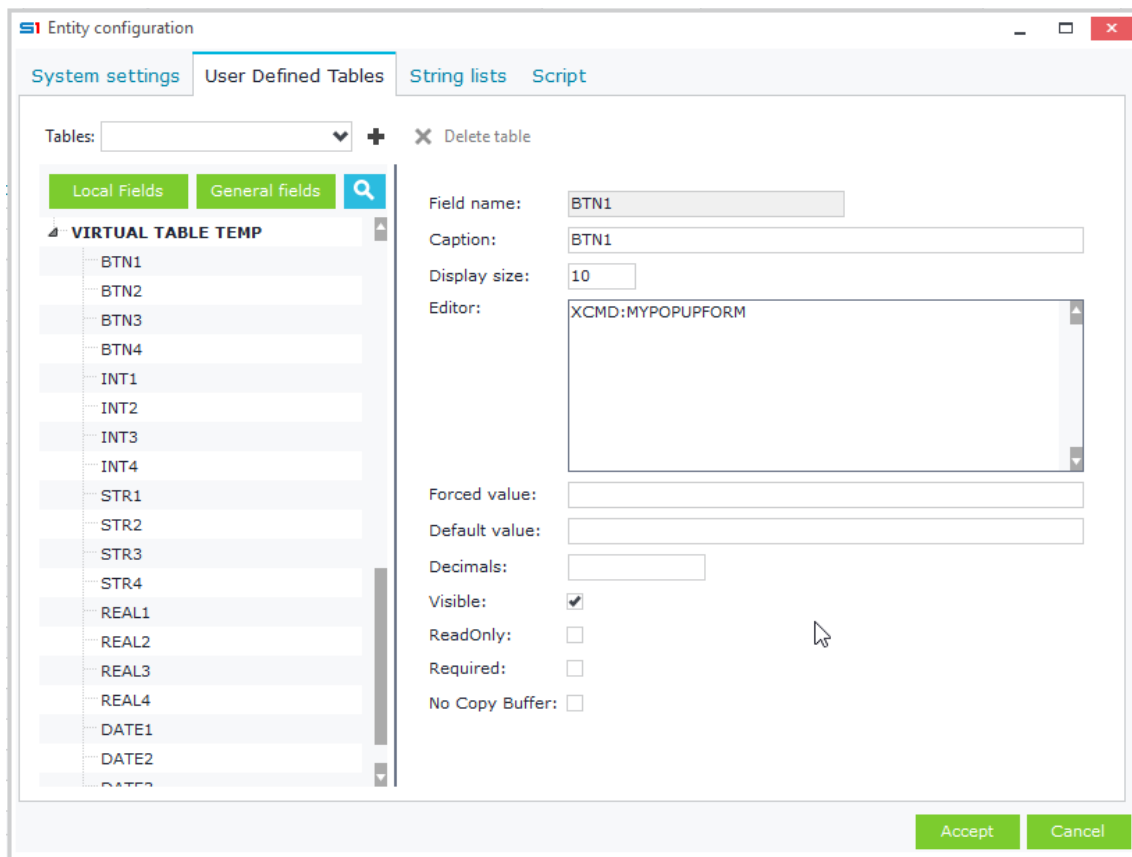


Figure B4.11

C. Data Controls

Data entry controls display fields from database tables, virtual tables, view tables and database views that are added in the form through Configuration button (tab User defined tables). The controls created depend on the data types of the fields, which means that different types of fields create different controls (textboxes, memos, edit lists, etc). Fields can be added in a layout control of a form through drag and drop, either from the tab “User defined tables” under the window “Configuration” or from the window “Fields”.

C.1 String Textbox

String Textboxes display data from string (varchar or nvarchar) fields and allow users to edit them as unformatted text. For example, the field “Name” of Items (table MTRL) is displayed in a textbox control and allows text input.

Design

A String Textbox is automatically created when you drag and drop a string field inside a panel. Another way is by dragging and dropping fields inside the box “Fields” of panel properties. Moving fields between panels is permitted and can be performed from panels’ properties window (Figure C1.1).

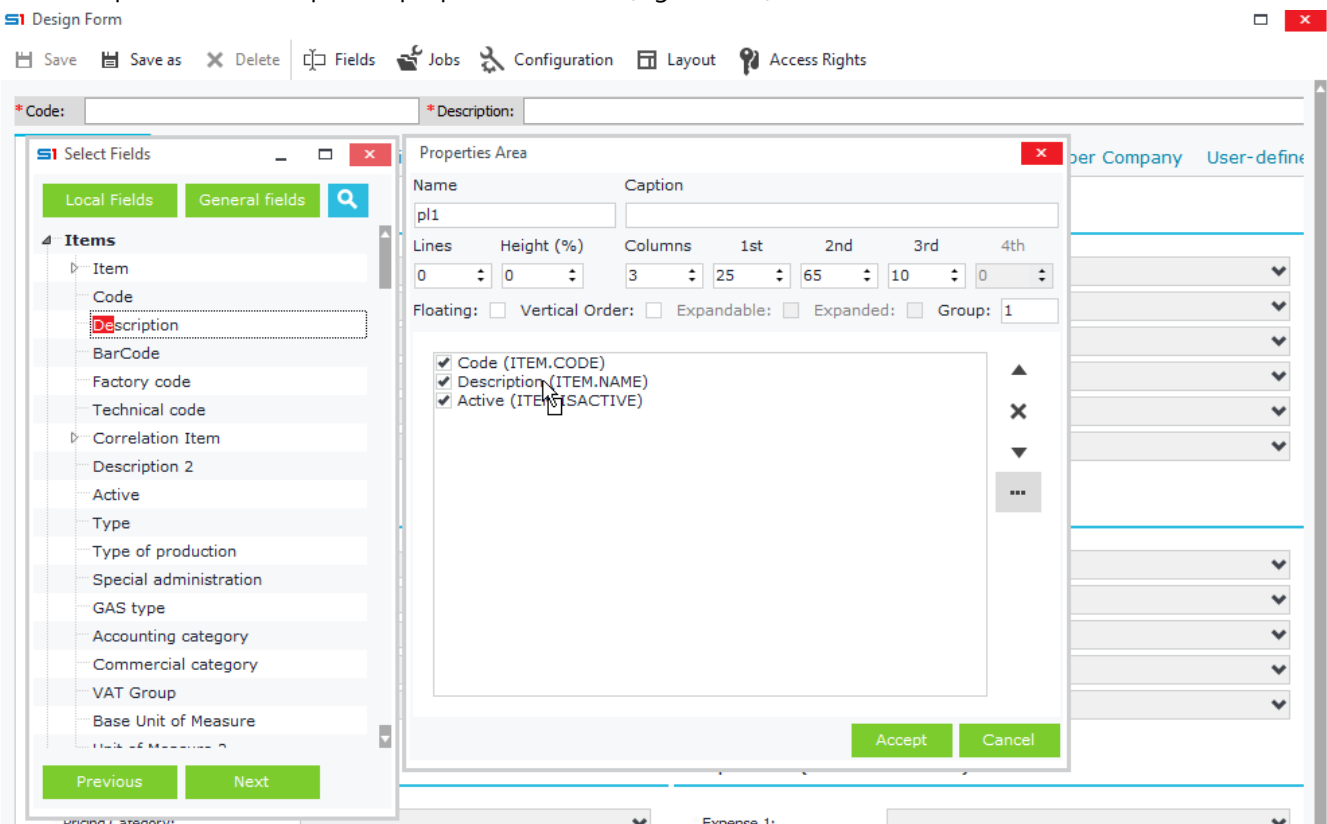


Figure C1.1 – Add field in panel

Properties

Click on the button “Configuration”, move to the tab “User defined tables” and click on a field in order to display and modify its the properties, which affect the way it is processed (Figure C1.2).

The screenshot shows the 'Entity configuration' window with the 'User Defined Tables' tab selected. On the left, a list of fields is shown under the 'MY TRDR' table. The right pane displays the configuration for the 'CCCMYTRDR' field. The configuration includes a caption 'MY TRDR', a display size of 12, and a custom editor 'CUSTOMER'. Other options like 'Forced value', 'Default value', 'Decimals', 'Visible' (checked), 'ReadOnly', 'Required', and 'No Copy Buffer' are also present.

Figure C1.2 – Field Properties

Field properties are summarized below.

PROPERTY	DESCRIPTION
Field name	Name of the field (Read only)
Caption	Title of the textbox
Display size	Column width size. Affects only fields that are added to grids.
Forced Value	Indicates a value that will be applied to the field when the record is submitted.
Default Value	Sets a default value to the field, that applies to new entries. Users can change it at runtime.
Visible	Indicates whether the field is displayed or not.
ReadOnly	Specifies whether the contents of the textbox can change by users at runtime. Changing the value of the textbox by script code is allowed.
Required	Specifies whether the textbox must be filled in order to submit a record.
Exclude from Copy Buffer	Visible only in custom fields. If selected then the value of this field is not copied, when a record is copied using the toolbar button “Copy”

C.2 Numeric Textbox

Numeric Textbox control is textbox that accepts only numeric input.

Design

This type of textbox is created the same way as string textboxes and its properties differ only when it comes to floats. This type of fields displays the number of decimals entered in the “Decimals” property inside the “Configuration” window of a screen form.

Default decimal places

SoftOne application gives you the option to display float fields with the same number of decimals places as the ones you have selected inside each company’s properties (Figure C2.1). This means that if you enter the letter **P**(Figure C2.2) in the property “Decimals” of a float field, then the field will display with the same number of decimal places as configured for company prices(field PDECIMALS inside table COMPANYY).

The available formats are summarized in the following table.

COMMAND	DESCRIPTION	CORRESPONDING COMPANY FIELD
P	Prices	COMPANY.PDECIMALS
V	Values	COMPANY.VDECIMALS
Q	Quantities	COMPANY.QDECIMALS
%	Percentages	COMPANY.DDECIMALS
C	Cost Prices	COMPANY.CDECIMALS

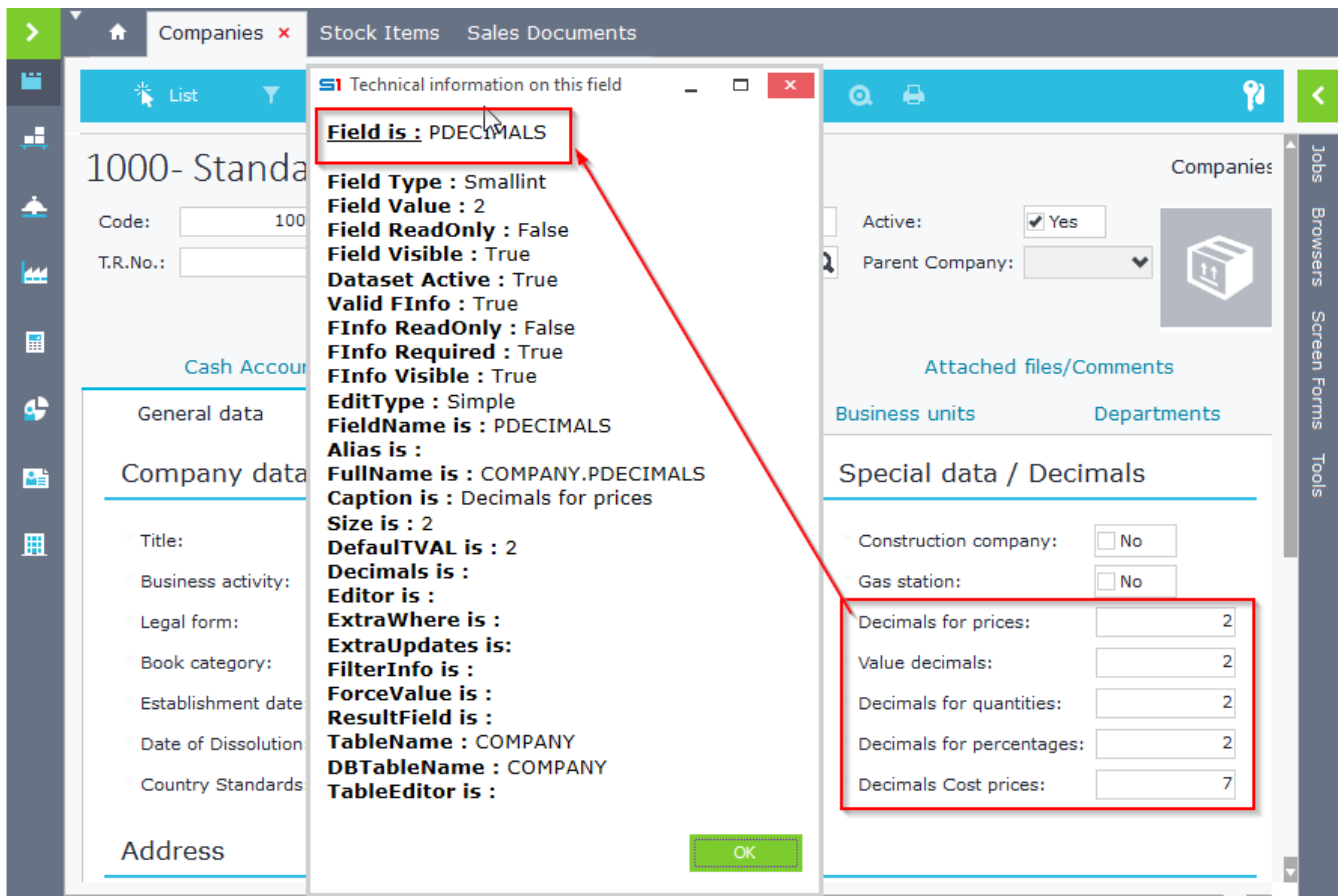


Figure C2.1

Entity configuration

General Settings | **User Defined Tables** | String lists | Script

Tables: + X Delete table

Local Fields | **General fields** |

- Description
- Country of origin
- Qty 1
- Qty 2
- Item Variation 1
- Item Variation 2
- Item Variation 3
- Lot
- Wholesale price
- Retail price**
- Altern. packag.
- Active
- Inserted on
- Max / Min level per branch
- Item extra fields
- Item Data per Company

Field name:

Caption:

Display size:

Editor:

Forced value:

Default value:

Decimals:

Visible: ☒

ReadOnly: ☐

Required: ☐

Accept Cancel

Figure C2.2

Calculation formulas

At runtime, there is an option of entering numbers inside numeric textboxes using formulas. Formulas must always begin with the equal sign "=" (Figure C2.3).

Companies | Stock Items | **Sales Documents** x

List | Filter | New | Save | Delete | Navigation icons

New entry

Series: 7001OFFR | Type: 7001 Quotation | Document: OFFR000001

Date: 01/04/18 | Branch: 1000 Head Offices | Warehouse: 1000 Central

General data | **Delivery - Transfer** | International transactions | Other data

Customer: 100 Customer 1 | Status: | Movement type: | Payment: | VAT Status: Normal

Cust. branch: | Exchange rate: 1 | TRP exchange rate: 1

Currency: 100 EURO | Comment:

Items

	Code	Description	Qty 1	Price	Disc. ...	Value
1	100	Item 1	1	=5*7		35,00

Figure C2.3 – Calculation formulas in numeric fields

C.3 Memo Textbox

Memo control is a multiline textbox that displays data from string fields. Examples of memo control fields are fields REMARKS which can be found in almost every object and table (TRDR, MTRL, FINDOC, etc.).

Design

In order to display a string field as memo you have to right click a view or page (in design mode) and choose to create a "New text". Then simply drag and drop a string field inside the "Fields" box of the properties window.

Properties

The table below summarizes all the memo textbox properties

PROPERTY	DESCRIPTION
Caption	Adds a title in the top line (header) of the control
Height	Changes the height (lines or percentage). If not entered then it auto fits the size of the panel.
Scrollbars	<p>Visibility of horizontal and vertical scrollbars can be achieved by entering the numbers 1, 2 or 3 inside the Decimals property of the selected field.</p> <p>1: Horizontal scrollbar visible 2: Vertical scrollbar visible (Figures C3.1 & C3.2) 3: Both scrollbars visible (Figure C3.4)</p>

The screenshot shows the 'Entity configuration' window with the 'String lists' tab selected. On the left, under 'Local Fields', the 'Comments' field is highlighted with a red box. On the right, the configuration for the 'REMARKS' field is shown. The 'Caption' is 'Comments', 'Display size' is 50, and 'Editor' is a multiline text area. The 'Decimals' property is set to 2, also highlighted with a red box. The 'Visible' checkbox is checked, while 'ReadOnly' and 'Required' are unchecked. At the bottom right, there are 'Accept' and 'Cancel' buttons.

Figure C3.1

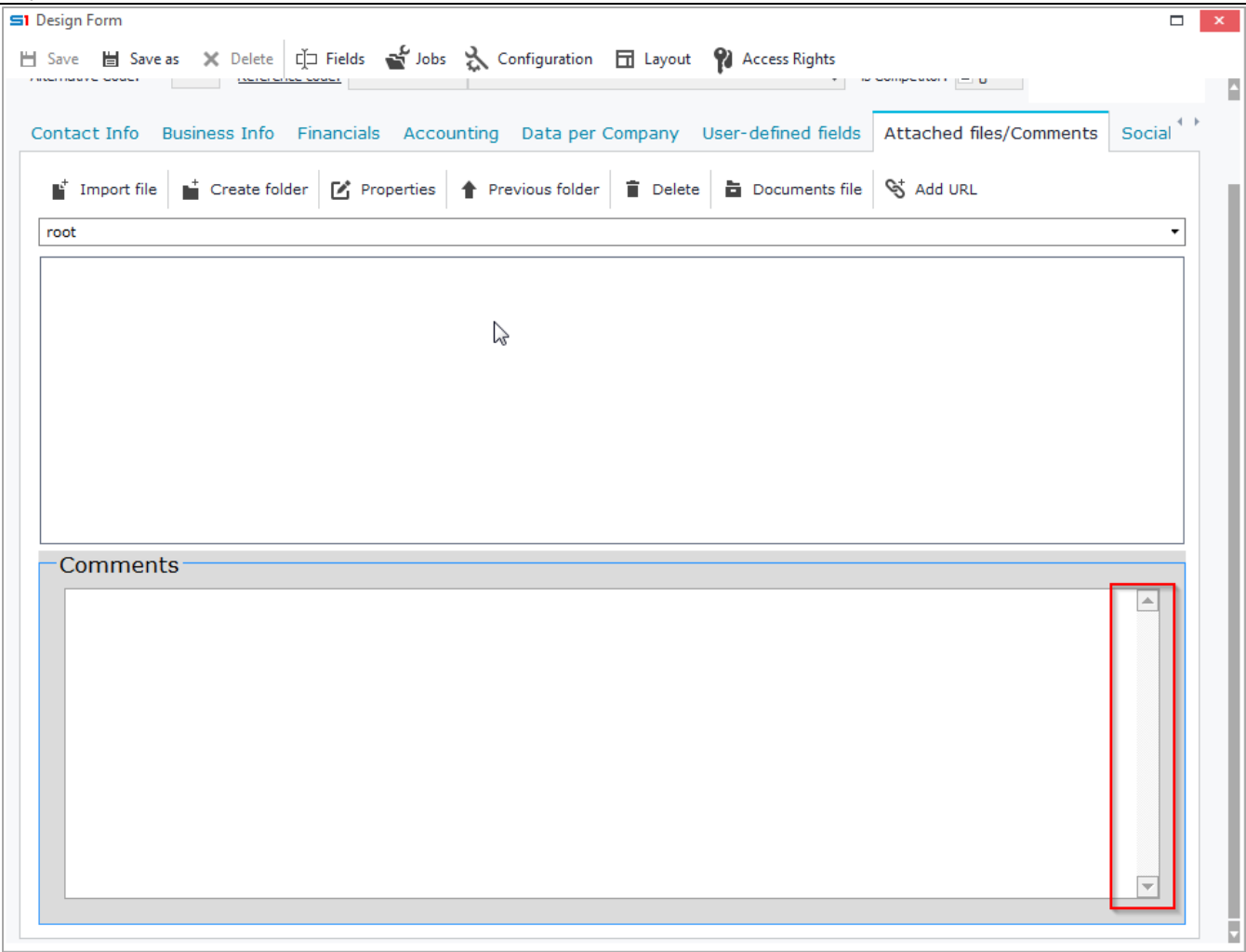


Figure C3.2

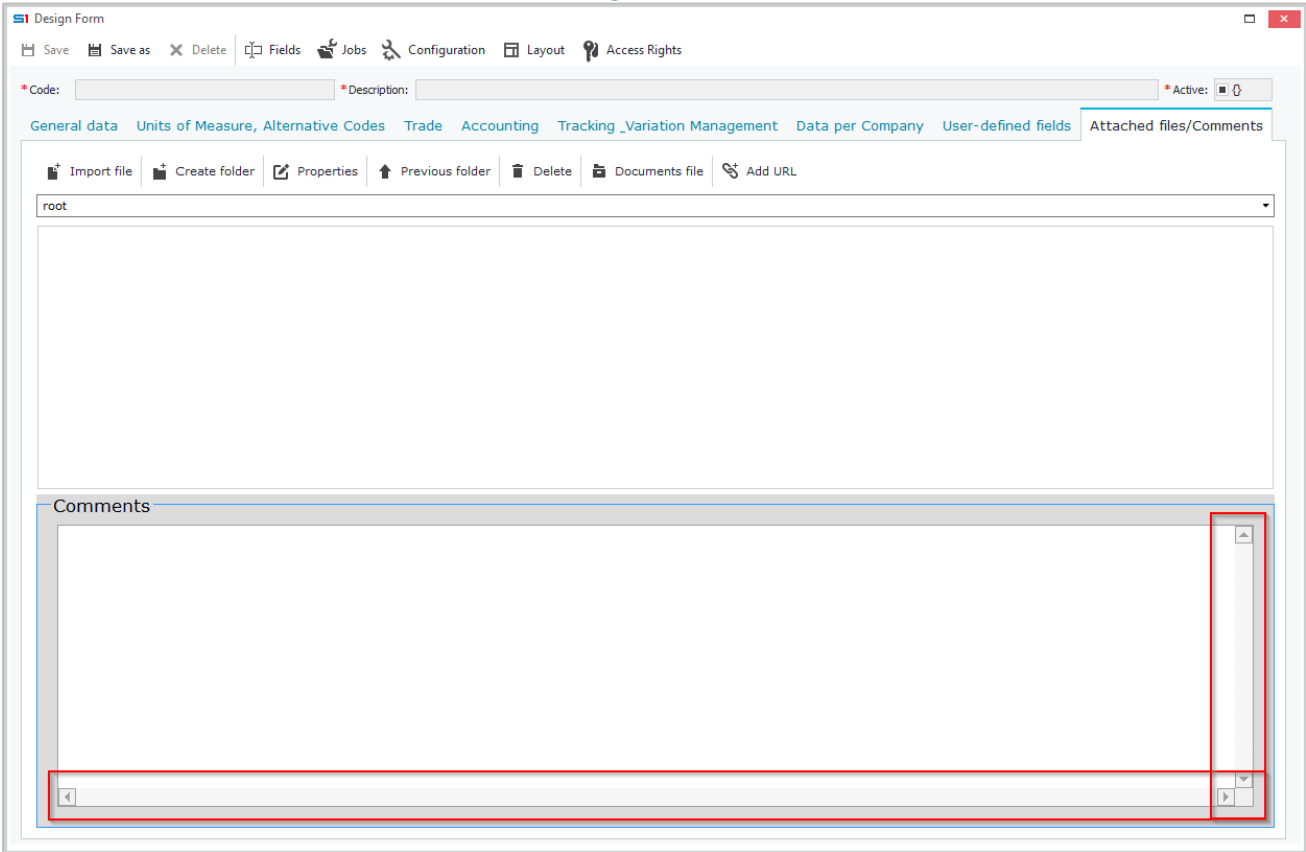


Figure C3.4

C.4 Password Textbox (Hash SHA-2)

This control is a masked textbox that hides data and displays characters with asterisks. It generates a SHA-256 encrypted hash and applies to string fields.

Design

In form design mode, open the configuration window, select a string field and enter the command **\$PASSWORDH** inside the Editor property on the right hand side of the window (Figure C.4.1).

In runtime mode the field is displayed as in figure C.4.2.

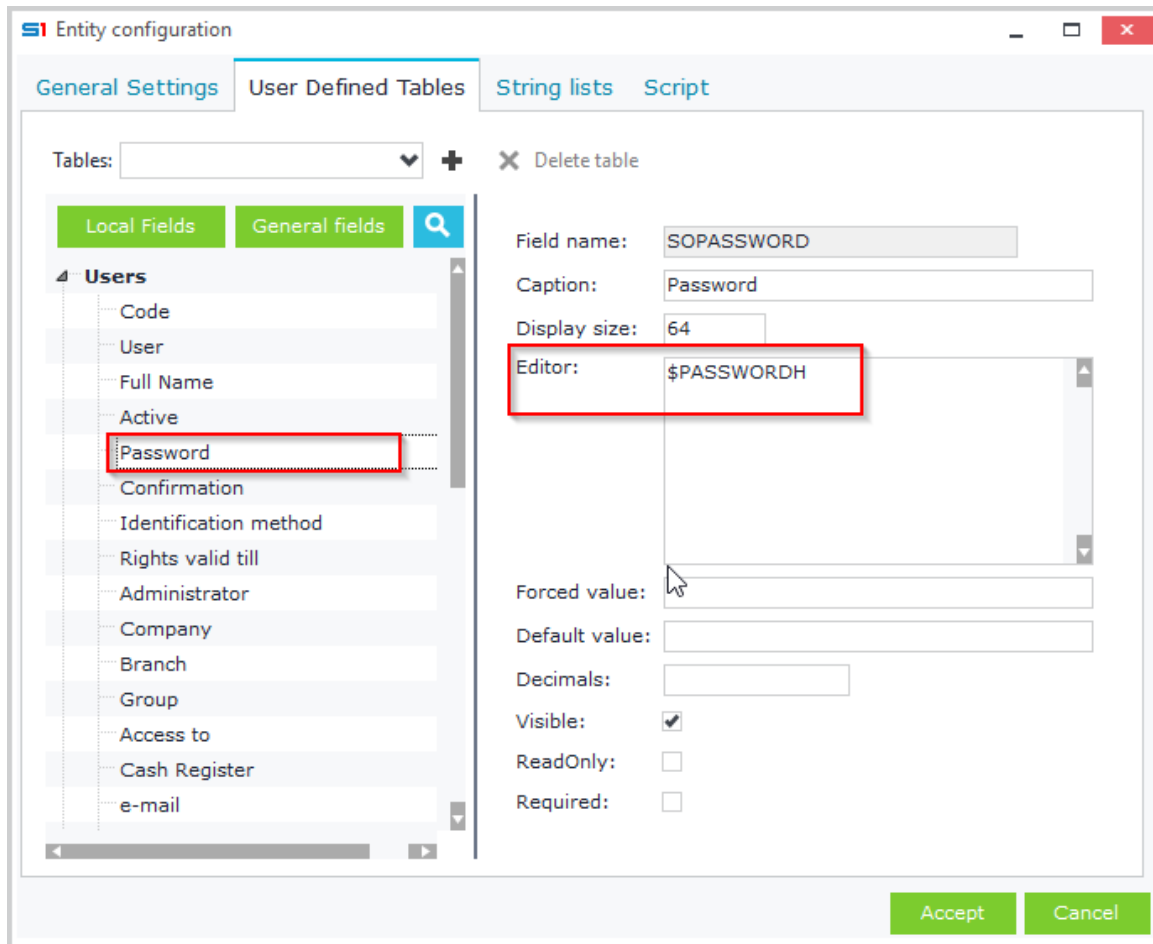


Figure C.4.1

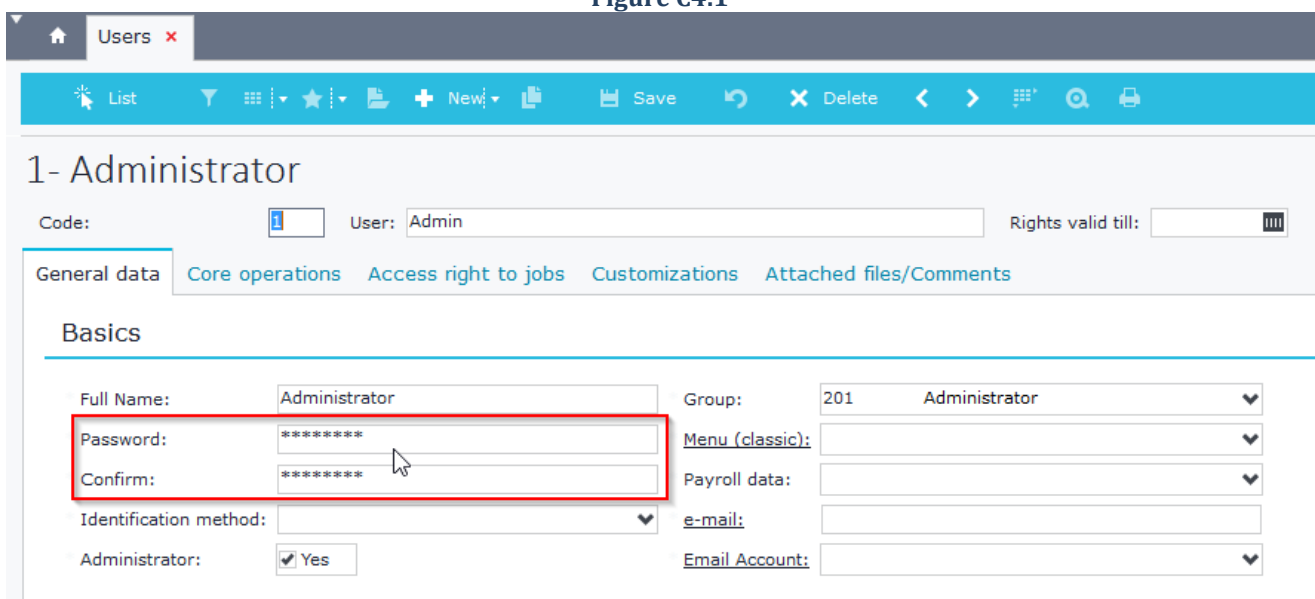


Figure C.4.2

C.5 Webpage / Email Textbox

These controls are textboxes, that add a link to their captions, which auto opens the corresponding programs (the default Browser for Webpage and the default email client for Email).

Design

In form design mode, open the configuration window and select the field you want to display as webpage or email. Then in Editor Property, on the right hand side of the window, add the command **\$WEBPAGE**(Fig. C5.1) for webpage or the command **\$EMAIL** for email field (Fig. C5.2). In runtime mode, fields are displayed as in figure C.5.3

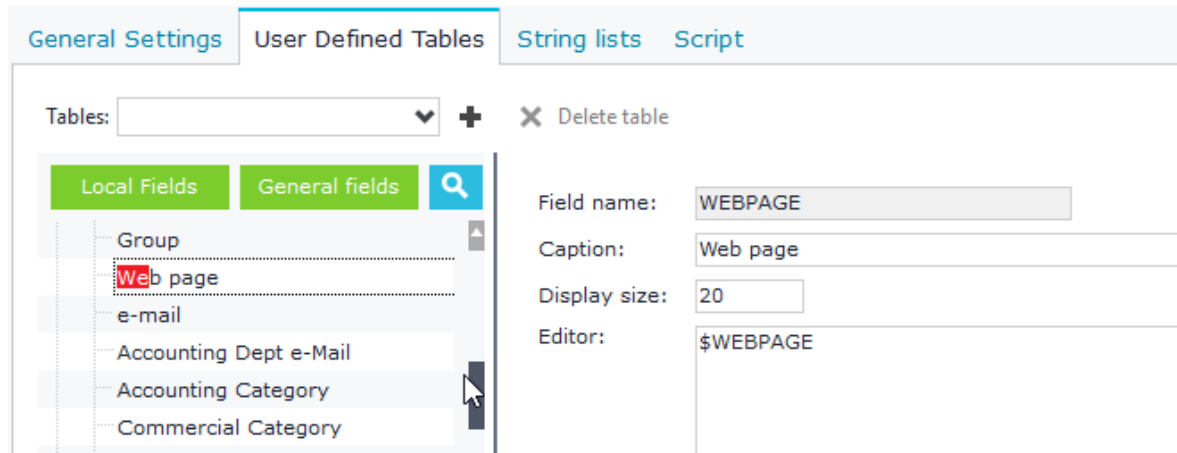


Figure C5.1

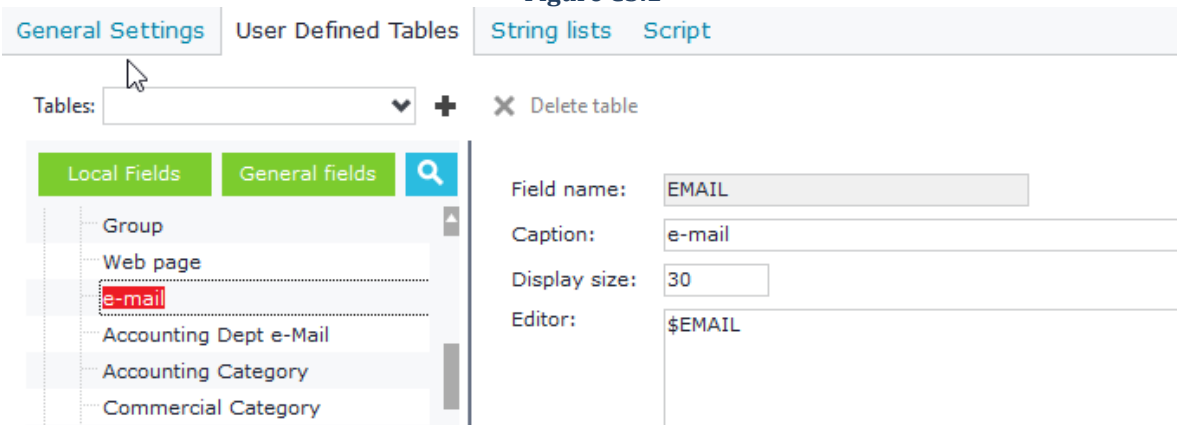


Figure C5.2

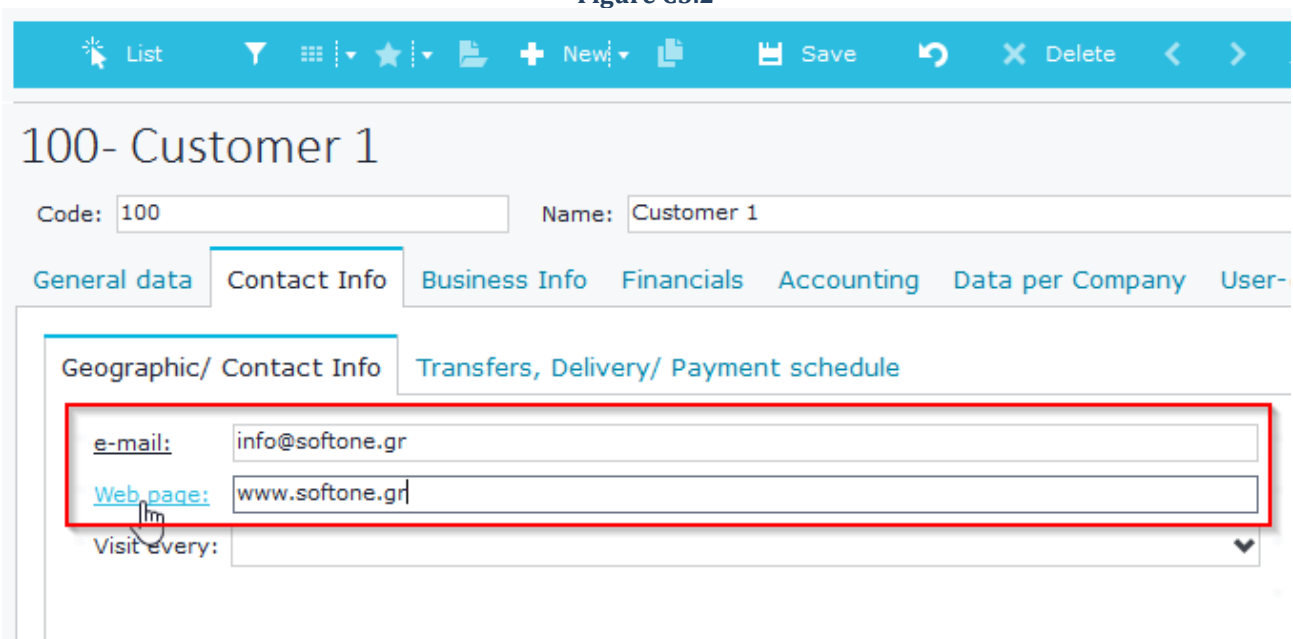


Figure C5.3

C.6 Grid

Grid control is used for displaying child database tables, database views and virtual tables that have been created through SoftOne database designer. Figure C6.1 shows the grid "Item lines" of Sales Documents entity and the grid "Alternative codes" of Items entity, which are used as child tables to the parent tables of their objects.

The screenshot displays two software interfaces. The top interface is for 'Sales Documents' (SISN000066- 05/01/2018). It features a header with document details and a 'General data' section. Below this is a table titled 'Items' with columns: Code, Description, Qty 1, Price, Disc. ..., and Value. The table lists 13 items, including Digital Camera 8 MP, TV LCD 21, Photo michachi DSLR 10 MP, Flash Camcorder, Flash Full HD Camcorder, HDD 60GB Camcorder, Digital Photo Frame 8", Home cinema 1000 Watt, Crutches, Pharmacy kit, Liquid decontamination (Lots), Medical oxygen bottle (lots), and Rescue Life Jackets. The bottom interface is for 'Stock Items' (Default Quantities, Stock Limits per W/h). It shows a table titled 'Alternative codes' with columns: Code, Description, Qty 1, Qty 2, Lot code, Item Vari..., Item Vari..., and Item. The table lists 5 items, all with the description 'Orange juice' and varying codes (23423, 664334w, cc24422, gg43434f5, ss434465).

Sales Documents: SISN000066- 05/01/2018

Series: 7062SISN Type: 7062 Sales Invoice - Delivery Note Document: SISN000066 Number: 66
 Date: 05/01/2018 Branch: 1000 Head Offices Warehouse: 1000 Central

General data Delivery - Transfer International transactions Other data

Customer: 174 Dimitris Stathopoulos Status: Approval status: Appro
 Cust. branch: Movement type: 1000 For Sale Cancelled: No
 Salesperson: 00001 Smith Payment: Printed: No
 Currency: 100 EURO Exchange rate: 1 TRP exchange rate: 1 VAT Status: Normal Check T.R.No.
 Comment:

Items Services

	Code	Description	Qty 1	Price	Disc. ...	Value
1	10003	Digital Camera 8 MP	1	122,00		122,00
2	10002	TV LCD 21	1	1.244,00		1.244,00
3	10004	Photo michachi DSLR 10 MP	1	12,00		12,00
4	10005	Flash Camcorder	1	122,00		122,00
5	10006	Flash Full HD Camcorder	1	34,00		34,00
6	10007	HDD 60GB Camcorder	1	31,00		31,00
7	10008	Digital Photo Frame 8 "	1	2,00		2,00
8	10009	Home cinema 1000 Watt	1	1,00		1,00
9	10010	Crutches	1	2,00		2,00
10	10011	Pharmacy kit	2	1,00		2,00
11	10012	Liquid decontamination (Lots)	2,00	2,00		4,00
12	10013	Medical oxygen bottle (lots)	2	1,00		2,00
13	10014	Rescue Life Jackets	2	3,00		6,00

Total qty: 955,00 Disc. 1(%): 0,00 Disc. 1: 0,00 Disc. 2(%): 0,00
 Net: 26.164,00 VAT: 6.017,72 Expenses: 0,00 Total: 32.181,72

Stock Items Default Quantities, Stock Limits per W/h Default Warehouses and Storage bins

Alternative codes Supplier codes Empty Containers\Accessories\Spare parts\Consumables

Alternative Code is Unique: Yes

	Code	Description	Qty 1	Qty 2	Lot code	Item Vari...	Item Vari...	Item
1	23423	Orange juice	1	0,00				
2	664334w	Orange juice	1	0,00				
3	cc24422	Orange juice	1	0,00				
4	gg43434f5	Orange juice	1	0,00				
5	ss434465	Orange juice	1	0,00				

Figure C6.1

Design

While in object form design, right click on a panel and select "New Datagrid" (Figure C6.2). Drag fields from the "Fields" or "Configuration" windows inside the datagrid as in Figure C6.3.

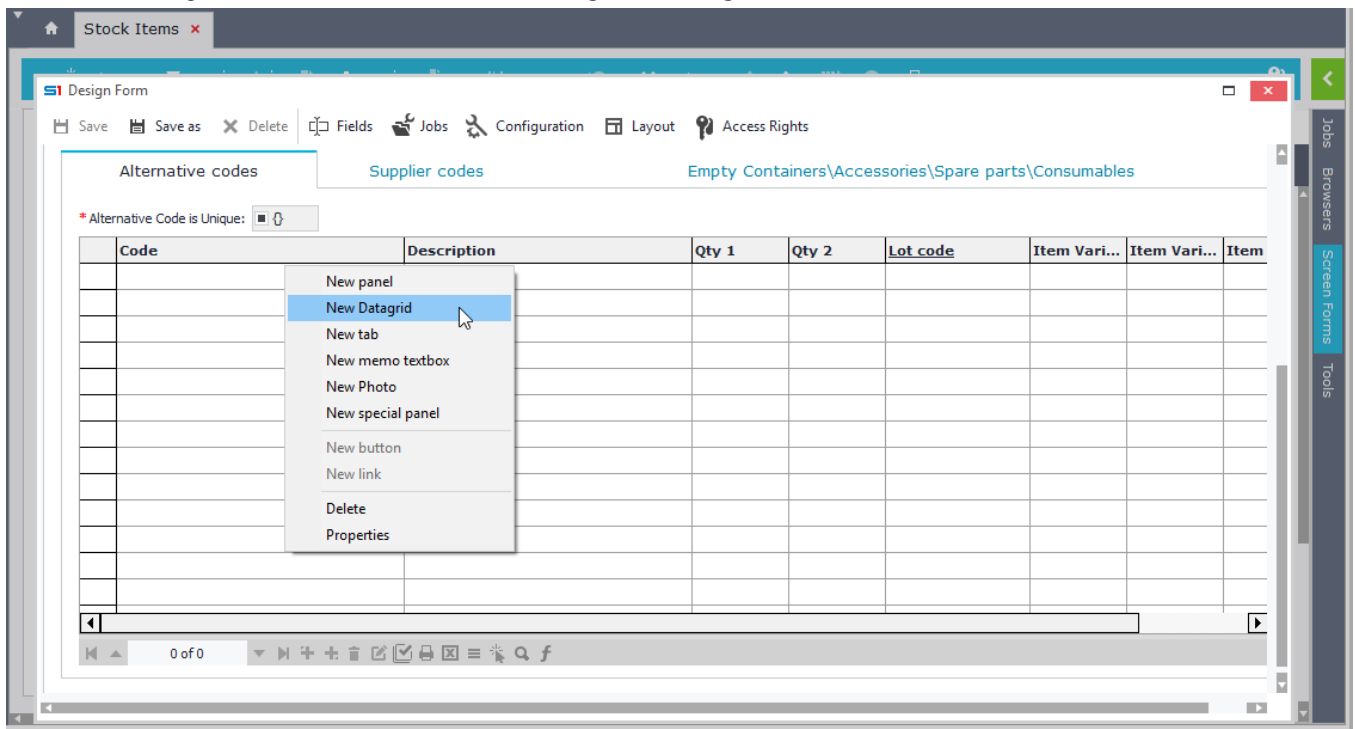


Figure C6.2

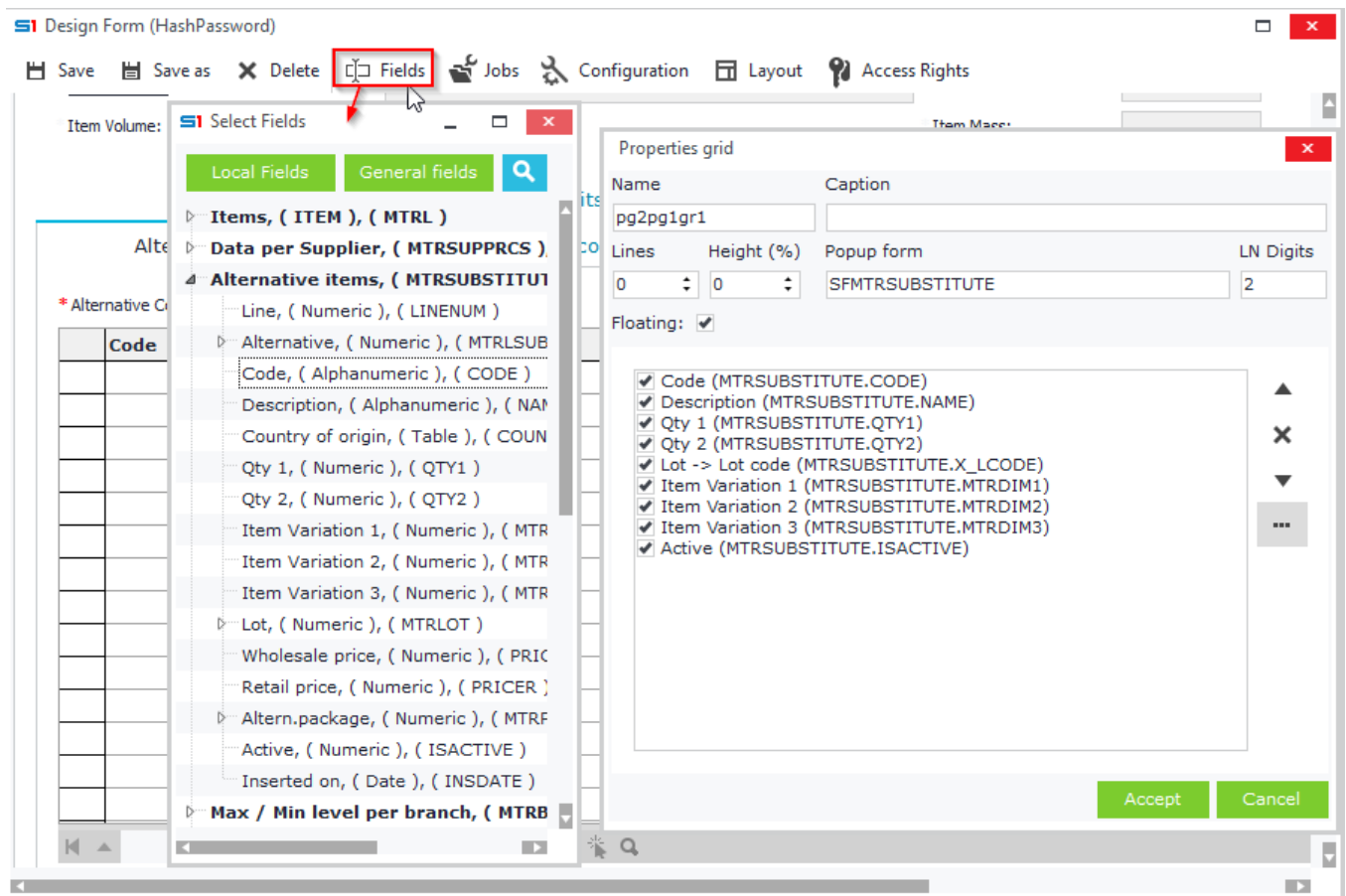


Figure C6.3

Properties

The table below summarizes all the grid properties (Figure C6.4).

PROPERTY	DESCRIPTION
Name	Sets the name of the datagrid
Caption	Sets the title of the datagrid
Lines	Sets the number of lines that will be displayed.
Height	Sets the percentage height of the datagrid.
Popup form	Defines the name of the pop-up form that will be displayed when user double clicks a datagrid row.
LN Digits	Sets the width (in digits) of datagrid's header column. For example if you enter the number 2, then the header column's width will be adjusted to be able to display up to number 99. Entries above 99 (e.g. 100) will display on the first two digits (Figure C6.4)
Floating	Auto fits the grid to fill the available space of a panel or tab. Works for grids inserted at the bottom of forms or tabs.
Fields	Fields that will be displayed in the selected grid.
Popup menu	String list that defines the right click context menu of the grid (Figure C6.4).
Fields With Totals	Numeric Columns (Field Names) that display totals in the footer of the grid (Figure C6.6).

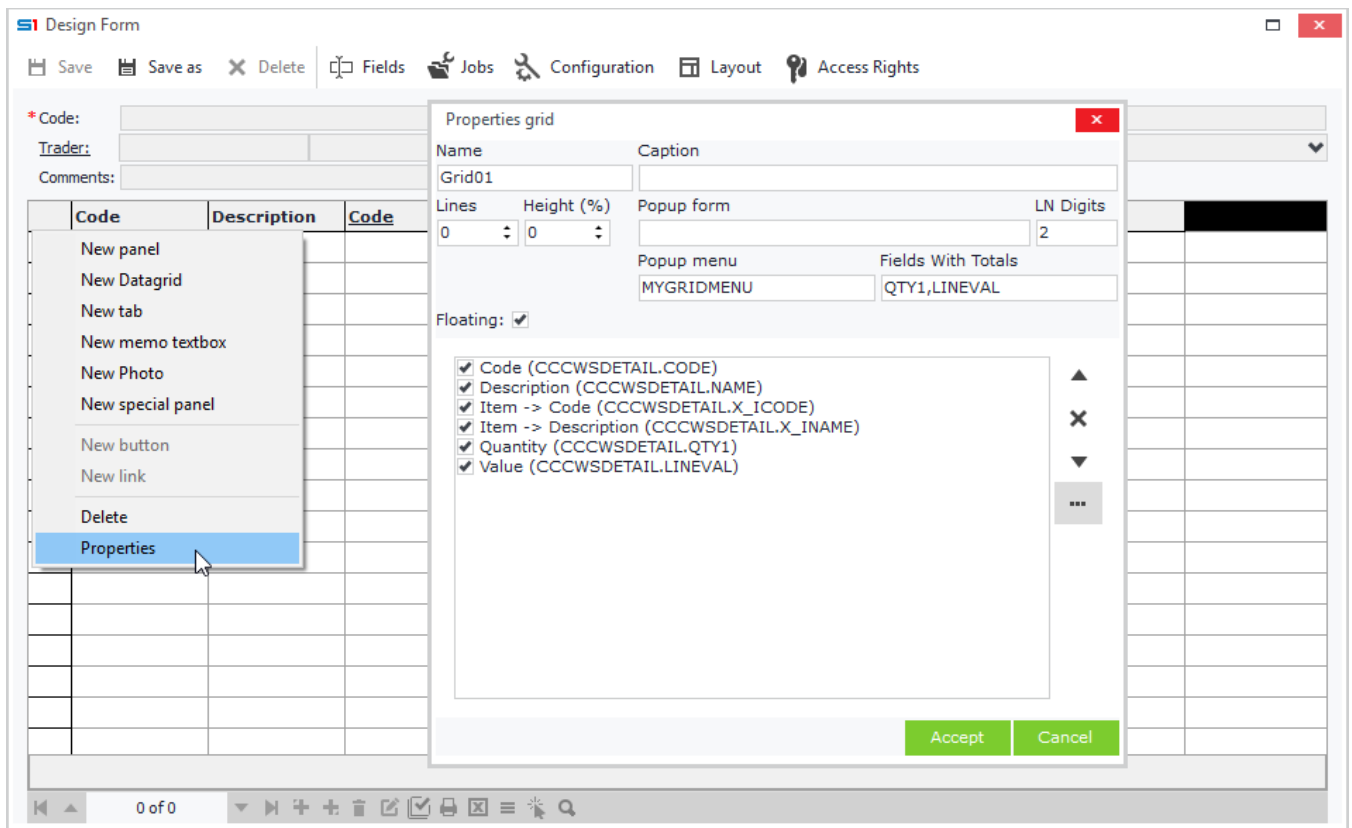


Figure C6.4

Sales Documents

Date: 05/01/2018 Branch: 1000 Head Offices Warehouse: 1000 Central

General data Delivery - Transfer International transactions Other data

Customer: 174 Dimitris Stathopoulos Status: Approval status: Approved
 Cust. branch: Movement type: 1000 For Sale Cancelled: No
 Salesperson: 00001 Smith Payment: Printed: No
 Currency: 100 Exchange rate: 1 TRP exchange rate: 1 VAT Status: Normal Check T.R.N.
 Comment:

Items Services

Code	Description	Qty 1	Price	Disc. ...	Value
99 50146	Soft1 ERP Service & Guarantee	10	4,00		40,00
10 50147	Soft1 ERP Projects	10	21,00		210,00
10 50148	Soft1 ERP Contract Management	10	12,00		120,00
10 50149	Soft1 ERP Individuals (Contacts)	10	12,00		120,00
10 50150	Soft1 ERP	10	23,00		230,00
10 50151	Soft1 ERP Group Sets	10	23,00		230,00
10 50153	Soft1 ERP Payroll (unlimited)	10	12,00		120,00
10 50162	Soft1 ERP Multilingual: English	10	12,00		120,00
10 50171	Soft1 CRM Sales	10	34,00		340,00
10 50172	Soft1 ERP Arrangements & Installments	10	321,00		3.210,00
10 50173	Car Oil CASTROL 10W-40 1L	10	1,00		10,00
11 50174	Car Oil MOTUL 10W-40 4L	10	23,00		230,00

13 of 110

Total qty: 955,00 Disc. 1(%): 0,00 Disc. 1: 0,00 Disc. 2(%): 0,00

Figure C6.5

Wire transfers - Documents

Date: Cash account: Series (suppliers): Series (creditors):

Job: Data Initialization (unpaid) Payment: No

Unpaid documents (Suppliers) Unpaid documents (Creditors) Totals per Supplier Totals per Creditor Totals per day Totals per month

Document	Document d	Age	Settlement ...	Delay	Balance	Value	Payme
00111	01/01/2018	156,00	01/01/2018	156,00	16.002,26	16.002,26	No
	01/01/2018	156,00	01/01/2018	156,00	25.798,19	25.798,19	No
	03/01/2018	154,00	03/01/2018	154,00	44.760,43	44.760,43	No
	13/01/2018	144,00	13/01/2018	144,00	44.760,43	44.760,43	No
058412	20/01/2018	137,00	20/01/2018	137,00	615,00	615,00	No
	02/02/2018	124,00	02/02/2018	124,00	25.798,19	25.798,19	No
					2.042.478,55	2.042.478,55	

1 of 96

Figure C6.6

C.7 Selector List

This control is a multi-column drop-down list (with magnifier) that displays data from database tables, inherited tables, database views and view tables, and stores their primary key (as defined in [SoftOne database designer](#)).

The selected record is displayed in two textboxes that are defined in the property "Selector fields" through [SoftOne database designer](#). The first textbox usually indicates the code of the selected record and the second one the description.

The columns of the drop-down list are the ones defined in the "Selector fields" property of database designer or the user defined columns created in the linked object (Figure C7.1).

Code	Name	Country	Currency
1590	Castrelian Brothers	FRANCE	EURO
110	Dustin Standt	Greece	EURO
117	Kalampoka Melina	Greece	EURO
111	Kelesidis Claus	Greece	EURO
126	Mizakis Stelios	Greece	EURO
102	Nick Panterben	Greece	EURO
906	Panayotou Petros	Greece	EURO

Figure C7.1

It is usually used in integer fields that are foreign keys to other tables, for example Customer (field TRDR) inside Sales Documents (table FINDOC) (Figure C7.2).

The link is made through the editor property of the field by adding the name of the linked database table, database view, view table or inherited table.

Extra features such as auto update of other fields can also be applied using any of the available editor attributes (See [Editor attributes](#)).

Figure C7.2

C.8 DropDown List (Memory Tables)

This control is a drop-down list that displays two text boxes horizontally (with arrow selector). Designing a combo box is the same as for selector list, which means that you enter the name of the table inside the editor property of a field (Figure C8.1).

The main difference from the selector list control is that it refers to memory tables and that it displays only the columns defined in the “Selector fields” property of the table through [SoftOne database designer](#).

This control is usually used in small integer fields to store the id of a record and link to a foreign memory table. The drop-down list box portion of the control is displayed when the user clicks the drop-down arrow.

Example of memory table drop down list is the field Country inside the table TRDR, which stores the id of Country in Customers entity (Figure C8.2).

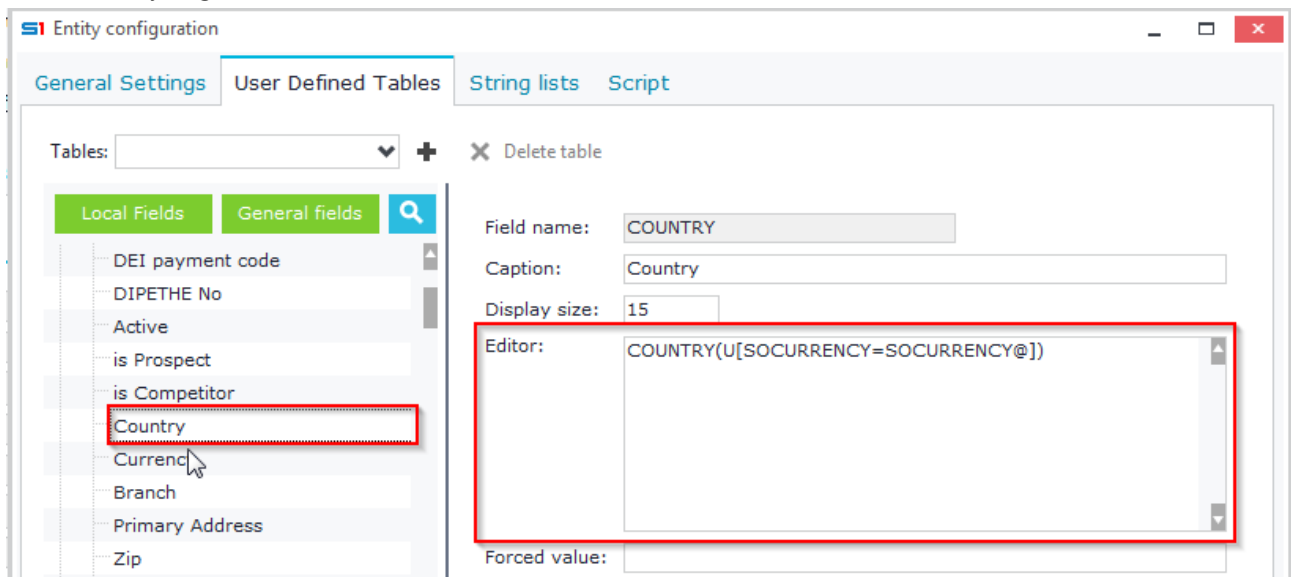


Figure C8.1

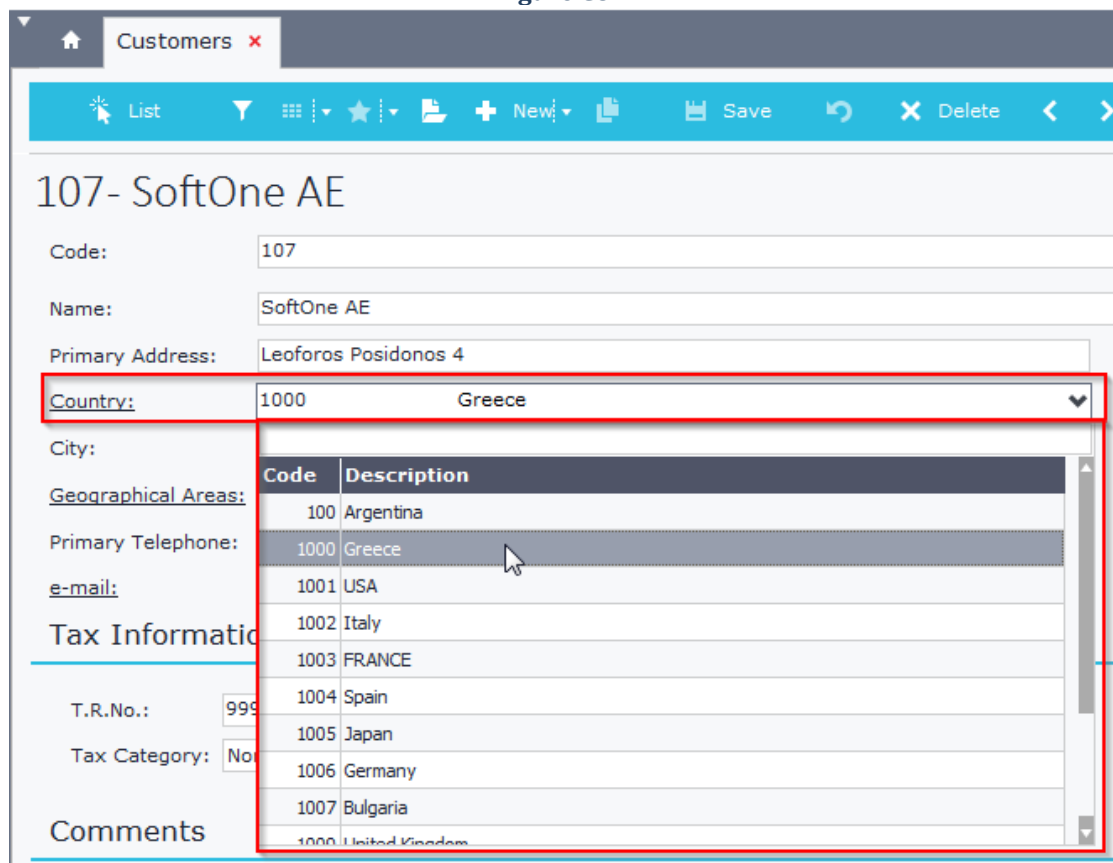


Figure C8.2

C.9 Text Picker

Text Picker Control is a drop-down list (with magnifier) that displays the selected data in one text box. When clicked it displays a one column drop down list of alphanumeric data and returns the selected record string.

The main difference from the other drop-down list controls is that it allows editing of the data inside the textbox, which does not affect the data of the drop-down list. It is usually used when you want to have a text list selector also allowing the users to enter free text.

This control is created through the editor property of a string field by entering the name of a table and using also the editor attributes **M,R[StringFieldColumn]**. This will return the defined column of the table (Figure C9.1).

Example of text picker control is the field COMMENTS inside the table FINDOC (Figure C9.2).

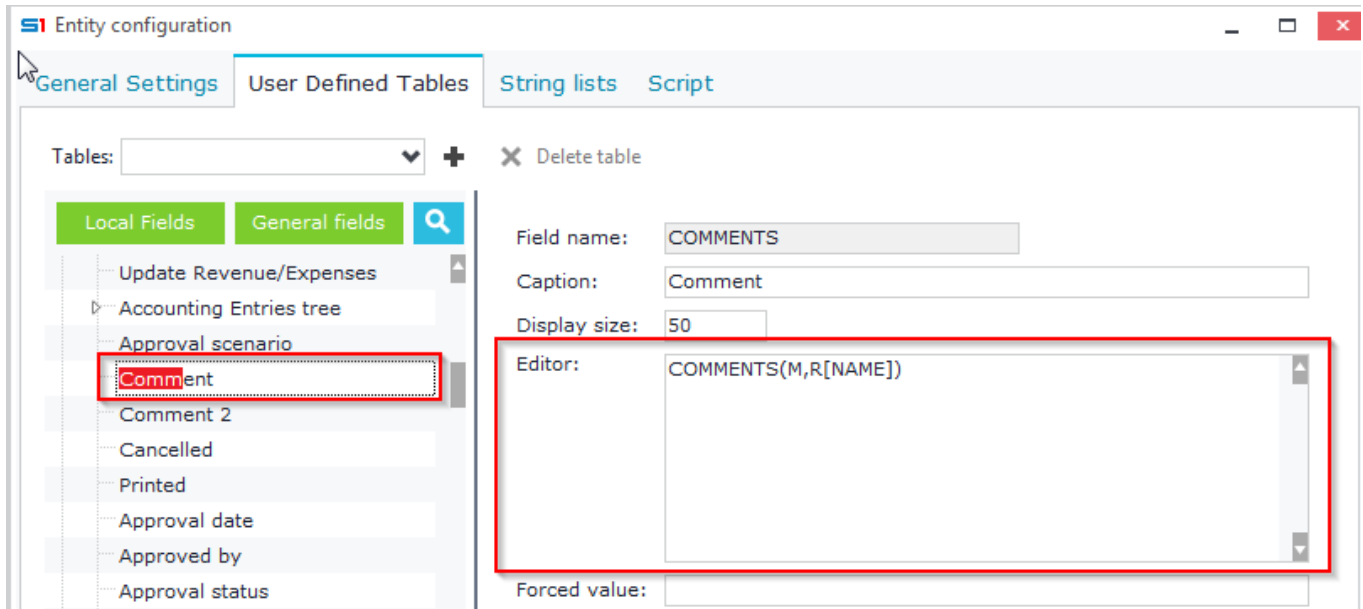


Figure C9.1

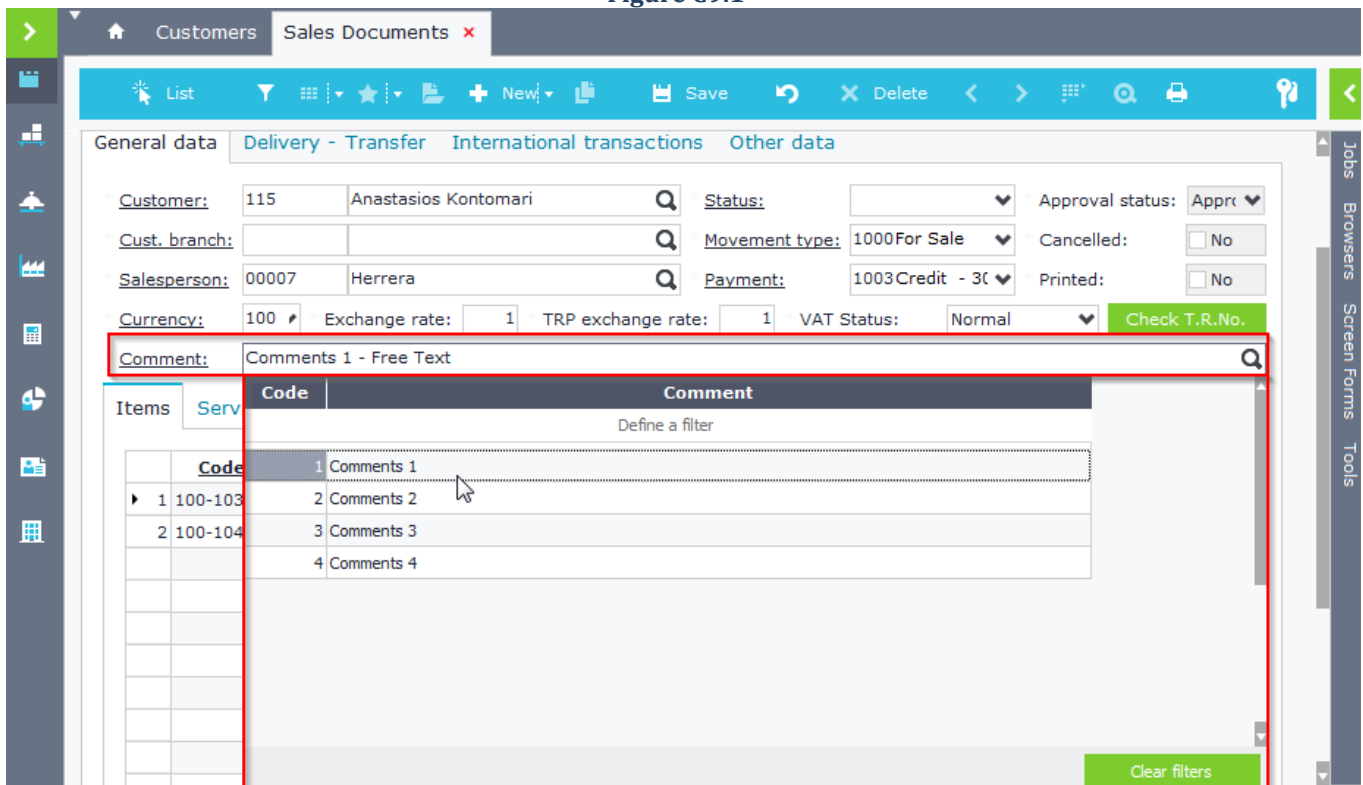


Figure C9.2

C.10 Checkbox

Checkbox is a three state (Yes / No / Null) control, usually associated with a small integer field. The selected state of the control returns value 1, while the unselected state returns 0 (Figure C10.1).

Figure C10.1

To create a checkbox control, you have to insert the command \$Y inside the Editor property of the preferred field. This can be done in form design mode by selecting the button "Configuration" and choosing the field from the table tree of the tab "User defined tables" (Figure C10.2).

Figure C10.2

Another way of displaying "Yes/No" data is through a combo box, using the editor \$YN.

Figure C10.3 displays the design of field MTRSUBSTITUTE.ISACTIVE that is added in a datagrid of Items form.

Figure C10.3

C.11 Multiple Checkbox

Multiple Check Box control displays one text box with a drop-down arrow selector. It is used in string fields and presents the data of a **memory table** or **string list** in a drop-down list control. It stores the selected record ids in comma delimited string (Figure C11.1).

	Password	Visit every
<input type="checkbox"/>	0	Sunday
<input checked="" type="checkbox"/>	1	Monday
<input type="checkbox"/>	2	Tuesday
<input checked="" type="checkbox"/>	3	Wednesday
<input type="checkbox"/>	4	Thursday
<input checked="" type="checkbox"/>	5	Friday
<input type="checkbox"/>	6	Saturday

☐ Select All Accept

Figure C11.1

This control is designed by entering the symbol # and the name of a memory table inside the editor property of a field. For string lists, use the symbols # \$ and the name of the string list (Figure C11.2).

Example of a multiple check box control is the field VISITEVERY inside the table TRDR, which presents a drop down list of the week days.

Entity configuration

General Settings | User Defined Tables | String lists | Script

Tables: + X Delete table

Local Fields | General fields

Payment from
Payment by
Payment every
Payment date
Week No.
Visit every
Warning Message
Group of expenses
Type of Customer
Scoring
Is also a supplier
Campaign

Field name: VISITEVERY
Caption: Visit every
Display size: 15
Editor: # \$ PAYDAY
Forced value:
Default value:

Figure C11.2

C.12 Image

Image control displays an image inside a form with the use of image data fields only. Field SODATA inside the table XTRDOCDATA is an example of image field. Figure C12.1 displays the use of the field XTRDOCDATA in an image control inside the Items object form.

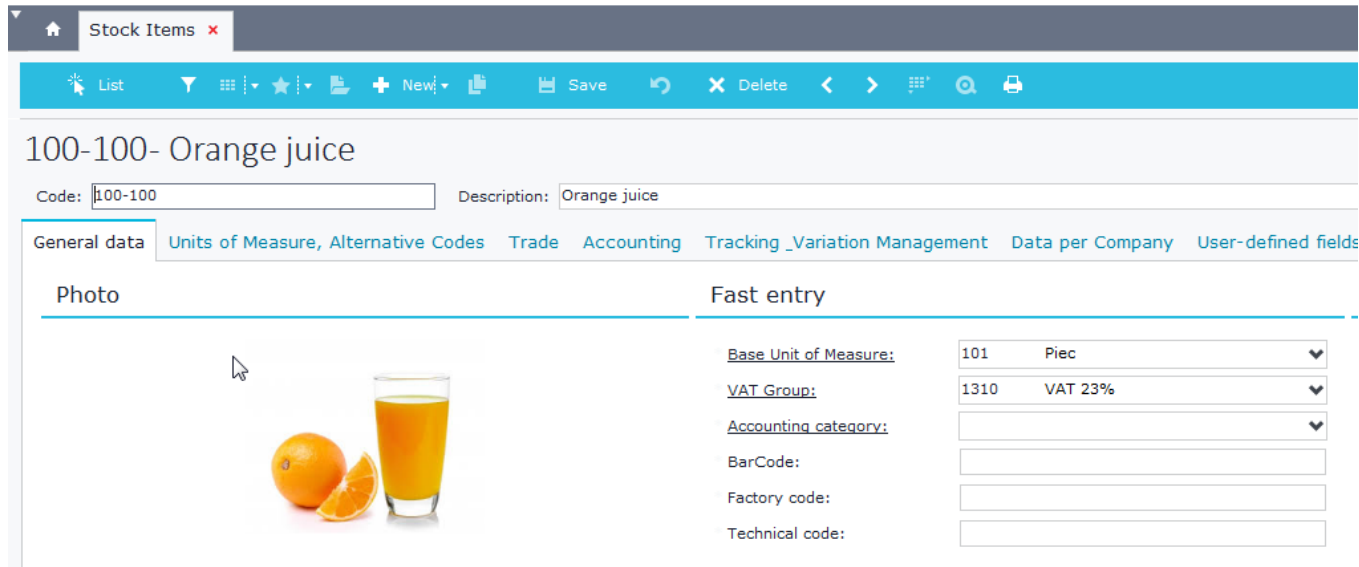


Figure C12.1

Image control can be created in design mode of a form, through the option “New picture” of the right click pop up menu. Inside the properties window of the image, simply drag and drop the preferred binary field (Figure C12.2).

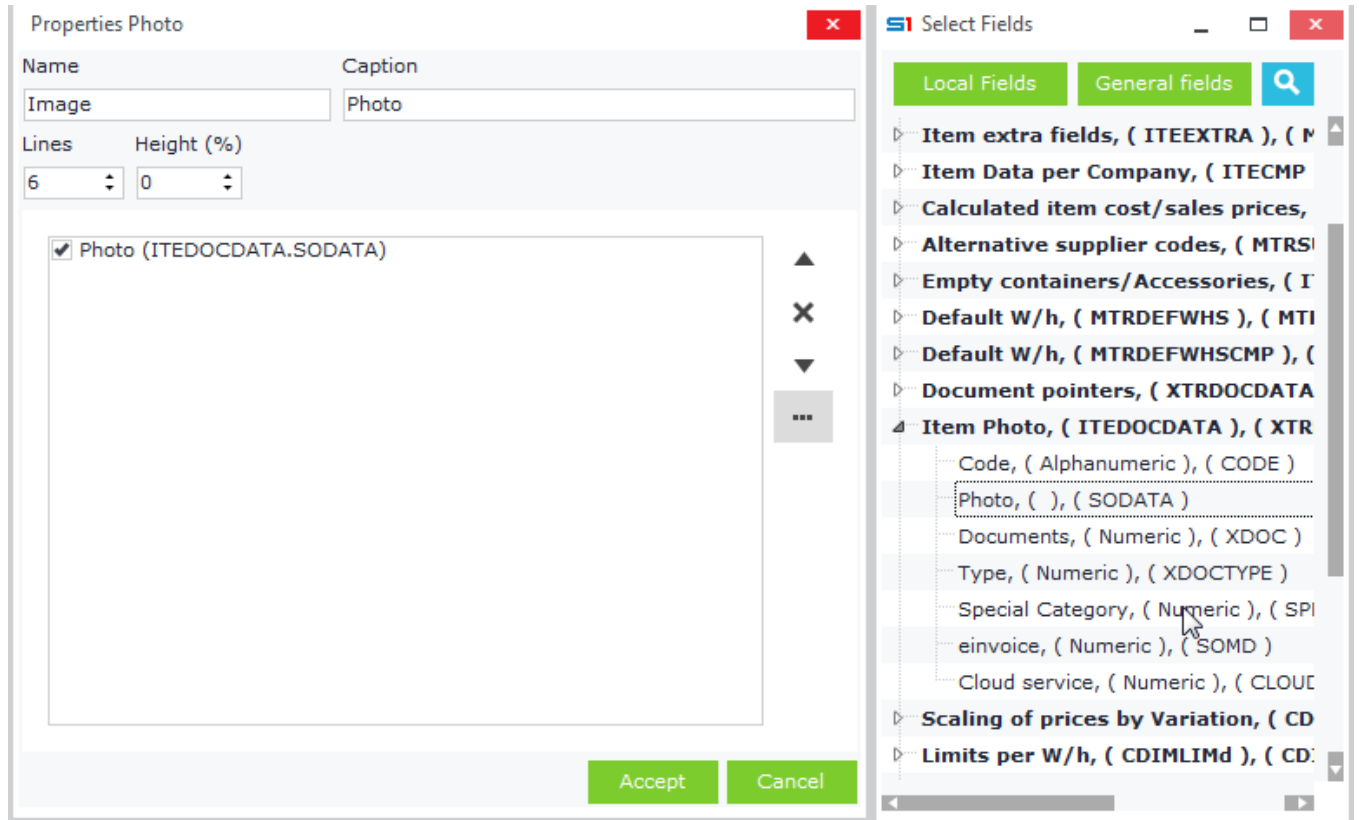


Figure C11.2

C.13 HTML Editor

This control displays an editor for html code that is stored in an image field. Example of this control can be found in field SOMAIL of the object Email (Figure C13.1).

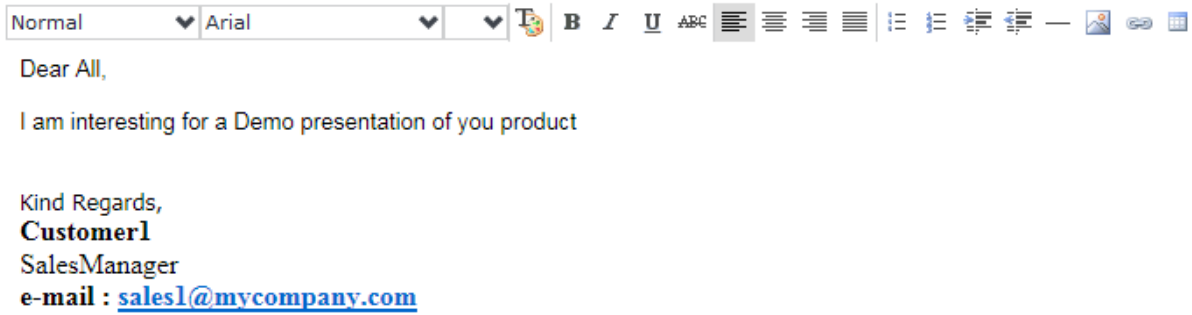


Figure C13.1

HTML editor is designed in form by creating a new special panel through the right click pop up menu. Then, inside the properties of the special panel you have to enter the command **HTMLEditor** inside the "Popup form" property and also the expression **USERPARAMS=TableName.BinaryFieldName** in the "Editor Params" property (Figure C13.2). "BinaryFieldName" is the name of the image field, that you have created in the table "TableName" from SoftOne database designer (Figure C13.3).

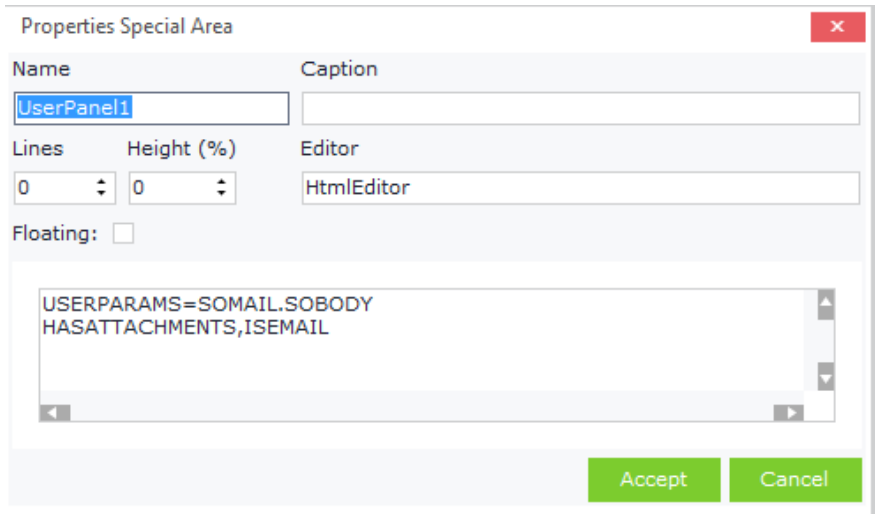


Figure C13.2

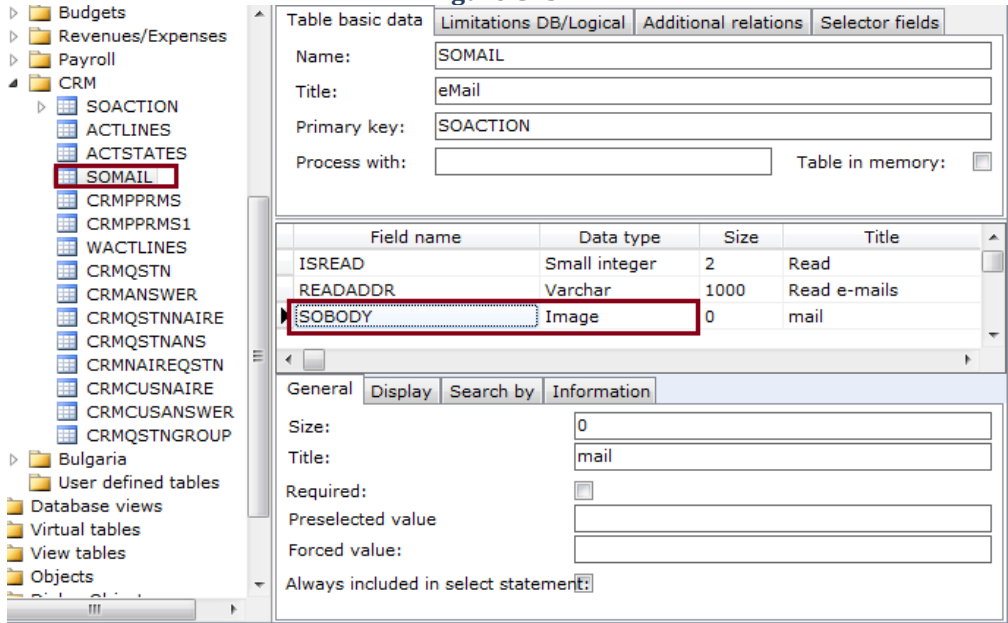


Figure C13.3

C.14 SoftOne Spreadsheet

SoftOne Spreadsheet is a spreadsheet displayed by combination of keys Alt + S on fields of the application. The specific control allows you to use simple calculations within a spreadsheet environment and insert the result (selected cell) within the field through which the tool was called (Figure C14.1).

By default, the spreadsheets in all fields are empty. However, it is possible to create default spreadsheets which will be matched to specific fields of application modules. All you need to do is save the desired spreadsheet file in the application folder with htm extension and provide as file name the name of the field (Object_Field.htm).

For example, if you want to match a file in the Barcode field of the item, you must save the file in the application folder under the name ITEM_CODE1.htm. Figure C14.2 depicts the spreadsheet displayed with Alt + S in the BARCODE field of the items.

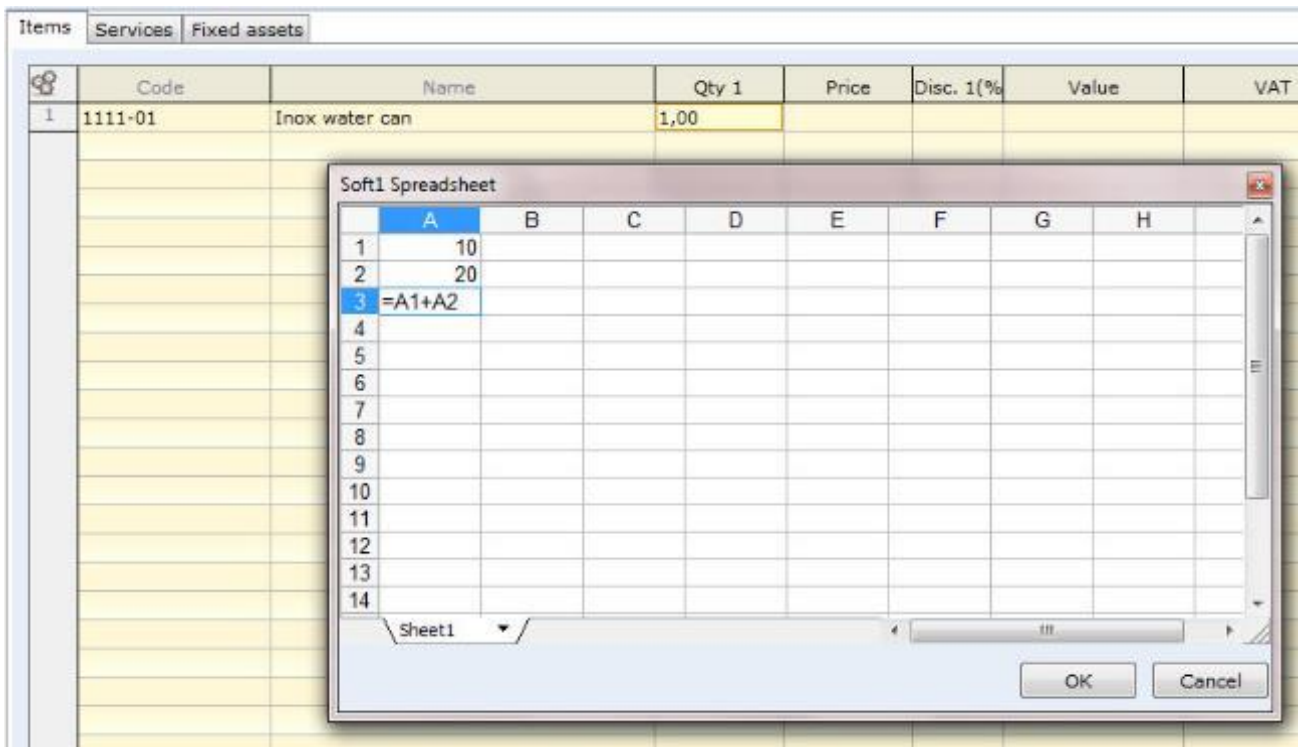


Figure C14.1

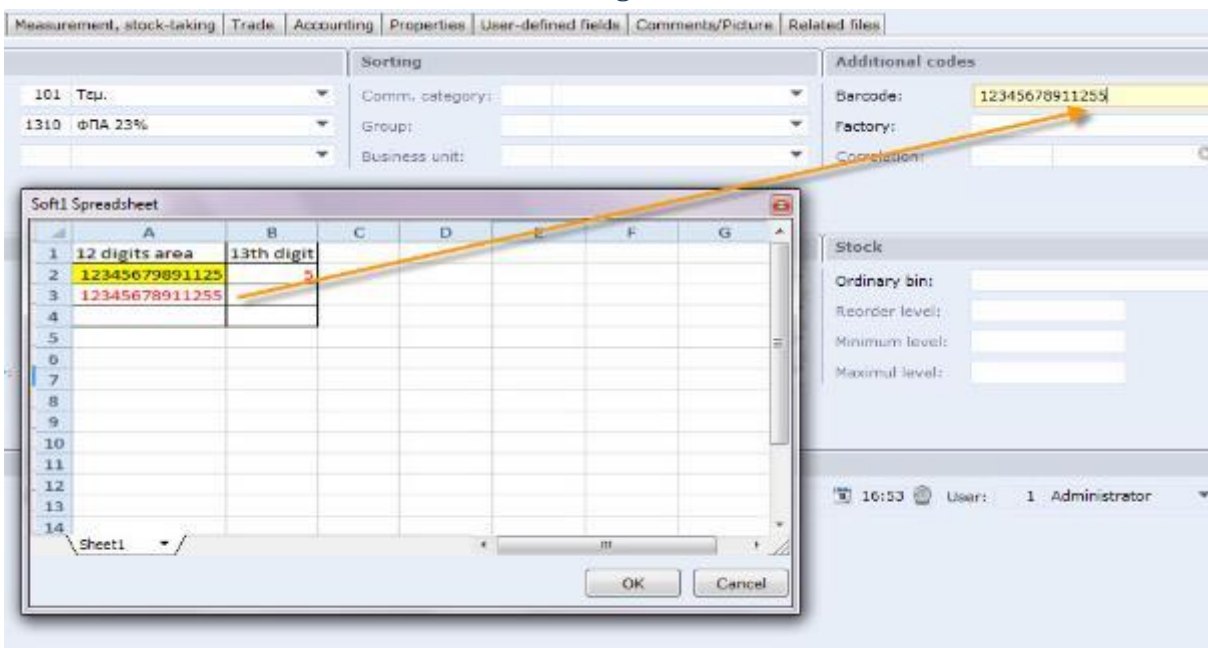


Figure C14.2

C.15 Datetime

This control is used for displaying datetime fields. Fields can be displayed in four different ways.

Date only

This control, which is the default one, displays a datetime field using a drop down calendar (Figure C15.1). Nothing is entered inside the editor property of the field (Figure C15.2).

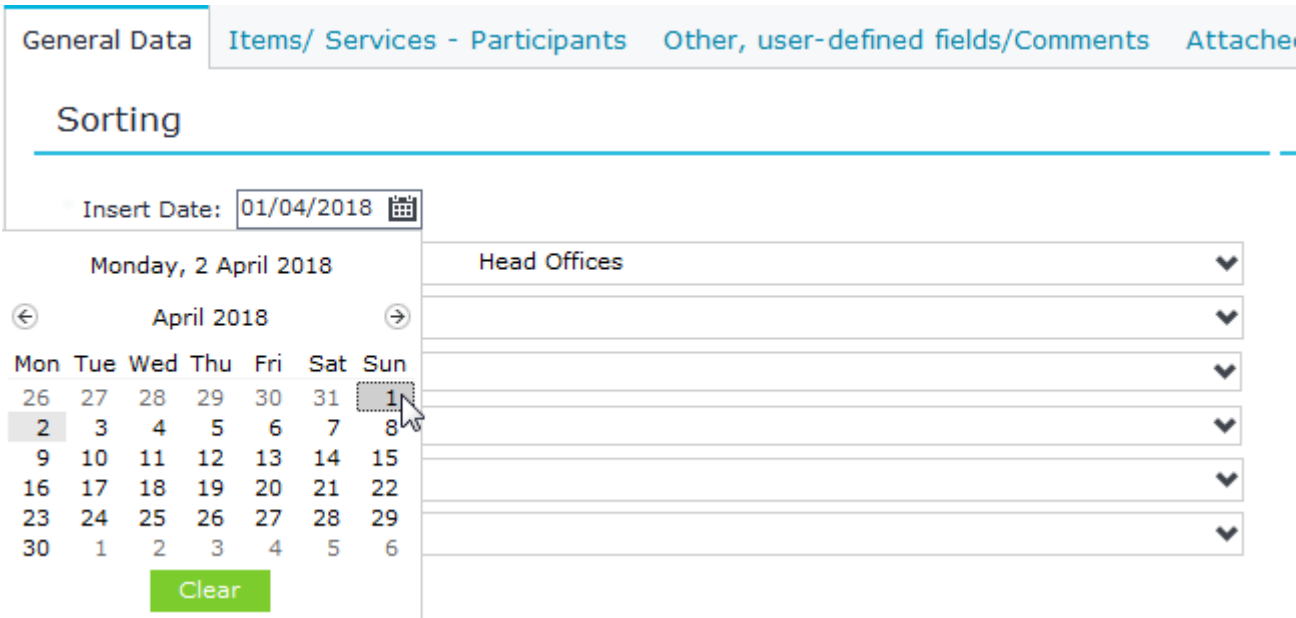


Figure C15.1

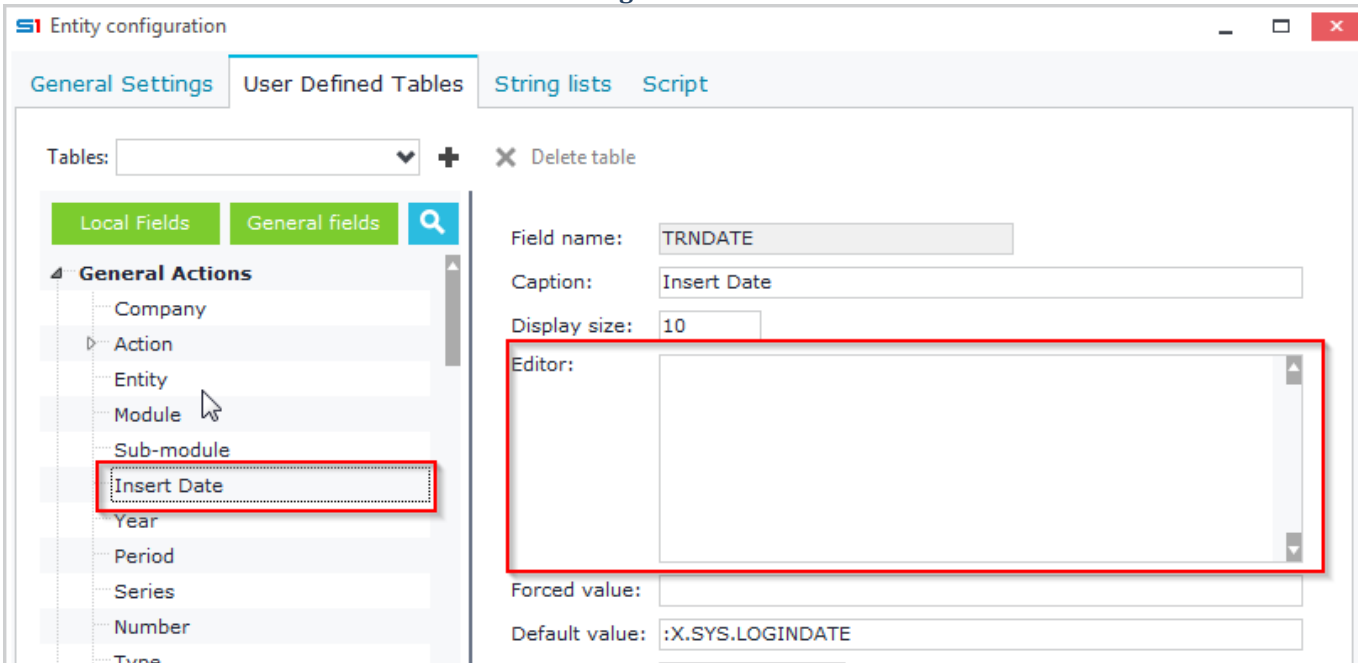


Figure C15.2

Date and time

This control displays a datetime field using a drop down calendar and its time using an updown control (Figure C15.3). The command **\$DT** must be entered inside the editor property of the field (Figure C15.4).

Figure C15.3

Figure C15.4

Time only

This is an UpDown control that displays only the time of a datetime field (Figure C15.5). The command **\$Time** must be entered inside the editor property of the field (Figure C15.6).

Figure C15.5

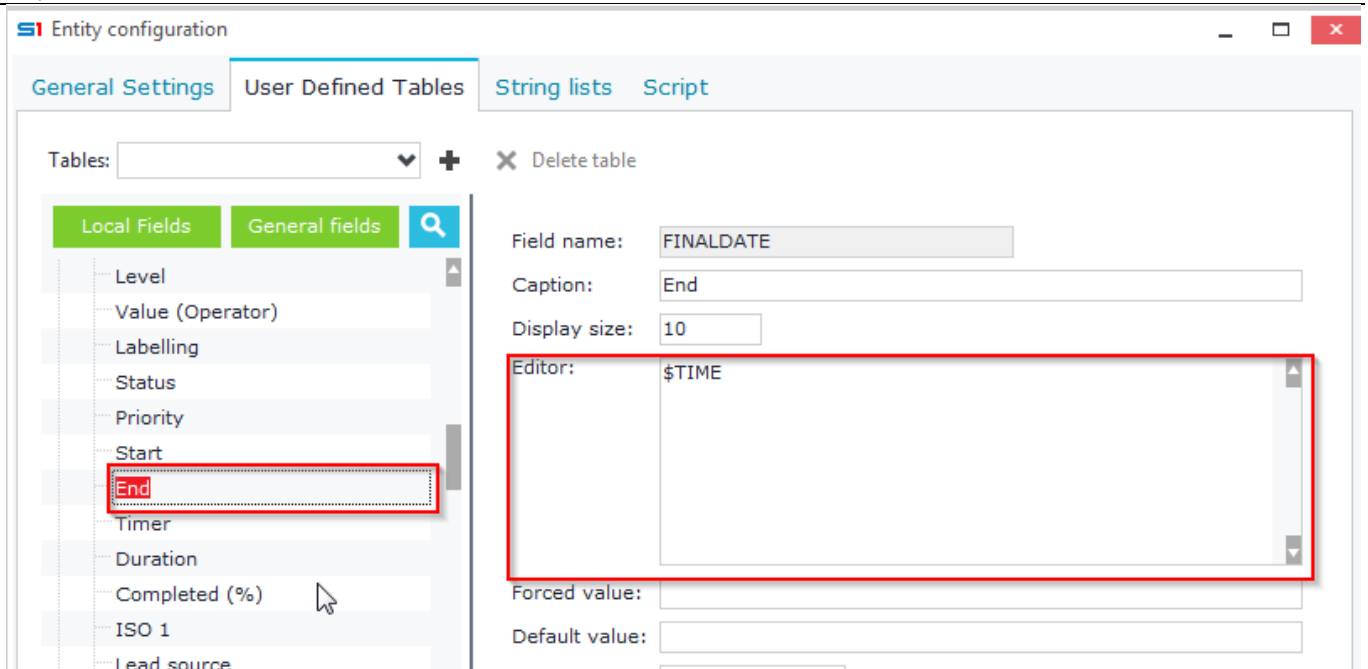


Figure C15.6

Time Range

This is a textbox control that displays the difference between two datetime fields in hh:mm:ss format (Figure C15.7). The command **\$TimeRange** must be entered inside the editor property of the field (Figure C15.8).

The screenshot shows a form titled 'Time'. It contains two rows of date and time fields. The first row has 'Start:' with a date field '01/04/2018' and a time field '10:10'. The second row has 'End:' with a date field '02/04/2018' and a time field '11:34'. To the right of these fields, there is a 'Duration:' field with the value '07:45:00"' and a 'Completed (%)' field. The 'Duration' field is highlighted with a red box.

Figure C15.7

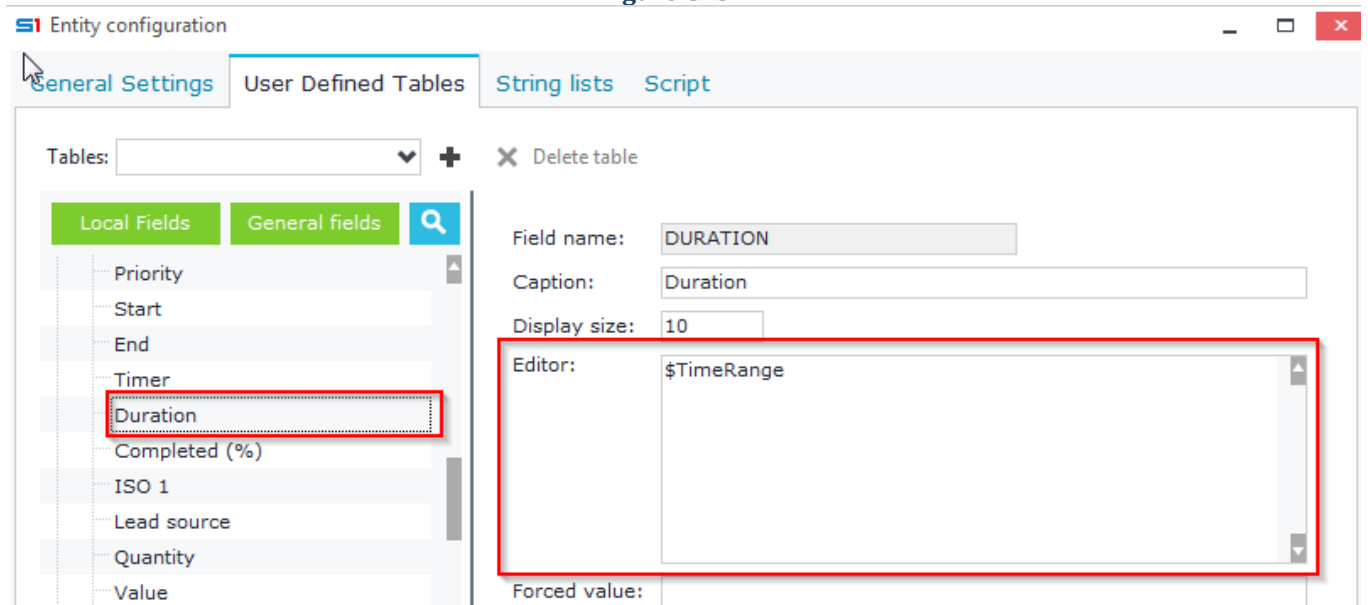


Figure C15.8

C.16 Embedded QuickView

It is a read only control that displays multiple data (columns) of a browser that exists in a different entity of the current form's entity.

Figures C16.1 and C16.2 display data and image of the selected row items. These data exist in the browser 'ItemData' that is created in 'Items' entity as in Figure C16.3.

You can also use this control to display data from form fields (not only datagrid rows) such as Customer financial data (columns inside a browser of 'Customers' module).

The screenshot shows a software interface with a top toolbar containing icons for List, Filter, Star, New, Save, Delete, and navigation arrows. Below the toolbar is a tabbed interface with 'General data', 'Delivery - Transfer', 'International transactions', and 'Other data'. The 'Delivery - Transfer' tab is active, displaying a form with fields for Customer (168 Charitopoulos Manolis), Salesperson (00004 Houston), Approval status (Approved), Canceled (No), Printed (No), Cust. branch, Status, Movement type (1000 For Sale), Payment, Currency (100), Exchange rate (1), TRP exchange rate (1), VAT Status (Normal), and a Comment field. A green button labeled 'Check T.R.No.' is visible. To the right of the form is a red-bordered box containing an image of a black and orange digital watch.

Below the form is a tabbed interface with 'Items' and 'Services'. The 'Items' tab is active, displaying a table with columns: Code, Description, Qty 1, Price, Disc. ..., and Value. The table contains three rows: 1 20013 Dive Watch (Qty 3, Price 777,40, Value 2.332,20), 2 20002 Pulsar watch (Qty 4, Price 318,50, Value 1.274,00), and 3 20005 Artillery Compass (Qty 15, Price 52,00, Value 780,00). A red box highlights the second row (2 20002 Pulsar watch). To the right of the table is a red-bordered box containing a list of financial data: Wholesale price (318,50), Retail Price (330,75), Day balance 1 (849), Balance cost (21.101), Expected (0), and Reserved (8).

Figure C16.1

The screenshot shows the same software interface as Figure C16.1, but with the 'Services' tab selected in the 'Items' section. The 'Delivery - Transfer' form remains the same. The 'Services' tab displays a table with columns: Code, Description, Qty 1, Price, Disc. ..., and Value. The table contains three rows: 1 20013 Dive Watch (Qty 3, Price 777,40, Value 2.332,20), 2 20002 Pulsar watch (Qty 4, Price 318,50, Value 1.274,00), and 3 20005 Artillery Compass (Qty 15, Price 52,00, Value 780,00). A red box highlights the third row (3 20005 Artillery Compass). To the right of the table is a red-bordered box containing a list of financial data: Wholesale price (52,00), Retail Price (54,00), Day balance 1 (958), Balance cost (28.282), Expected (0), and Reserved (1). Above the table, a red-bordered box contains an image of a black and silver compass.

Figure C16.2

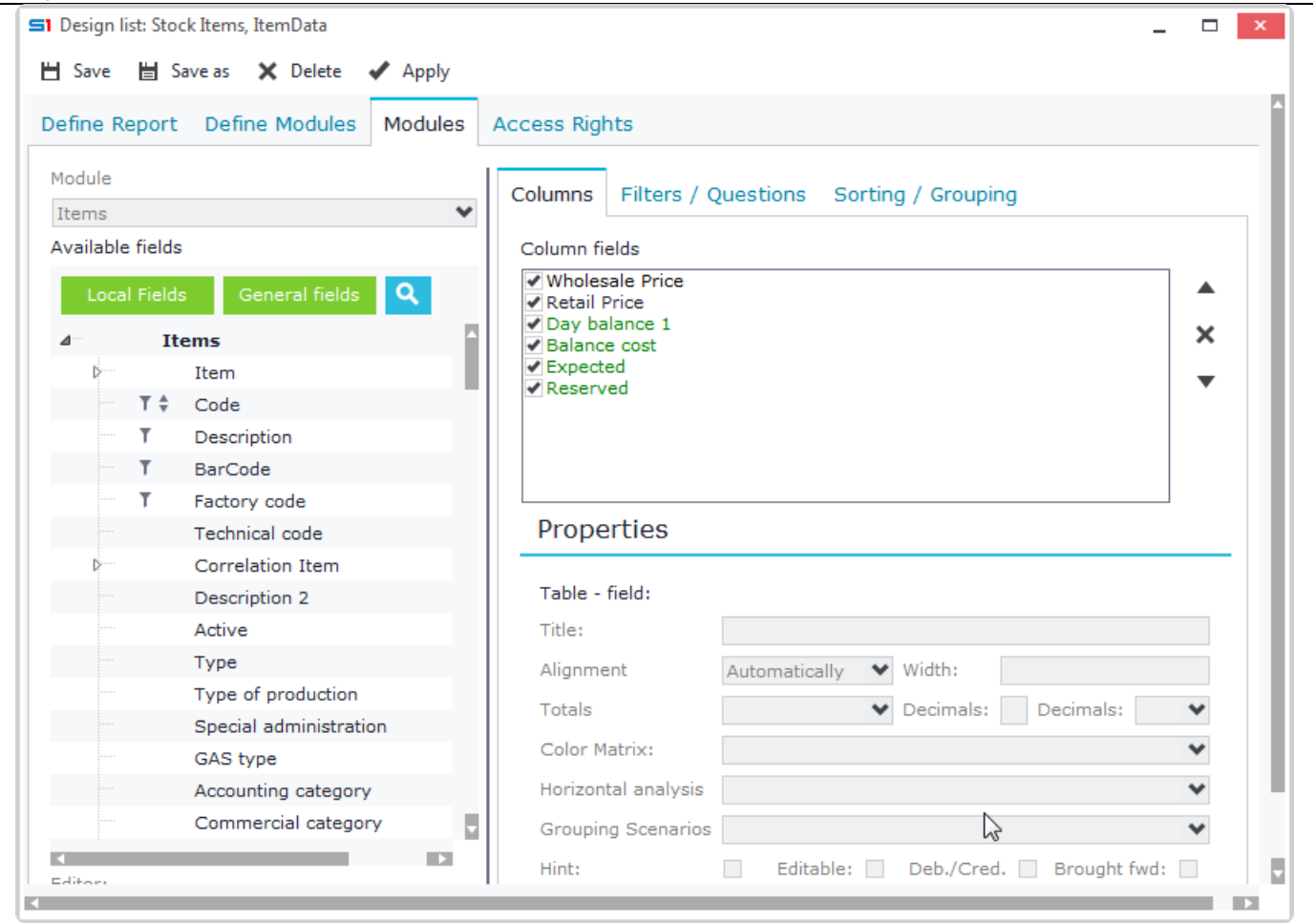


Figure C16.3

Design

Use a special area, enter the command **XQuickView** in Editor property and then the following commands as in Figure C16.4:

```
[PARAMS]
MODULE=ITEM
QUICKVIEW=ItemData
FIELD=ITELINES.X_CODE
```

The table below summarizes the parameters needed to create an embedded quickview.

PARAMETER	DESCRIPTION
MODULE	Name of the entity (module) that the data exist.
QUICKVIEW	Name of the browser. For images use the command IMAGE.
FIELD	Name of the field (inside current form) that will be used to retrieve the data. (e.g. For Customers inside Sales Documents use: SALDOC.TRDR)

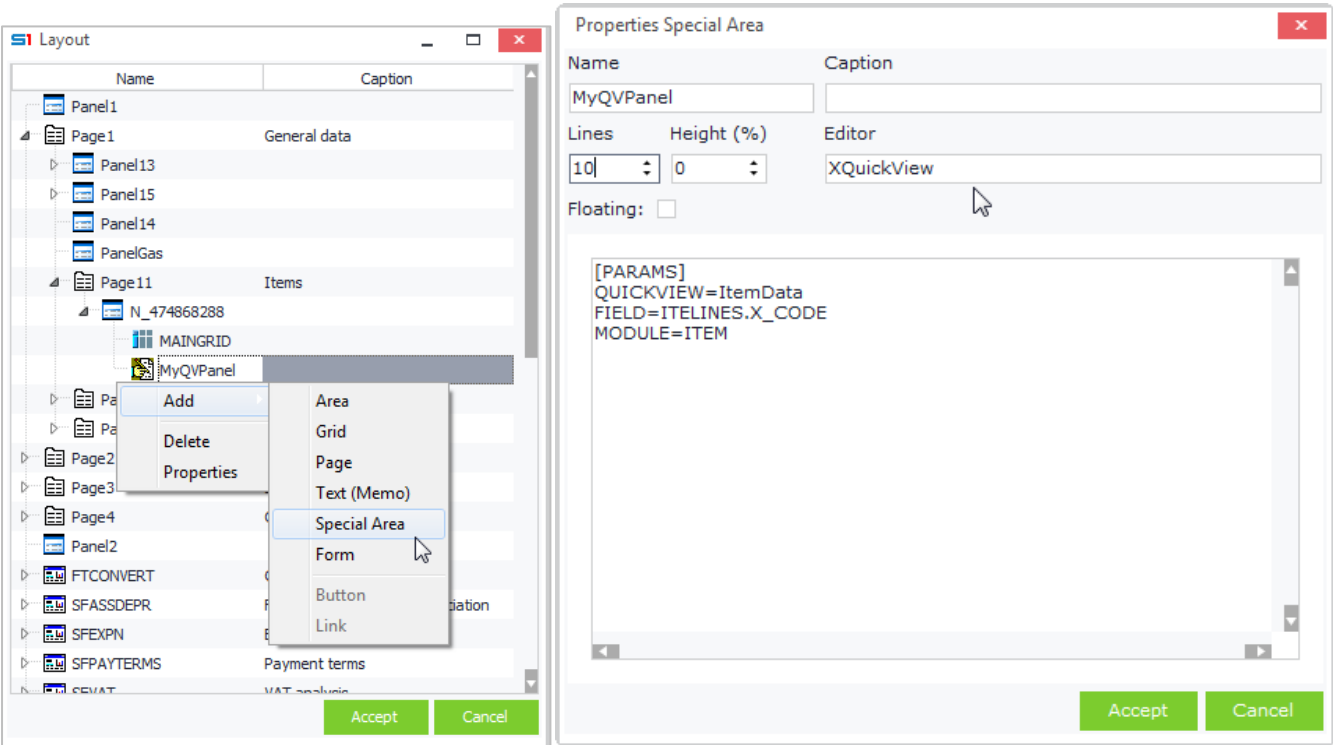


Figure C16.4

Images are displayed using the parameter **QUICKVIEW=IMAGE** (Figure C16.5):

[PARAMS]
MODULE=ITEM
QUICKVIEW=IMAGE
FIELD=ITELINES.X_CODE

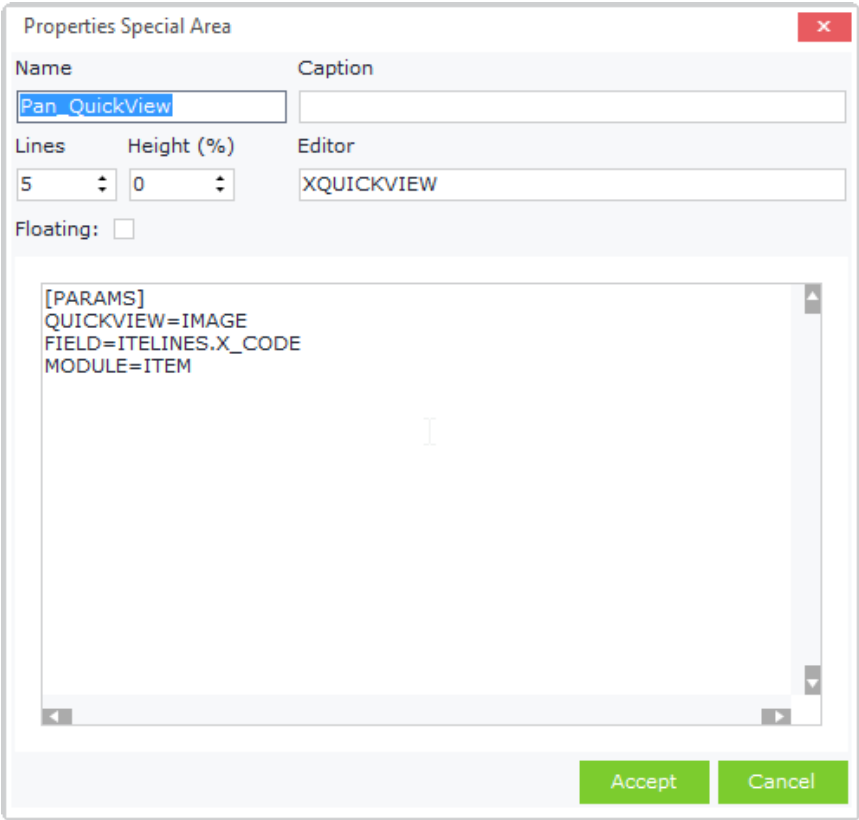


Figure C16.5

D. Dialog Controls

D.1 File Picker

This control displays a file picker dialog window. It returns the full path of the selected file (Figure D1.1). Varchar fields are used to display this control in a form of an object, by adding the command **\$FILENAME** in their editor property (Figure D1.2).

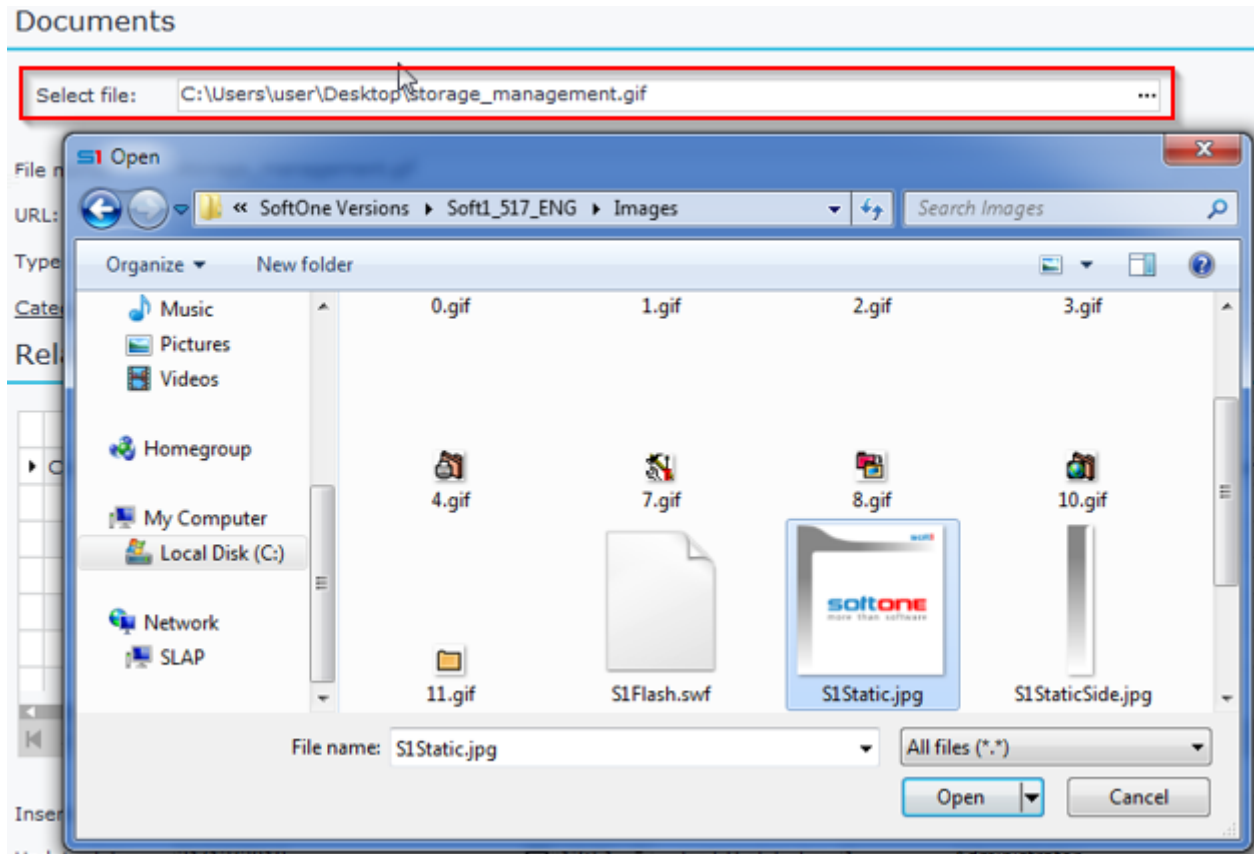


Figure D1.1

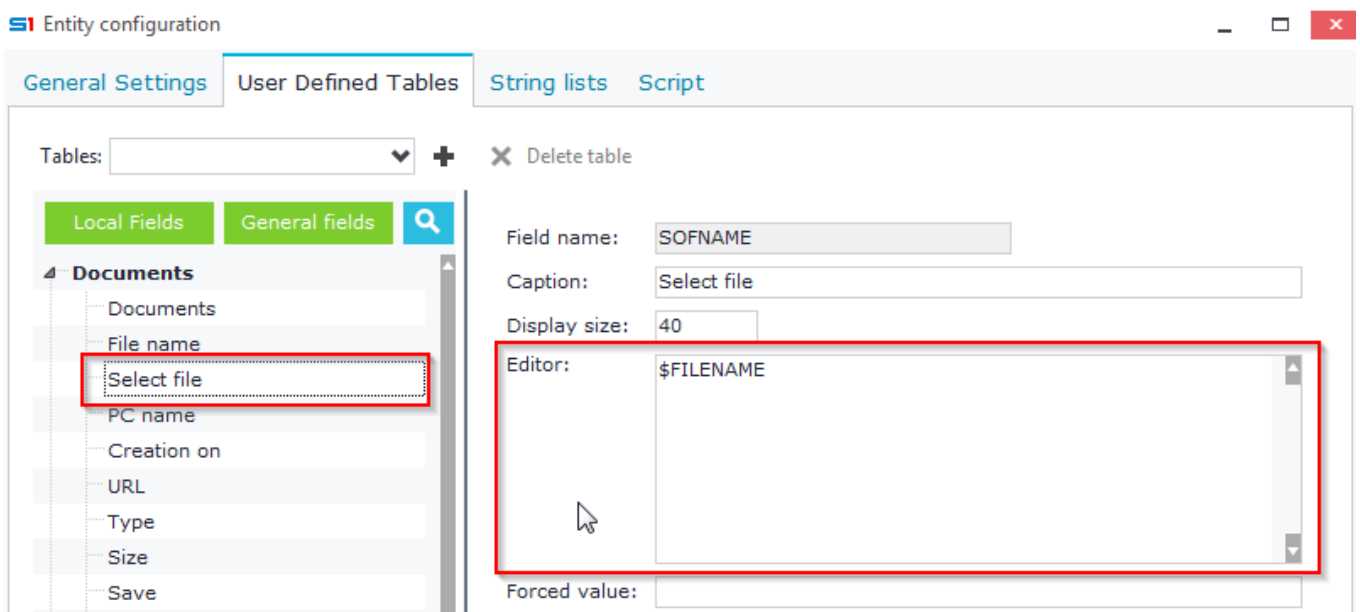


Figure D1.2

D.2 Color Picker

This control displays a color window and returns the selected values (Figure D2.1). Varchar fields are used to display this control in a form of an object, by adding the command **\$COLOR** in their editor property (Figure D2.2).

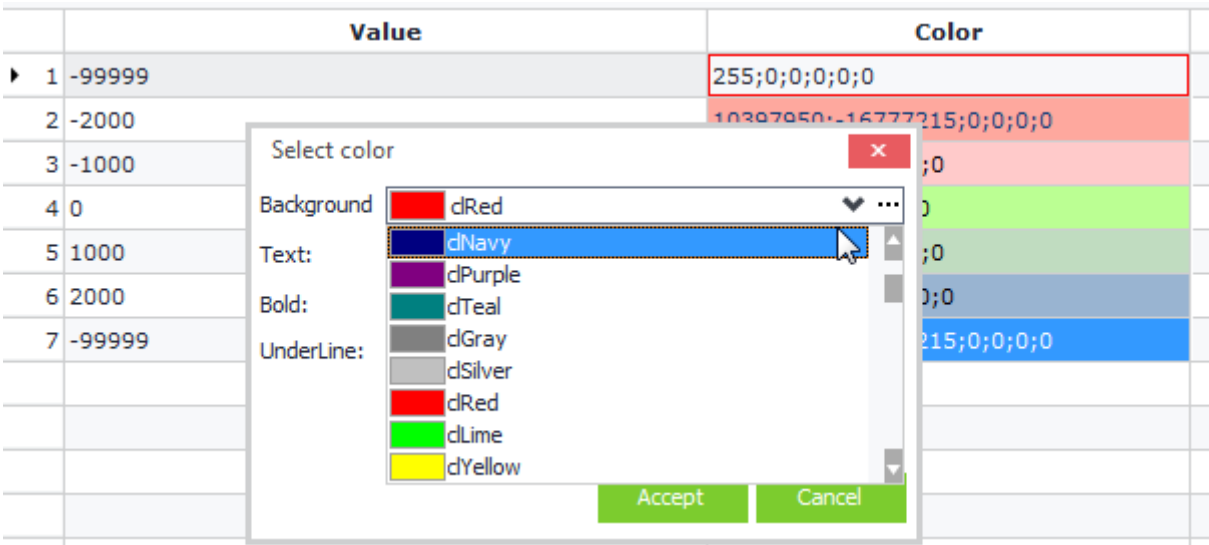


Figure D2.1

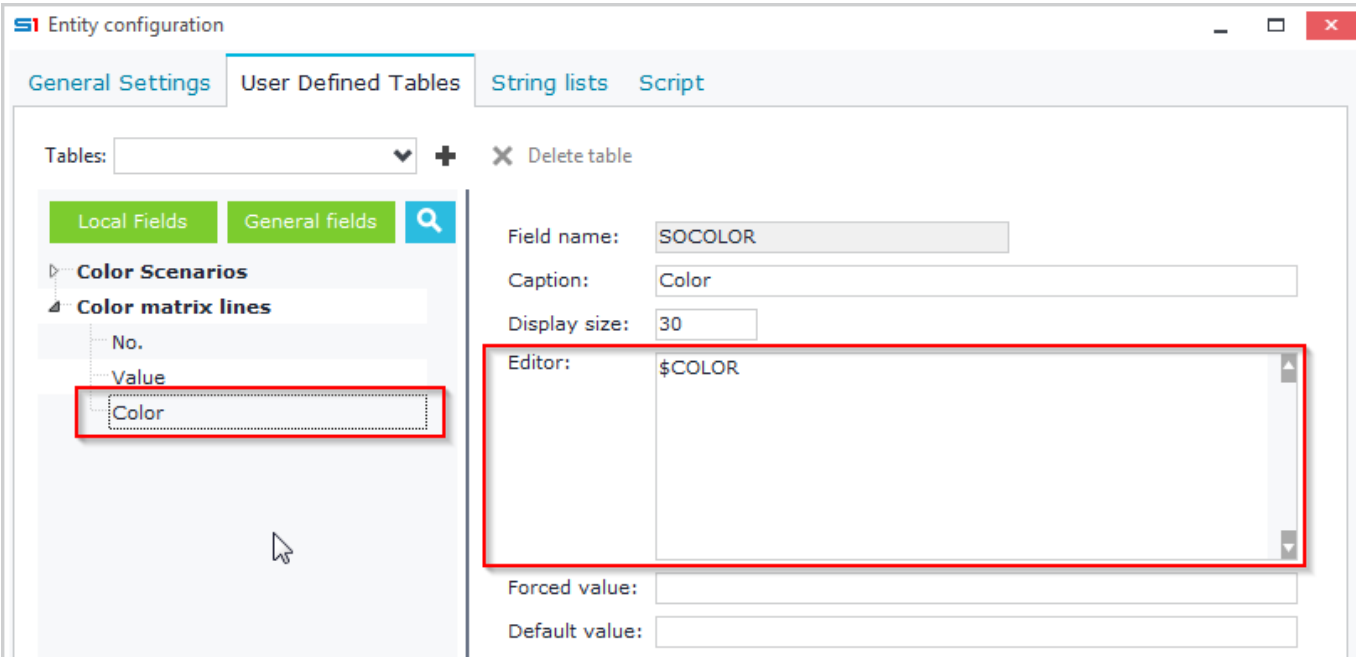
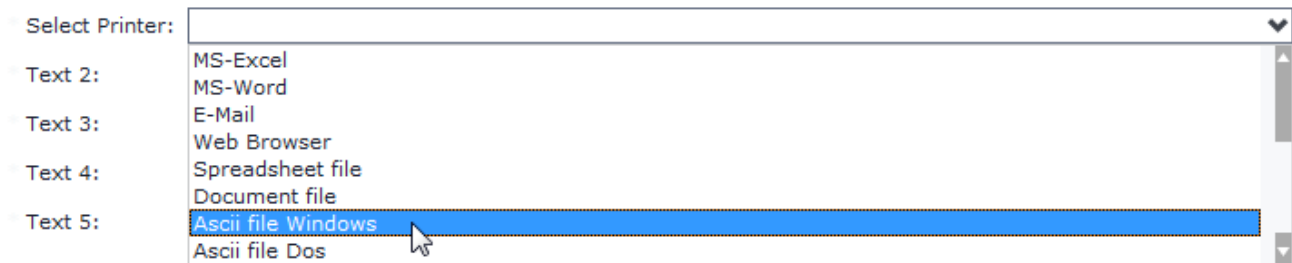


Figure D2.2

D.3 Printers Picker

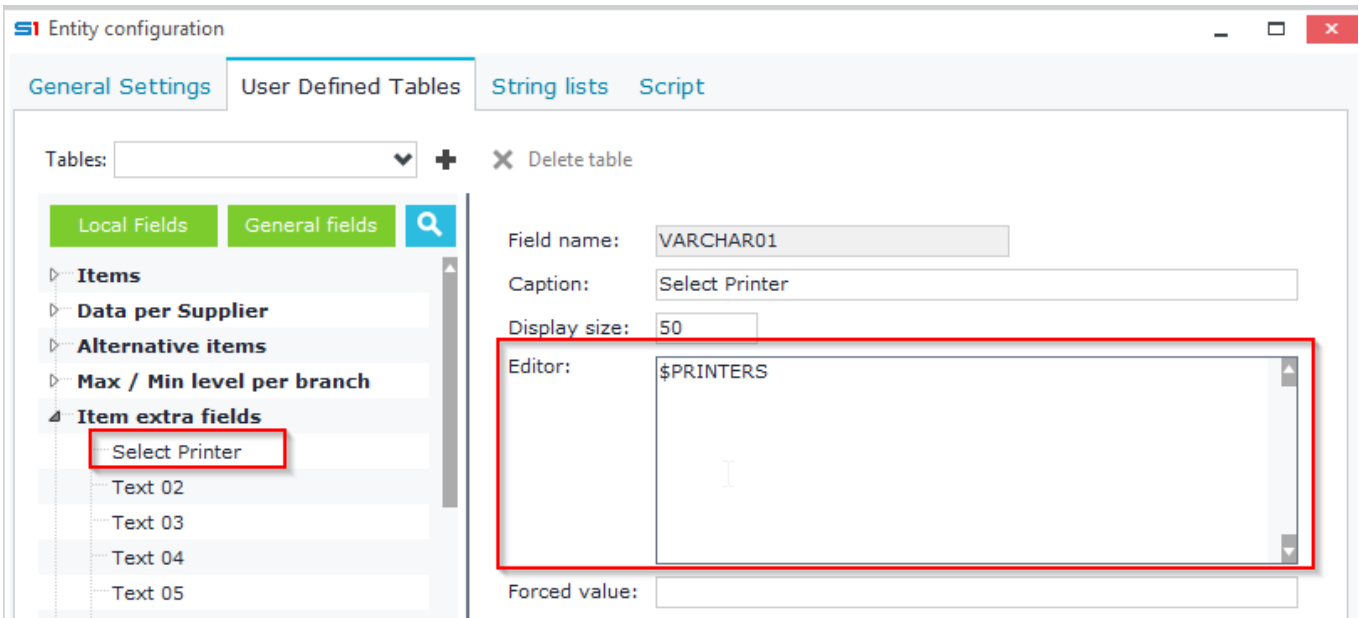
This control displays a printer dialog window and returns the selected printer string (Figure D3.1). Varchar fields are used to display this control in a form of an object, by adding the command **\$PRINTERS** in their editor property (Figure D3.2).

Text



The screenshot shows a 'Select Printer' dialog box. It has a 'Select Printer:' label and a list of printers. The list includes: MS-Excel, MS-Word, E-Mail, Web Browser, Spreadsheet file, Document file, Ascii file Windows (highlighted in blue), and Ascii file Dos. A mouse cursor is pointing at the 'Ascii file Windows' option.

Figure D3.1



The screenshot shows the 'Entity configuration' window. The 'User Defined Tables' tab is selected. On the left, under 'Item extra fields', the 'Select Printer' field is highlighted with a red box. On the right, the configuration for this field is shown. The 'Field name' is 'VARCHAR01', the 'Caption' is 'Select Printer', and the 'Display size' is '50'. The 'Editor' property is set to '\$PRINTERS' and is also highlighted with a red box. The 'Forced value' is empty.

Figure D3.2

D.4 Time Scheduler

This control displays the SoftOne scheduler window and returns the selected values in semi-colon text (Figure D4.1). String fields are used to display this control in a form of an object, by adding the command **\$RECCURENCE** in their editor property (Figure D4.3).

Figure D4.1

Figure D4.2

E. Command Controls

E.1 Buttons

Command Buttons can be created in any panel of a form and run when users click on them. The event handler that controls the actions performed when the command button is clicked is **EXECCOMMAND(cmd)**, where cmd refers to the id of the button. Handling the "On Click" event is done through form script code (JavaScript / VBscript).

Buttons can be used to execute any of the following jobs:

- **Display / Open a sub form in an object**

Use the sub form's name (e.g. Mypopupform) and the command **XCMD:Mypopupform** to open a sub form inside an object form.

- **Display / Open an object**

Use the name of the object and the command **XCMD:ObjectName** to open an object.

Enter the command in the editor property of a numeric field.

Alternatively you can use form script code that handles the button click through the function EXECCOMMAND(cmd) and then use the EXEC function to open the object.

The following command opens the customer object and locates a the record defined from the field SALDOC.TRDR. **X.EXEC('XCMD:CUSTOMER[AUTOLOCATE='+SALDOC.TRDR+',FORM=Myform]')**

- **Display a relative job of an object**

Relative job commands can be found in Annex [Relative Job Commands](#).

- **Execute custom form script code**

Use buttons to execute code written in VBscript or JavaScript inside object forms.

- **Execute SBSL script**

Use buttons to execute custom functions and code written in SBSL script.

Buttons are inserted in forms using one of the following ways (command RUNB or database fields):

Command RUNB_buttonID=buttonCaption (Recommended)

Right click the "Fields" property of a panel, select the option "Add field" from the pop up menu and enter the command RUNB_ButtonID=MyButton (Figure E2.3). Buttons are displayed as in Figure E2.4.

Hyperlink buttons are added using the command **RUN_ButtonID=ButtonCaption**

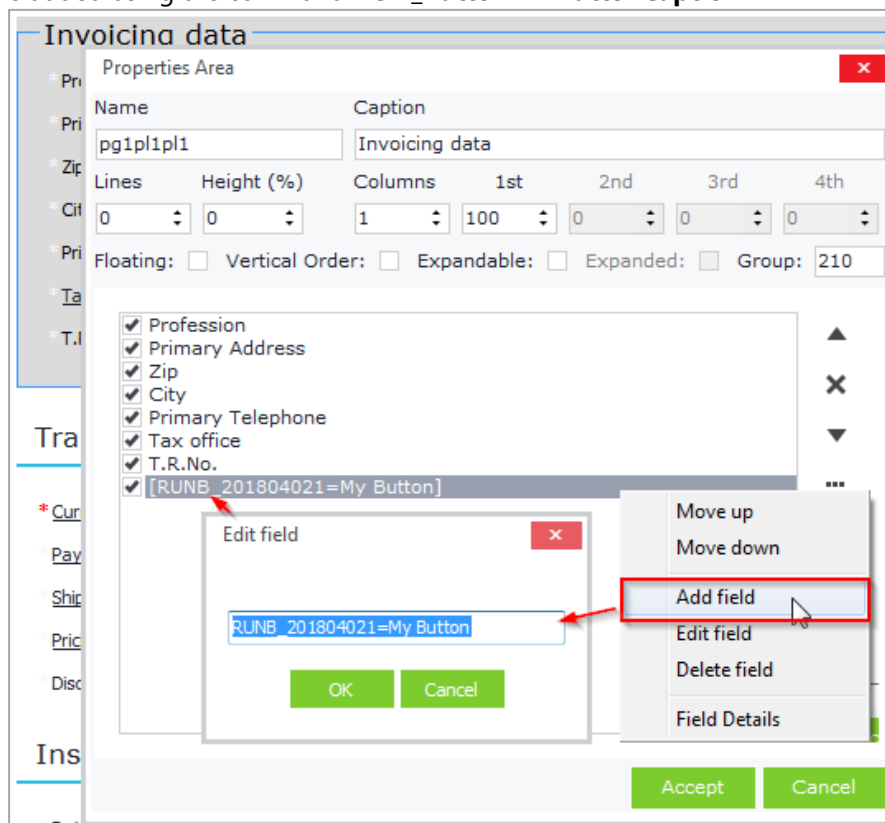


Figure E2.3

* Code: * Name:

Basics | Contact | Commercial data | Financials | Accounting | User-defined fields | Comments/

Invoicing data

Profession:

Primary Address:

Zip: ▼

City:

Primary Telephone:

Tax office: ▼

T.R.No.:

Figure E2.4

Database numeric fields

- Add a numeric field inside a panel
- Add the command **XCMD:xxx** (xxx=id of the button defined by the user) in the Editor property of the field (Figure E1.1). Note that the id of the button should be a number larger than 100.000, in order not to conflict with SoftOne internal buttons.
- Enter the number **1** inside the Decimals property of the field (Displays the **button** control). The field is displayed as **hyperlink** if you leave the Decimals property blank.

Entity configuration

General Settings | User Defined Tables | String lists | Script

Tables: + X Delete table

Local Fields | General fields

Text 01
Text 02
Text 03
Text 04
Text 05
MY BUTTON
Number 02
Number 03
Number 04
Number 05
Table 01
Table 02
Table 03

Field name: NUM01
Caption: MY BUTTON
Display size: 14
Editor: XCMD:20180402
Forced value:
Default value:
Decimals: 1

Figure E1.1

Virtual table numeric fields

- Create a virtual table in database designer
- Insert the virtual table in the "User defined tables" tab of the configuration window of an object
- Select the numeric field and follow the previous steps.

E.2 Hyperlinks

Fields that are linked to tables through the editor property, display a link to their caption. This link redirects to the corresponding table or object depending on the editor command.

Examples of fields with hyperlink captions are all the selector list and combo box controls, that link through their editors to specific objects and tables (Figure E2.1).

More specifically links can be created by:

- Fields with lookup to memory-tables that create drop-down lists, which redirect to full data process of the memory-tables (e.g. Geographic Zones, Currencies etc.).
- Fields with lookup to tables that create selector list controls, which redirect to the objects defined in the editor property (e.g. Items in document lines, Customer in document, etc.).
- Webpage and e-mail fields that lead to the corresponding application.
- Command fields that respond as buttons.

The use of hyperlinks renders secondary the importance of the job menus because the operator does not have to "remember" where each job is located. In reality, each job constitutes a job menu of its related jobs and all can be accessed in an easy way, regardless of "where you are at the time" and "where the target-job is located in the menu".

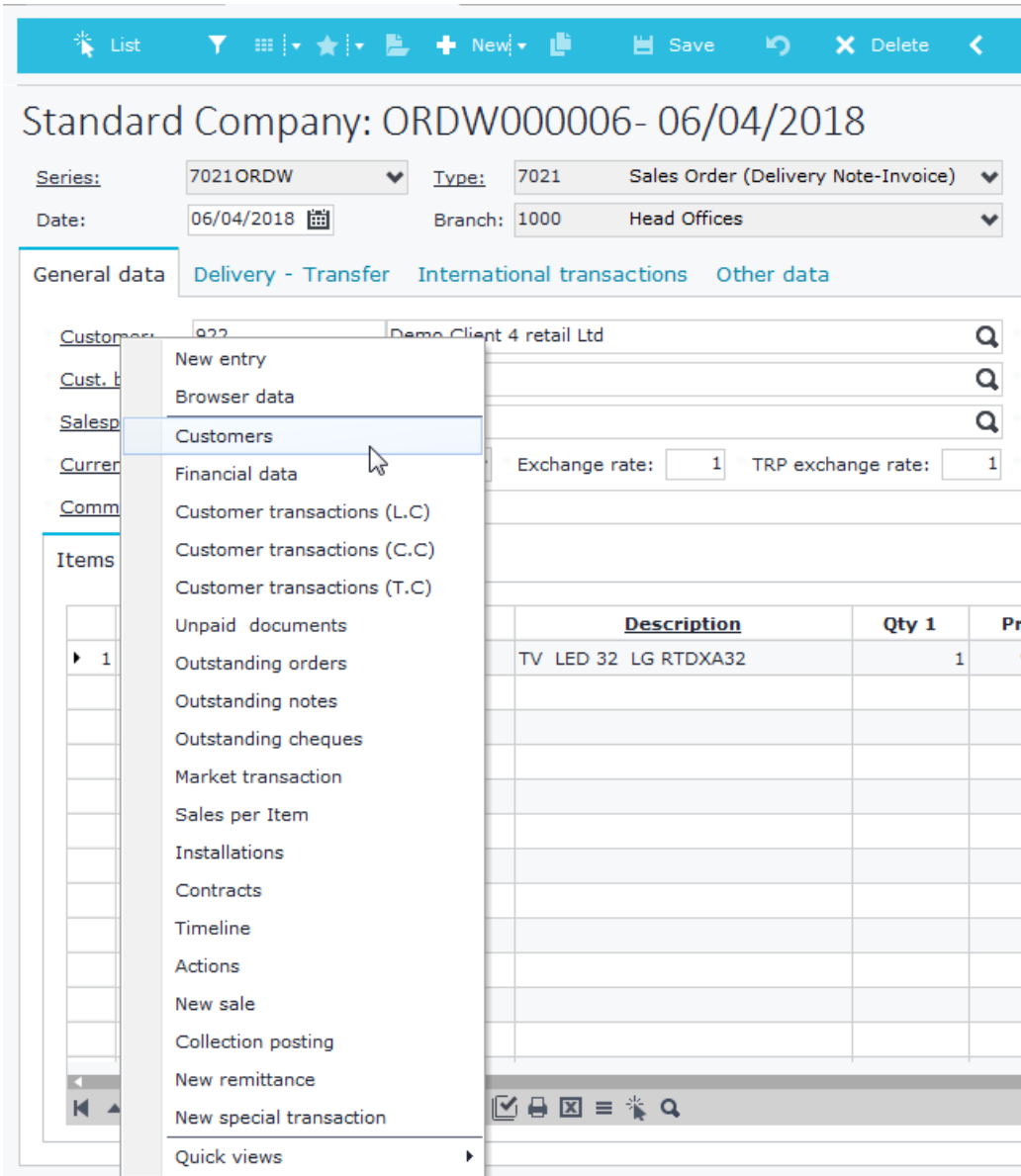


Figure E2.1

F. Attached / Related Files

“Attached Files” is a control for adding attachments (files, shortcuts, URL) to any object, including custom ones. The following Figures (F1 and F2) show that “Attached files” may be included in some objects (e.g. ITEM object), but not included in some others (e.g. Resources - RSRC object).

The screenshot shows the '100-100- Orange juice' form in the 'Stock Items' tab. The form has a top bar with navigation icons and a sidebar with 'Jobs', 'Browsers', 'Screen Forms', and 'Tools'. The main area contains fields for 'Code' (100-100), 'Description' (Orange juice), and 'Active' (Yes). Below these are tabs for 'General data', 'UoM, Alternative Codes', 'Trade', 'Replenishment/Supply', and 'Accounting'. The 'Attached files/Notes' tab is selected, displaying a file browser with a 'root' directory and a file named 'AsciiExpor...'.

Figure F1

The screenshot shows the '100- Sales vehicle Opel Astra DHB-3032' form in the 'Sales Documents' tab. The form has a top bar with navigation icons and a sidebar with 'Jobs', 'Browsers', 'Screen Forms', and 'Tools'. The main area contains fields for 'Code' (100), 'Description' (Sales vehicle Opel Astra DHB-3032), and 'Code 1'/'Code 2'. Below these are tabs for 'Basics', 'Data per Company', 'User-defined fields / Other', and 'Remarks'. The 'Fast Entry' section contains fields for 'Materials module', 'Material', 'Contact module', 'Contact', 'Trader module', 'Trading party', and 'Unit cost/Hours'. The 'Sorting' section contains fields for 'Branch', 'Dept.', 'Business unit', 'Geographical location', 'Project', 'Category', and 'Setup Cost'. The 'Vehicle data Analysis' section shows a table with columns for 'Date', 'Kilometers', 'Ageing ratio', 'Ageing ratio (K...', and 'Value after deduction'.

Date	Kilometers	Ageing ratio	Ageing ratio (K...	Value after deduction
30/10/2017	1,200.00		1.00	13,860.00
05/12/2017	1,450.00		2.00	13,720.00
13/12/2017	1,900.00		3.00	13,580.00

Figure F2

The process of adding Attachments in any object is explained below, using as an example the internal object **Sets / Kits** (Object ID: **70**, Object Name: **SPCGPS**). The same process can be used for any custom object.

- Create a new string list inside a record of Soft1 Designer and name it **CUSTOMDOCINFO** – this keeps the references of all the custom objects that use “Attachments” (the similar SoftOne internal string list is SOFTDOCINFO). Use the script of the SBSL example “[Get String List Data](#)” to retrieve all the data of either SOFTDOCINFO or CUSTOMDOCINFO (Figure F3).

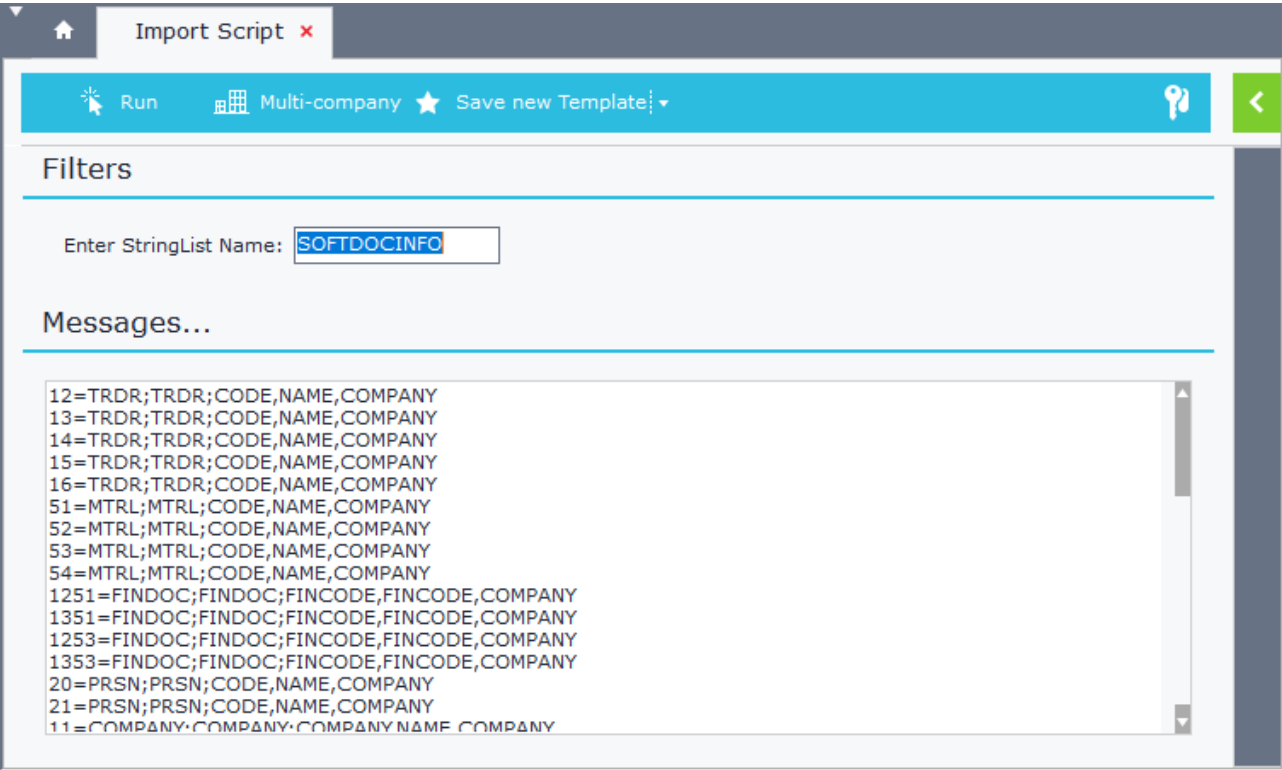


Figure F3

- Enter a new record as in Figure F4.
 - **Key:** Object id (in this case its 70).
 - **Value:** Main TableName;Primary Key;Varchar Field1(CODE);Varchar Field2(NAME);COMPANY field. The last three fields are used in Documents File code, description fields.
- Save the record and close Soft1 Designer. There is no need for DB sync, since the DB schema has not changed.

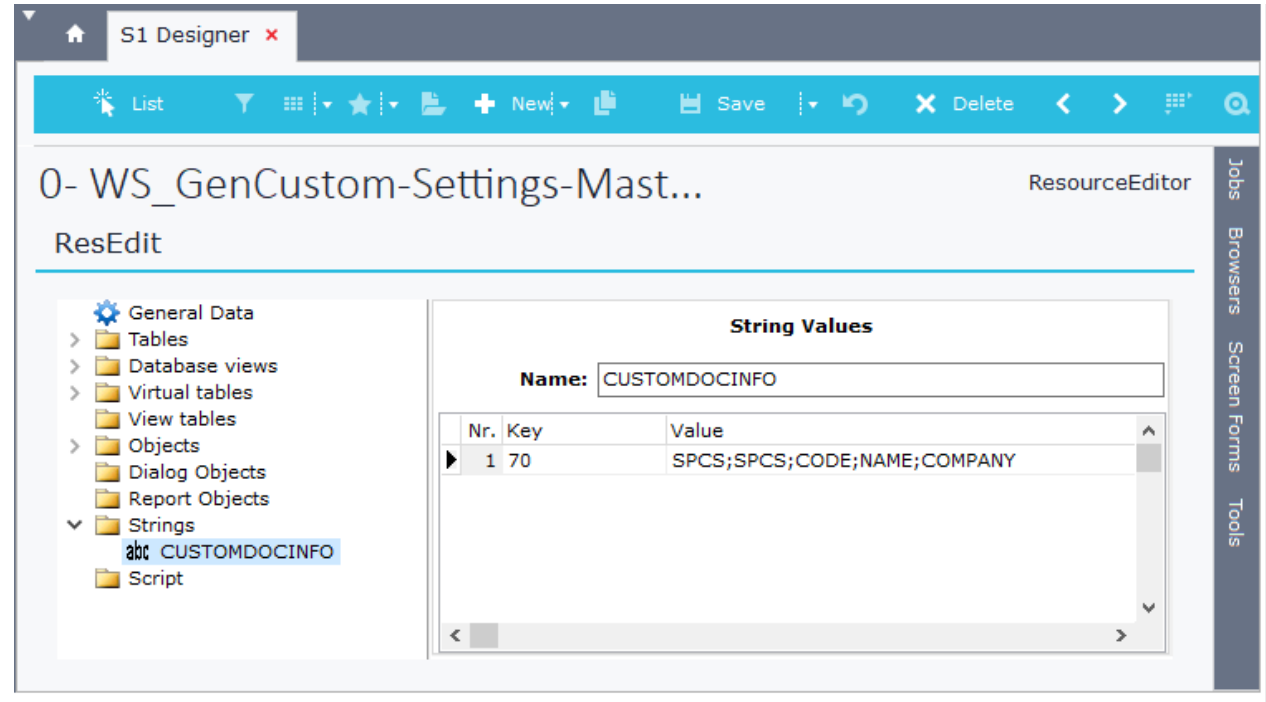



Figure F4

- Open the object form (Sets / Kits) in design mode and add the table XTRDOCDATA (Document Pointers) as in Figure F5.
 1. Click on "Configuration" button and move to "User Defined Tables" tab of "Entity Configuration" window.
 2. Click on "Add new table" 
 3. Select "Document pointers" from the "Available Tables" window.
 4. Select XTRDOCDATA from the drop-down list "Tables"
 5. Check that "Document pointers" is added.

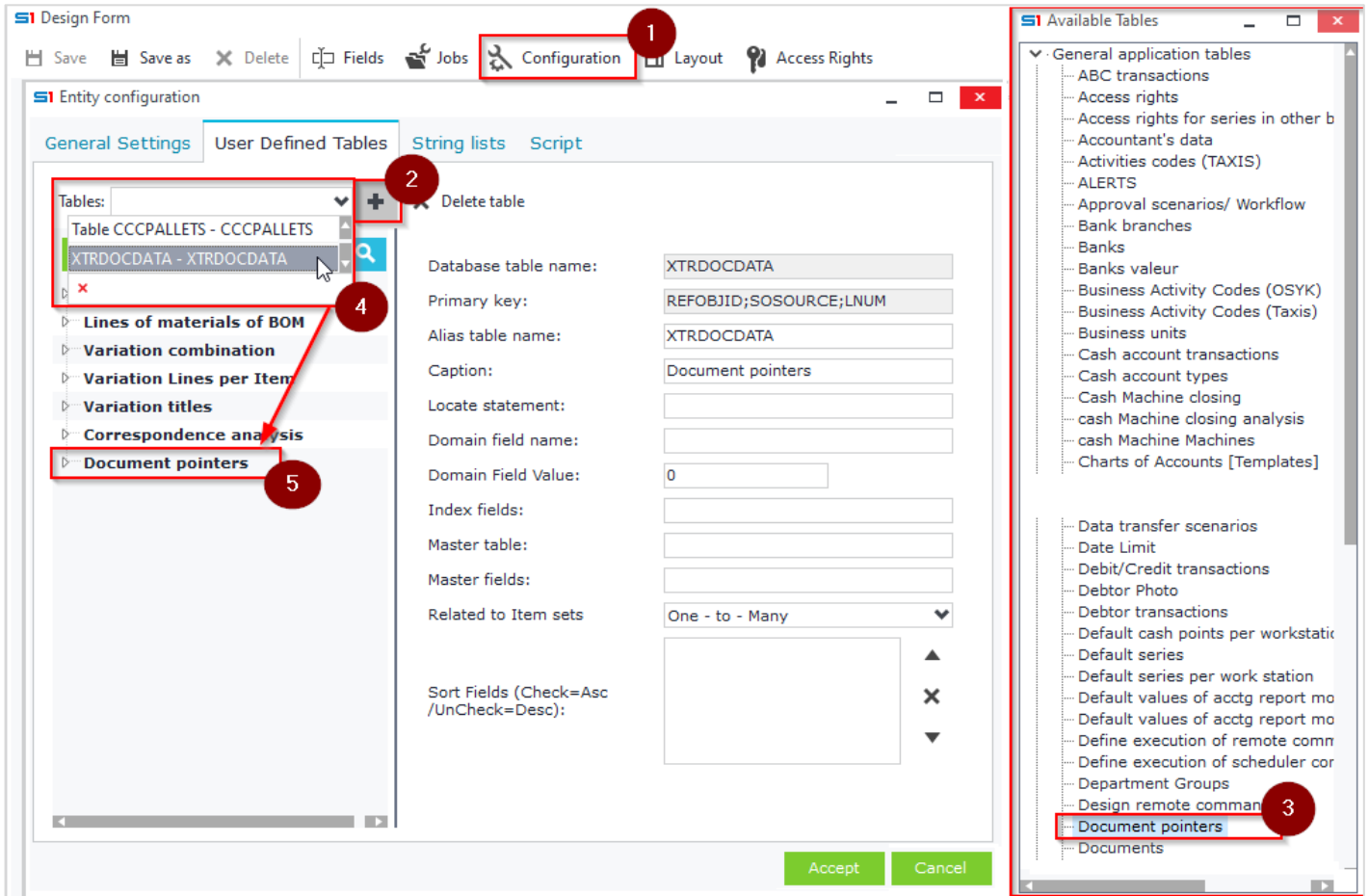


Figure F5

- Alter the "Alias table name" to any unique name (e.g. CCCMYDOCDATA).
- Enter the following "Locate statement": **REFOBJID;SOSOURCE;XDOCTYPE=:SPCS;70;1** (Figure F6)
REFOBJID;SOSOURCE;XDOCTYPE=:PrimaryKey;ObjectID;1

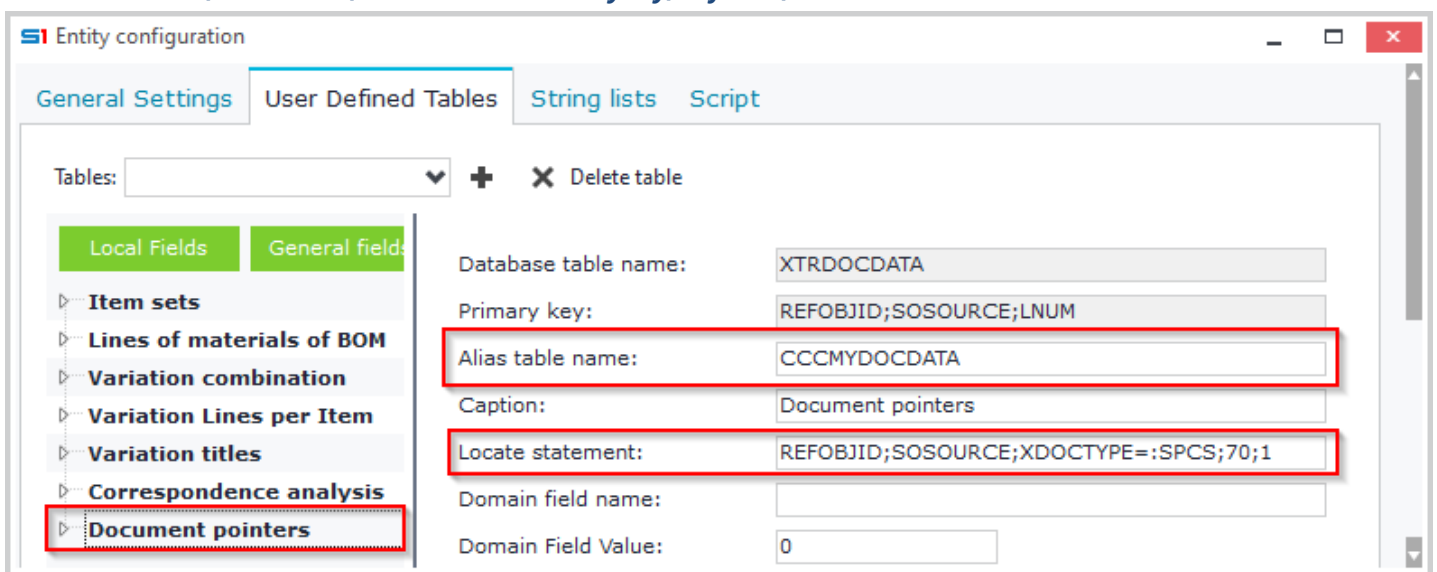
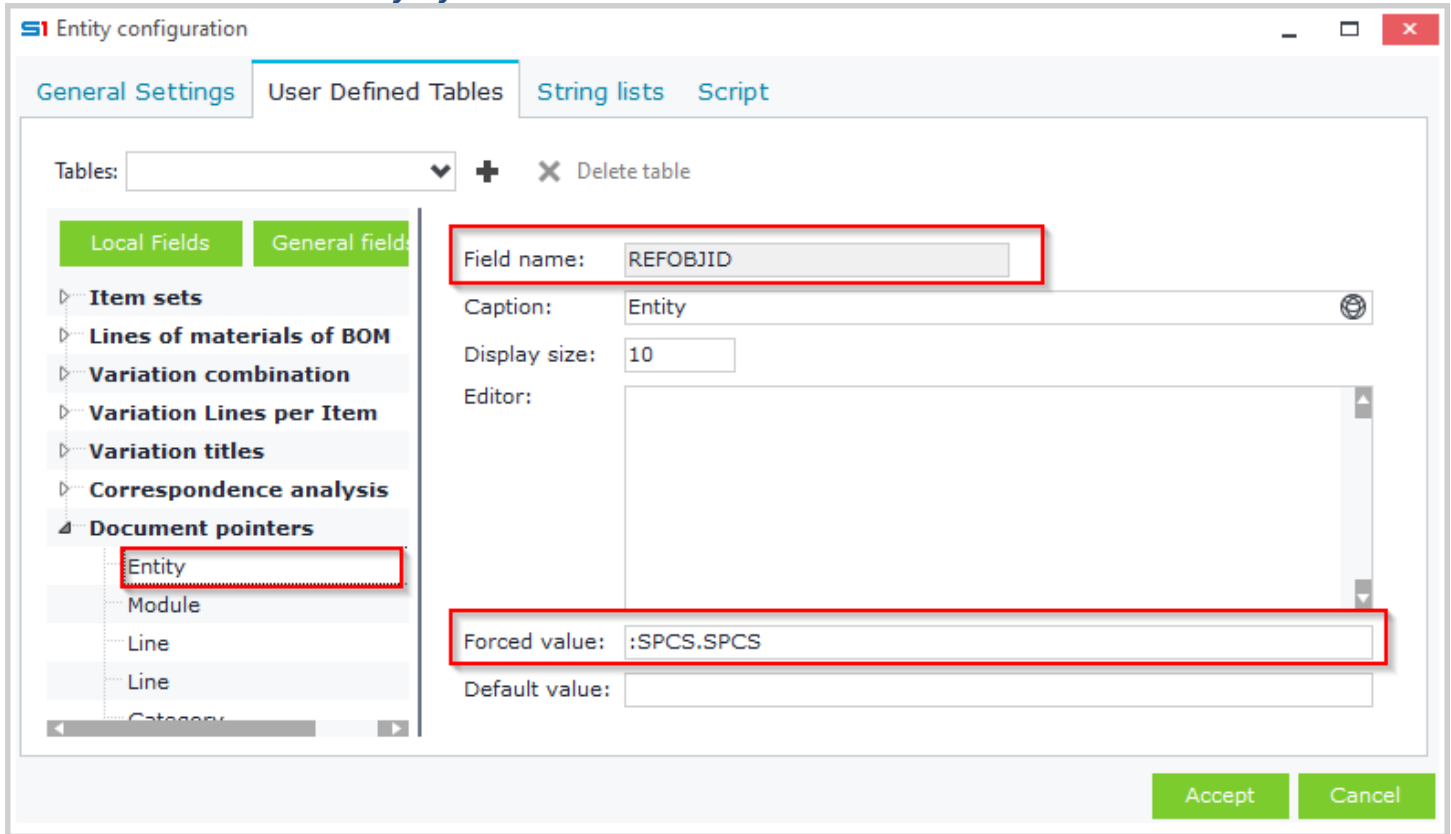


Figure F6

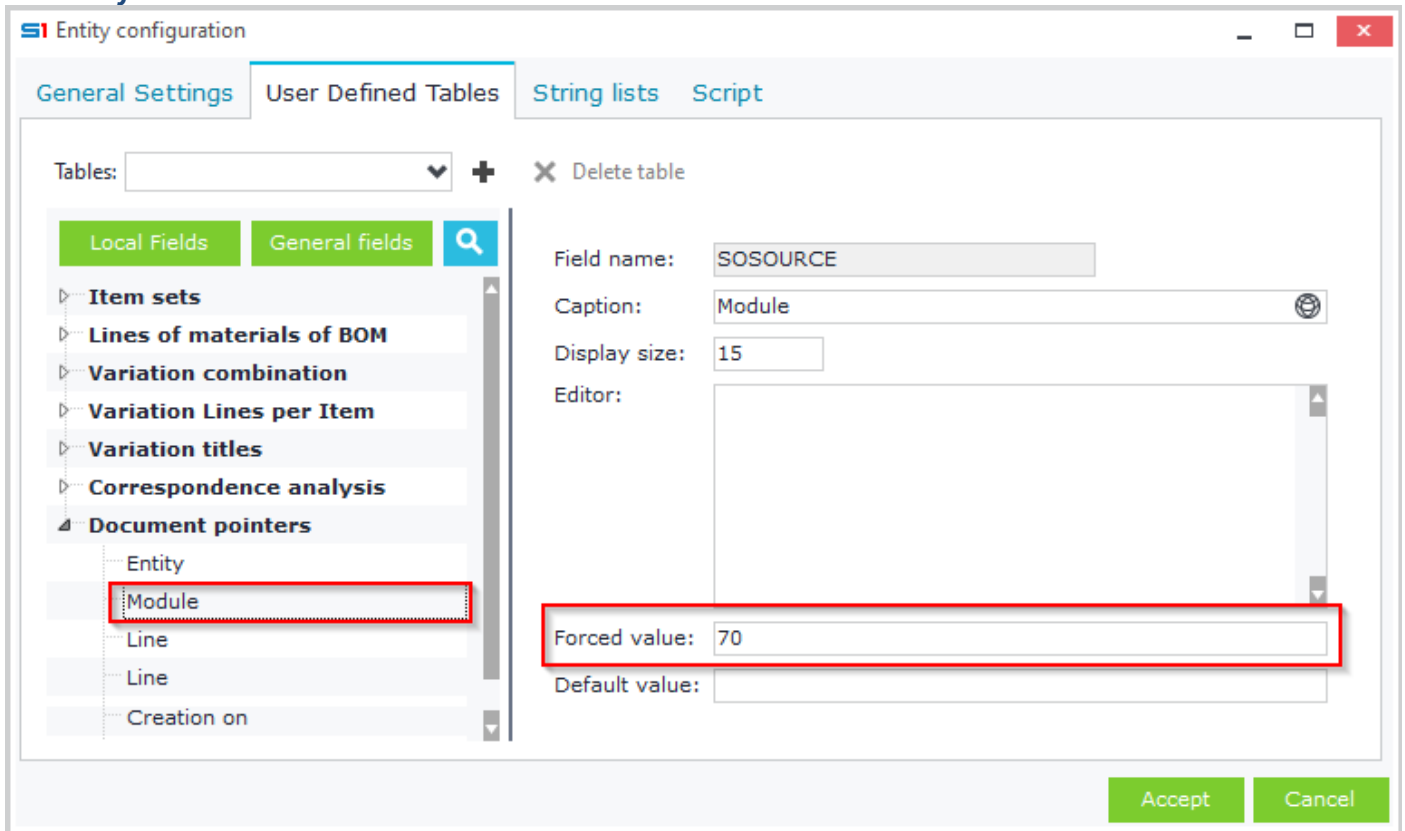
- Select the field Entity (REFOBJID) of Document pointers table and enter the forced value: **SPCS.SPCS** (Figure F7)
:MainTableName.PrimaryKey



The screenshot shows the 'Entity configuration' window with the 'User Defined Tables' tab selected. In the left sidebar, under 'Document pointers', the 'Entity' field is highlighted. The main area shows the configuration for 'Field name: REFOBJID'. The 'Caption' is 'Entity', 'Display size' is '10', and 'Editor' is empty. The 'Forced value' is set to ':SPCS.SPCS'. The 'Default value' is empty. At the bottom right, there are 'Accept' and 'Cancel' buttons.

Figure F7

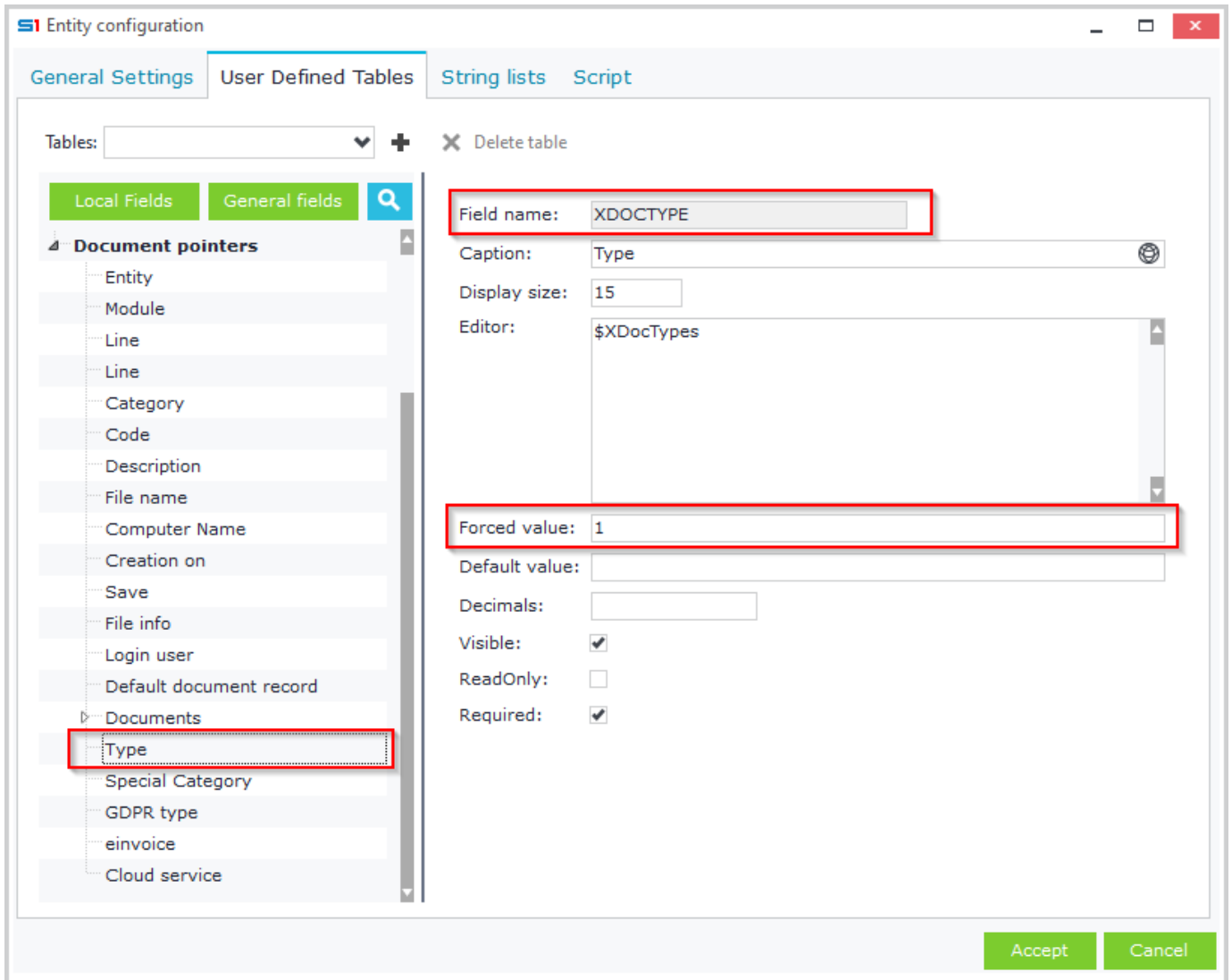
- Select the field Module (SOSOURCE) of Document pointers table and enter the forced value **70** (Figure F8)
ObjectID



The screenshot shows the 'Entity configuration' window with the 'User Defined Tables' tab selected. In the left sidebar, under 'Document pointers', the 'Module' field is highlighted. The main area shows the configuration for 'Field name: SOSOURCE'. The 'Caption' is 'Module', 'Display size' is '15', and 'Editor' is empty. The 'Forced value' is set to '70'. The 'Default value' is empty. At the bottom right, there are 'Accept' and 'Cancel' buttons.

Figure F8

- Select the field Type (XDOCTYPE) of Document pointers table and enter the forced value **1** (Figure F9)

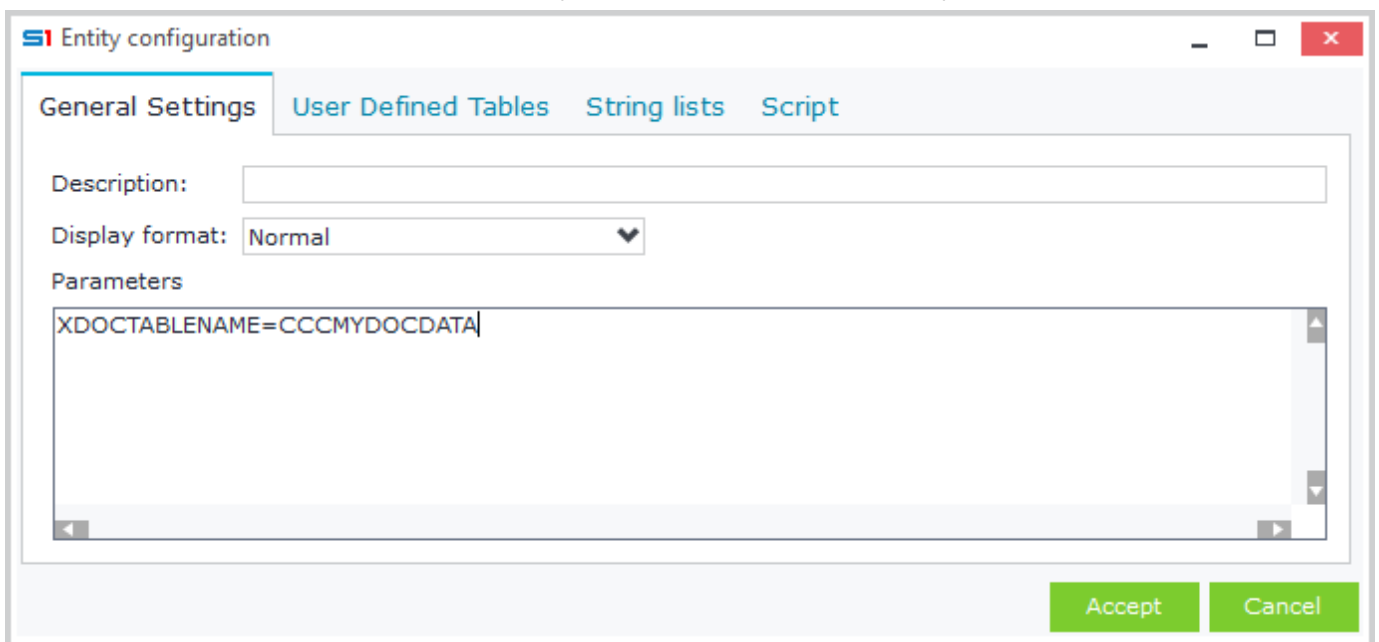


The screenshot shows the 'Entity configuration' window with the 'User Defined Tables' tab selected. In the left sidebar, the 'Document pointers' table is expanded, and the 'Type' field is highlighted. The main configuration area shows the following details for the 'Type' field:

- Field name:** XDOCTYPE
- Caption:** Type
- Display size:** 15
- Editor:** \$XDocTypes
- Forced value:** 1
- Default value:** (empty)
- Decimals:** (empty)
- Visible:** ☒
- ReadOnly:** ☐
- Required:** ☒

Figure F9

- In General Setting tab of the form add the command (Figure F10):
XDOCTABLENAME=CCCMYDOCDATA (XTRDOCDATA Alias Table Name)



The screenshot shows the 'Entity configuration' window with the 'General Settings' tab selected. The configuration details are as follows:

- Description:** (empty)
- Display format:** Normal
- Parameters:** XDOCTABLENAME=CCCMYDOCDATA

Figure F10

Chapter 1 – Screen Forms

- Finally, click on “Layout” and add a new tab. Add a special area and enter the editor: **DocFrm** (Figure F11).

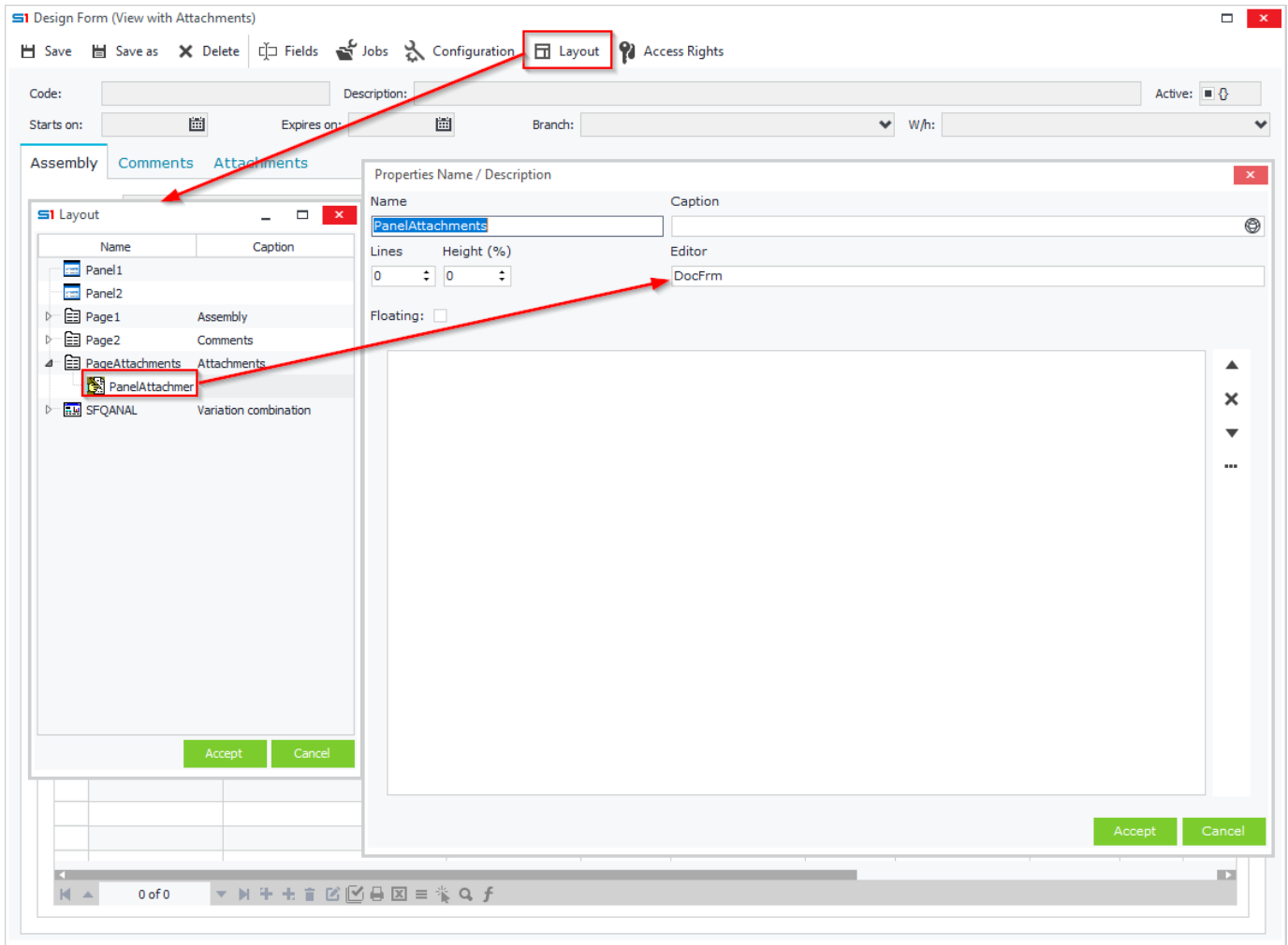


Figure F11

- Save the form, insert or locate a record and check the functionality of the new control (Figure F12).

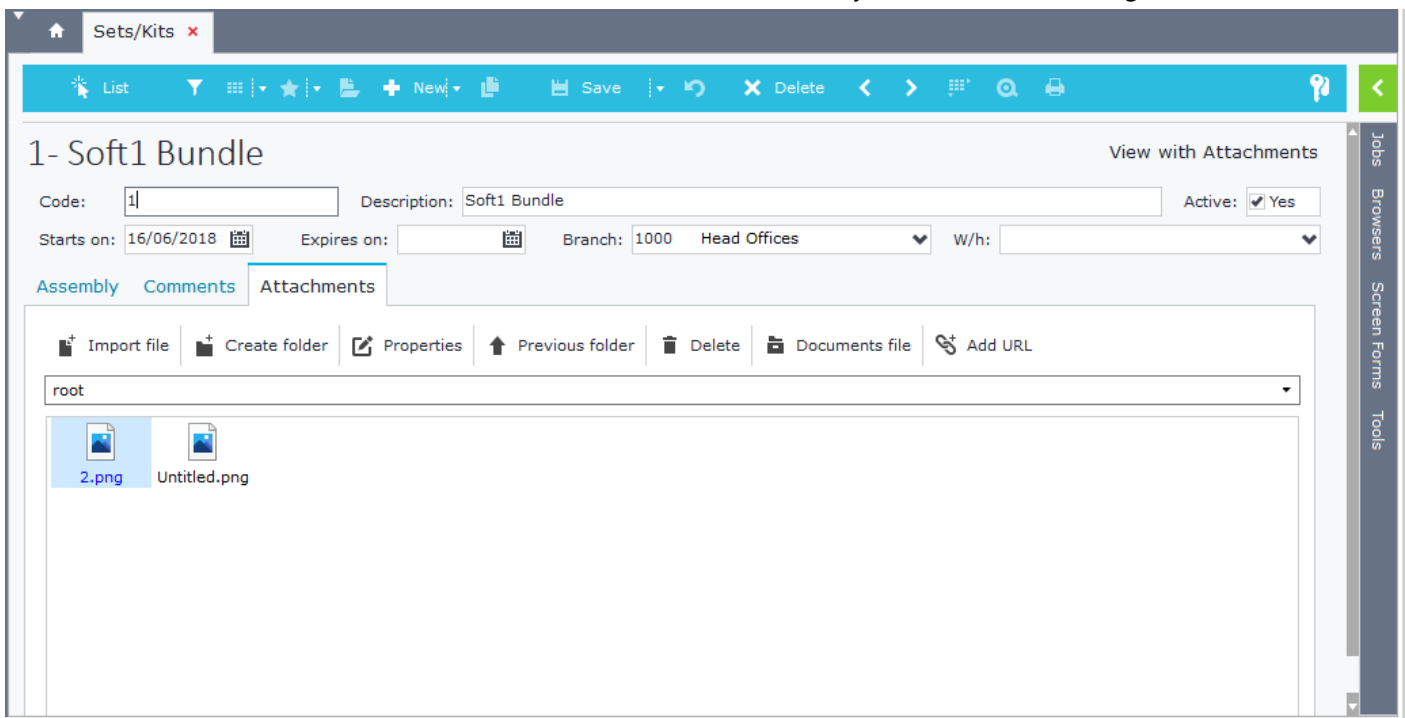


Figure F12

G. Editor Commands

The table below summarizes all the available editor commands and their use.

EDITOR COMMANDS		
Command	Operation – Usage	Data Type
\$DT	Date & Time in different fields	Datetime
\$TIME	Only Time (Returns the current Date)	Datetime
\$PASSWORD	Password Mask (Displays text with asterisks)	Varchar
\$PASSWORDH	Hash Password (Displays text with asterisks)	Varchar
\$Y	Boolean field (Displays checkbox control)	Small Integer
\$YN	Boolean field in datagrids (Displays “Yes/No/Null” drop-down list)	Small Integer
\$WEBPAGE	Web page fields - Link to internet browser	Varchar
\$EMAIL	Email fields - Link to e-mail application	Varchar
\$PRINTERS	Displays printers window dialog	Varchar
\$IMAGE	Displays image control (usage in Image fields)	Image
\$DATERANGE	Date from to	Datetime
\$TIMERANGE	Time from to	Datetime
\$COLOR	Displays SoftOne color selection window	Varchar
\$ONECOLOR	Displays Windows color selection window: $(65536 * R) + (256 * G) + B$ = $(256 \wedge 2 * \text{Red}) + (256 \wedge 1 * \text{Green}) + (256 \wedge 0 * \text{Blue})$	Varchar
\$FILENAME	Displays file selection window and returns its path	Varchar
\$FOLDERNAME	Displays folder selection window and returns its path	Varchar
\$MASKEDIT	Code mask creation window	Varchar
\$RECURRENCE	Displays SoftOne scheduler window	Varchar
\$MEMOEDIT	Word wrap in datagrid lines for memo textbox fields	Varchar
\$LIST:OBJECT	Lookup from preselection lists created from browsers. Tables used are SOLIST and SOLISTLNS.	Varchar
\$XRUN	Runs external program. \$XRUN(D[0,C:\Xplorer.exe,"/" + CUSEXTRA.VARCHAR01]) The above example runs SoftOne application using the switch parameters added in field CUSEXTRA.VARCHAR01.	Varchar

H. Editor Attributes

Inside Editors, that retrieve data from other Tables, Views or Strings (e.g. ITEM), it is possible to enter extra commands that have various functions, such as entries filtering, fields updating, change links etc.

The following table summarizes all the available commands as well as examples of their syntax.

Note: Line breaks and Quotes are not permitted in editors, instead you can use the sql function char().

EDITOR ATTRIBUTES		
Command	Operation – Use	Syntax Example
W (Master Table Selectors)	Filters table data using extra where clause in SQL statement. Usage in selector fields linked to master tables.	SALDOC(W[A.COMPANY=:X.SYS.COMPANY]) Lookup data from sales documents table using as filter the Company Login. Login system parameters (as Login Company) can be used by entering a colon : and then the name of the param(:X.SYS.COMPANY). CUSTOMER(W[A.CODE LIKE CHAR(97)+CHAR(37)]) Customers that code starts with letter 'a' (... AND A.CODE LIKE 'a%')
W (Memory Table Selectors)	Filters memory table data using the operators AND, OR, <>, >, <. Usage in selector fields linked to memory tables.	SERIES(F[COMPANY;SOSOURCE=:X.SYS.COMPANY;1351],W[FPRMS=7000 OR FPRMS=7001]) Display sales series from login company that FPRMS=7000 or FPRMS=7001
W (String Lists)	Filters String List data. Usage in fields linked to string lists.	\$CCCMYSTR (W[1,2,3]) Lookup data from the string list CCCMYSTR displaying only the values that the key is 1 or 2 or 3.
F	Filters memory table data. Usage in fields linked to memory tables.	MTRMARK(F[CCCMYFIELD;SODTYPE=1;51]) Lookup from the table MTRMARK filtering data using CCCMYFIELD = 1 and SODTYPE = 51 fields and values.
M, R	Defines the field that will be returned. Usually used in string fields.	CUSTOMER(W[A.CCCMYFIELD=1],M,R[CODE]) Lookup data from customers (inherited table) using specific filter from CCC field and return the customer's code. (Use the Editor of the above example in a varchar field)
U	Updates fields after selection. Usage in selector fields linked to tables.	SERIES(U[FPRMS;BUSUNITS;=FPRMS;BUSUNITS@]) Lookup data from series table and update the FPRMS and BUSUNITS fields with the values of the fields FPRMS and BUSUNITS of the selected series. The @ symbol at the end indicates that the fields on the left, will always be updated, no matter if they already have a value or not. If you do not enter the @ symbol, then the fields will be updated only if they do not already have a value.
H	Redirection to Object, using the hyperlink of the field label.	CCCTABLE(H[CCCOBJECT]) or CCCTABLE(H[#ObjectID;RecordID]) Lookup data from table CCCTABLE. Redirection to Object CCCOBJECT.
E,W[]	Exclusion Filter in String Lists. Usage in fields linked to string lists.	\$CCCMYSTR (E,W[5]) Lookup data from the string list CCCMYSTR displaying only the values that the key is not 5.
J	Link to specific field. Usage in Custom Design Printouts.	SALDOC(J[A.FINDOC])
I	Defines the field that will be updated when user clicks the button "Input" in fields of datagrids with editors.	ITEM(I[QTY1]) In sales documents, when user selects from the available items and clicks on the button "Input" then the data entered will update the QTY1 field.
L	Local Editor that lookups data from a table, db view or virtual table that is inside an object.	1. CCCMYVTTABLE(L,F[CCCMYVTTABLEFIELD=:OTHERTABLE.FIELD]) 2. #WHOUSE(L) (example in object Company → BRANCH.WHOUSES)

C	<p>Removes the sql expression "AND A.COMPANY= <LoginCompany>" from the where clause.</p> <p>Usage in selector fields linked to master tables.</p>	<p>SALDOC(W[A.CCCField in (1,2,3)],C)</p> <p>Lookup data from sales documents of all the companies, using also an extra where clause: "AND A.CCCField in (1,2,3)".</p>
A	<p>Removes the sql expression "AND A.ISACTIVE=1" from the where clause.</p> <p>Usage in selector fields linked to master tables.</p>	<p>CUSTOMER(A)</p> <p>Lookup data from Customer inherited table (Active and Inactive records)</p>
Z	<p>Applies to Soft1 Designer String Lists only. Displays both Key and Value of the string list.</p>	<p>\$SODTYPE(Z)</p> <p>Displays data in format "Key – Value"</p>

2. BROWSERS

- A. [Modules](#)
- B. [Columns](#)
- C. [Filters](#)
- D. [Grouping](#)
- E. [Sorting](#)
- F. [Parent / Child Browsers](#)
- G. [User-defined Tables](#)

Overview

Browsers are used for displaying data from database tables and views. They serve a lot of features which can be found either on runtime or in design mode. Browsers are displayed when users open any object from the menu. Browsers are saved as blob data inside the table CSTINFO (CSTTYPE=0) and can be easily transferred through different installations as .cst or .auv files using the ["Custom Administration"](#) tool.

The first screen displayed when you double click an object is the "Filters screen", where users can define filters and then view the records by clicking on the button **List**. The screen that appears next is the "List screen" (runtime mode), which displays records in one or more datagrids depending on the design of the selected browser.

In runtime mode, users can filter, group, sort, copy and paste data and use any other option that is available in the right click pop up menu. Browsers also allow users to alter the data of the records as long as the option "Editable" is enabled through the design mode of the browser.

Another great feature of browsers is the ability to print the records displayed in any available system printer or in MS-Word, Document file, MS-Excel, Spreadsheet file, ASCII file, PDF file, HTML file and even attach the file and send it through e-mail.

Horizontal or vertical grouping of data is also available through the button "Grouping" of the toolbar.

Pivot analysis is yet another great tool embedded in browsers that displays data containing summarized numeric columns in pivot table. Pivot tool provides all the functions needed to perform full analysis of your data.

The following image (Figure A1) displays the design mode of the "Sales documents" browser. The available fields are displayed in the left-hand side of the window, while the right part shows the selected fields (Columns) that will be displayed in runtime of this particular browser.

Designing a browser is very easy. Just select a browser from the menu on the right pane and click the "Design" icon. The available design options are summarized in the following sections.

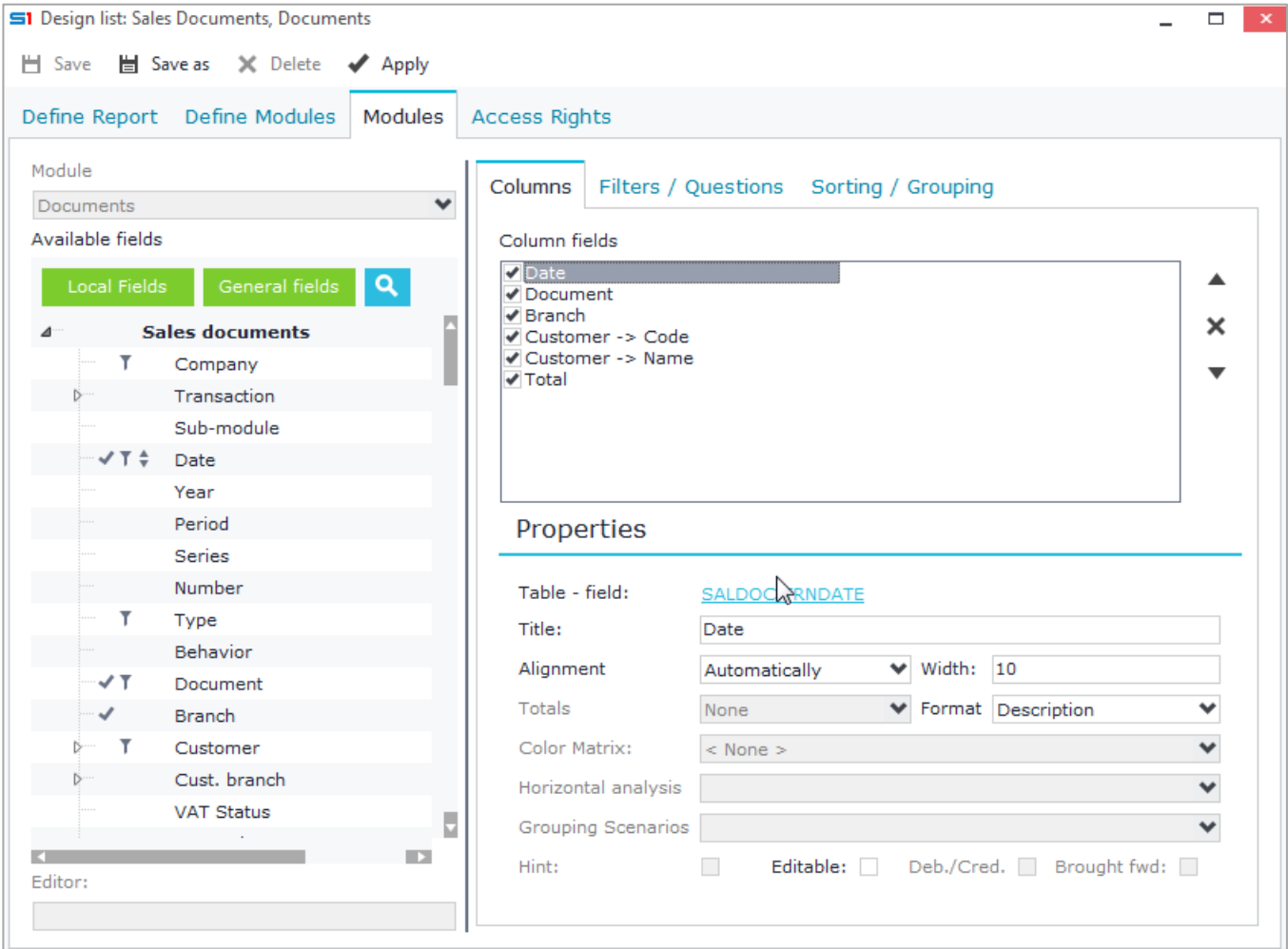


Figure A1 – Sales Documents Browser design

The toolbar buttons in design mode of a browser are the following (Figure A2).

Figure A2–Browser toolbar buttons

Save

Saves the changes of the browser using its current name.

SoftOne default browsers cannot be overwritten. This means that when you use “Save” button in those browsers the “Save as” window will appear.

Save as

Displays the “Save as” window in order to save the browser with a different name.

Delete

Deletes the selected browser from the database.

Apply

Applies the changes of the browser and makes it possible to run it and see if it fits your needs.

Notice that this button **does not save the browser** in the database. You need to use the “Save” or “Save as” buttons in order to permanently save a browser in the database for later use.

A. Modules

The properties of the tab “Define Modules” (Figure A3) affect all the columns of the browser and are explained in the table below.

Design list: Sales Documents, Sales List

Save Save as Delete Apply

Define Report Define Modules Modules Access Rights

+ Add X Delete

Document List

Table: Sales documents

Expand by Sales documents

Properties

Title: Document List

Space from left: 0

Fixed columns: 0

Totals file:

Ascending numbering: ☐ 0

Page Break: ☐

Display zeros: ☐

Show Title: ☐

Show fields Titles: ☒

Dividing line at the end: ☐

Dividing line per entry: ☐

Auto preselection: ☐

Auto fit columns size: ☐

Section Width:

Max Width: 150

Figure A3

PROPERTY	DESCRIPTION
Expand by	Defines the table of the module that will be used to display the data.
Fixed columns	Number of columns that are kept visible while scrolling the browser results.
Totals file	Enter a name of a file that will be created in SoftOne Temp path, containing the total values of the numeric fields of the browser. The path where the file is saved is: <Drive letter>:\Documents and Settings\ <Windows user >\Application Data\SoftOne\Temp
Ascending Numbering (Line Digits)	Number of digits that will be displayed in the header column of browser results. For example, if you enter the number 2, then the header column width will be adjusted in order to be able to display up until entry 99. This does not mean that the browser will be limited to this number of rows, but simply that the header column of records below 99 will not be displayed, because the header column width does not auto fit. The default value is 2.
Display zeros	It specifies whether the numeric columns of zero values will be blank or displayed with zero.
Auto preselection	Auto selects all the records displayed in the browser.
Auto fit columns size	Automatically adjusts the size of the columns of the browser based on the column contents.
Quick View	Enables the right click option “Quick view” in fields that are linked to this table through their editor property. For example, in item lines of Sales documents object the right click option “Quick view” is enabled if you have created a “Quick View” browser in Items object.
Force Pivot	Execution of the browser opens the Pivot Analysis tool automatically.
Fast entry	It allows inserting new entries directly from the browser. All the record required fields must be filled.

B. Columns

The “Modules” tab (Figure B1) is split in two panels. The left panel contains the fields that are available to display as columns and filters or for grouping and sorting. Fields are inserted on the right hand-side of the window either using double click or using drag-and-drop. “**Columns**” tab consists of the fields that will be displayed as columns of the browser in runtime. Change the position of the columns using drag and drop or the arrows on the right side.

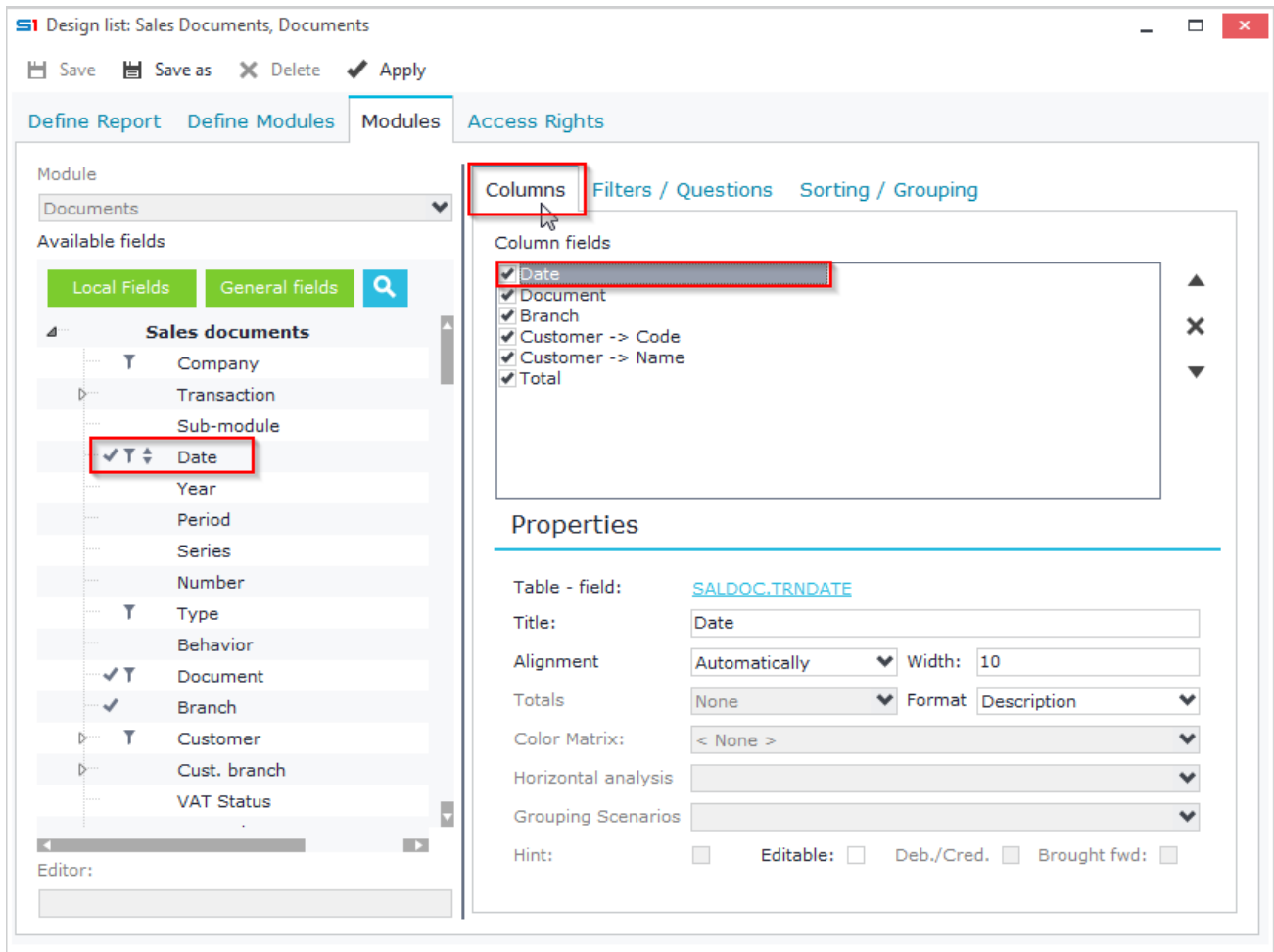


Figure B1

Unchecked columns are not visible in runtime, but they are executed in the SQL select statement and they can be used to compute other columns. Figure B2 shows a browser in runtime mode.

Documents					
Date	Document	Branch	Code	Name	
23/12/2018	HSSN344343	Head Offices	209	Maniopoulos Argyris	
23/12/2018	HSSN344343	Head Offices	209	Maniopoulos Argyris	
23/12/2018	HSSN344343	Head Offices	209	Maniopoulos Argyris	
23/12/2018	HSSN344343	Head Offices	209	Maniopoulos Argyris	
23/12/2018	HSSN344343	Head Offices	209	Maniopoulos Argyris	
23/12/2018	HSSN344343	Head Offices	209	Maniopoulos Argyris	
23/12/2018	HSSN344343	Head Offices	209	Maniopoulos Argyris	

Figure B2

Field Properties

Once finished adding columns, proceed with the configuration of each column at the bottom part of “Columns” as shown in Figure B3.

Design list: Sales Documents, Horizontal Analysis

Save Save as Delete Apply

Define Report Define Modules Modules Access Rights

Module: Documents

Available fields:

Local Fields General fields

Sales documents

- Company
- Transaction
- Sub-module
- ☒ Date
- ☒ Year
- ☒ Period
- Series
- Number
- Type
- ☒ Behavior
- ☒ Document
- Branch
- Customer
- Cust. branch
- VAT Status

Editor:

Columns Filters / Questions Sorting / Grouping

Column fields:

- ☒ Date
- ☒ Period
- ☒ Document
- ☒ Behavior
- ☒ Customer -> Name
- ☒ Net

Properties

Table - field: SALDOC.NETAMNT

Title: Net

Alignment: Automatically Width: 17

Totals: Sum Decimals: 2

Color Matrix: values (normal)

Horizontal analysis: Date

Grouping Scenarios: Monthly analysis

Hint: ☐ Editable: ☐ Deb./Cred. ☐ Brought fwd: ☐

Figure B3

The available field options are:

PROPERTY	DESCRIPTION
Table – field	Name of table and field (Read only).
Title	Title of the field.
Alignment	Cell alignment of the field.
Width	Column width.
Totals	Applies only to numeric fields. Displays the totals of the field at the bottom line of the browser. Available functions are the following: Min, Max, Average and Sum.
Decimals	Number of decimal places.
Color matrix	Color zone that will be applied to a field, depending on the range of values.
Horizontal analysis	Horizontal Analysis of a numeric field based on another field of the browser (e.g. Period)
Grouping Scenarios	Extra grouping of the horizontal analysis. For example, if you have selected a date field in horizontal analysis then you can perform grouping by month.
Hint	Displays the title of the field on mouse over the column.
Editable	Sets the column in edit mode and allows user to alter field data through browser.

Figure B4 displays a sales browser grouped (horizontally) by customer name, using also **horizontal analysis** of the total value in months.

Date	Period	Document	Net Jan 2018	Net Feb 2018	Net Mar 2018	Net Apr 2018
Name : Kostas Sotiropoulos [1]			0,00	0,00	54.596,00	0,00
Name : Maniopoulos Argyris [8]			0,00	0,00	0,00	0,00
Name : Michael Jackson [35]			147.933,20	18.506,19	0,00	1.346.195,52
Name : Nicholas Angel [2]			106.716,00	0,00	0,00	0,00
13/01/2018	January	SISN000069	46.465,00			
21/01/2018	January	SISN000070	60.251,00			
Name : Nick Panterben [7]			1.890,00	0,00	0,00	15.000,00
Name : Nikolakakis Phaidon [1]			340,24	0,00	0,00	0,00
Name : Nomikos Evangelos [1]			0,00	0,00	15.095,00	0,00

Figure B4

C. Filters

This tab contains the fields that will be displayed in the first dialog (filters) of the browser and will be used to filter the data of the browser. Add the fields using drag and drop or double click (Figure C1).

PROPERTY	DESCRIPTION
Table – field	Name of table and field (Read only).
Title (From – To)	Title of the field as it will be displayed in filter dialog.
Value (From – To)	Default value of the field. You can also use any of the available system parameters.
Locked	Set the mode of the field to Read only.
Required	Defines that users have to fill this field in order to list the data of the browser.

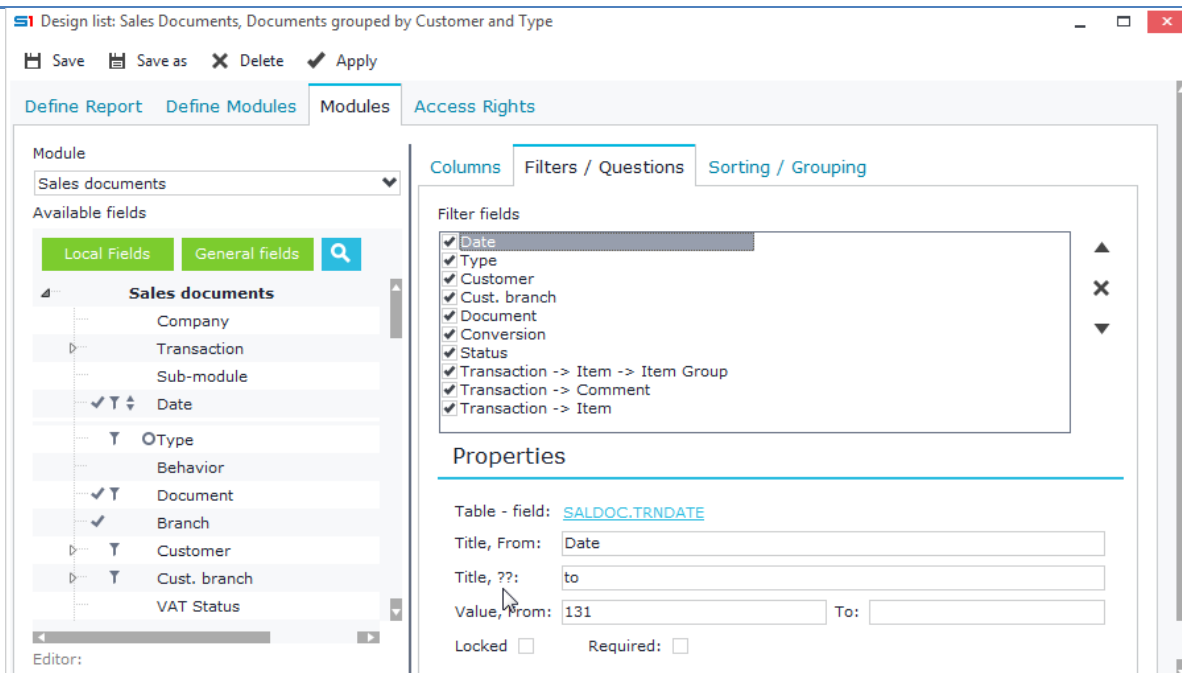


Figure C1

D. Grouping

This tab contains the fields that will be used for grouping the data of the browser in runtime. Add the fields using drag and drop or double click and select grouping layout (vertical or horizontal).

Additionally, you can set the following parameters (Figure D1):

PROPERTY	DESCRIPTION
Activate	Activates grouping for the selected field.
Layout	Defines the grouping layout, horizontal or vertical.
Grouping intervals	Defines the grouping intervals depending on the type of field.
Auto Run	Auto group after running the browser.
Detailed View	Selecting the cost accounting view displays all the selected columns of operating results (see above), otherwise only the grouping and value fields columns shown as totals, will be displayed.
Auto Filter	Filters the group entries and displays only the ones included in the results of the browser.
Paging	Exports the group index in different worksheets of Excel file.

Note: While running the browser with grouping, then the grouping columns are also used as sorting columns.

SI Design list: Sales Documents, Documents grouped by Customer and Type

Save Save as Delete Apply

Define Report Define Modules Modules Access Rights

Module: Sales documents

Available fields: Local Fields General fields

Sales documents

- Company
- Transaction
- Sub-module
- ✓ T Date
- Year
- Period
- Series
- Number
- Y O Type
- Behavior
- ✓ T Document
- ✓ Branch
- Y Customer
- Y Cust. branch
- VAT Status

Editor:

Sorting / Grouping Fields

- ✓ ▲ Date
- ✓ ▲ Customer -> Name
- ✓ ▲ Type

Properties

Table - field: SALDOC.X TNAME

Sorting: Ascending

Grouping

Activate: ☒

Layout: Vertical

Grouping intervals: Auto

Position: 0 Digits: 0

Paging: ☐ AutoFilter: ☒

Detailed View: ☒ AutoRun: ☒

Figure D1

E. Sorting

Inside this tab you define the fields that will be used to sort the results of a browser. Ascending or descending sort can be performed by selecting the appropriate option (Figure E1).

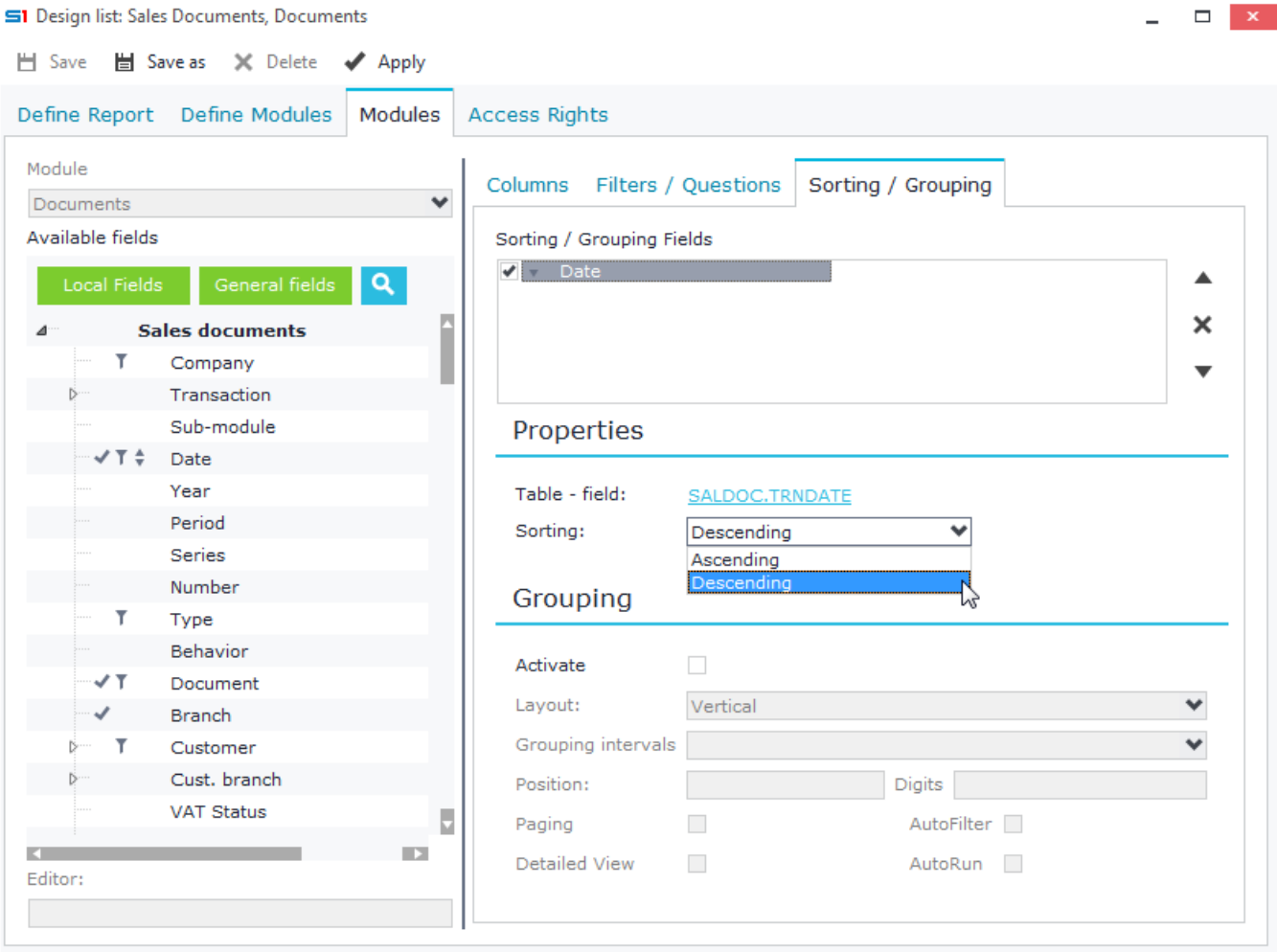



Figure E1

The flag "**Master-Detail tree view**" of "**System Settings**" allows you to display secondary bands in two ways:

- [illegible]

[illegible]

94 | Page

Design of the above is achieved through the "**Define Modules**" tab, where you set the tables that will be used inside the browser. Click on button "**Add**" to insert a <New module> and then select the detail table (e.g. Item Lines) after clicking on the button  (Figure F3).

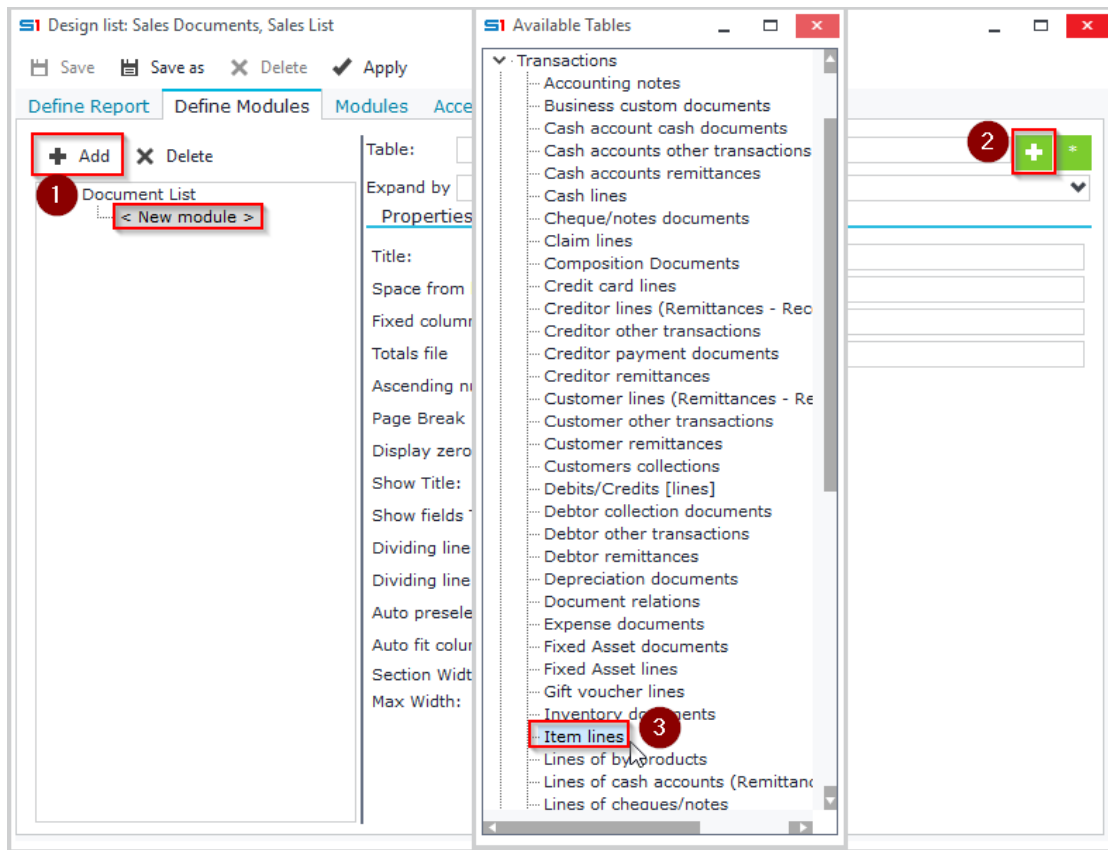


Figure F2

Now, from "**Modules**" tab you can select the new module (e.g. "**Item lines**") and add any columns to your browser (Figure F3). Repeating the above steps you can add more than one bands (e.g. "**Service lines**").

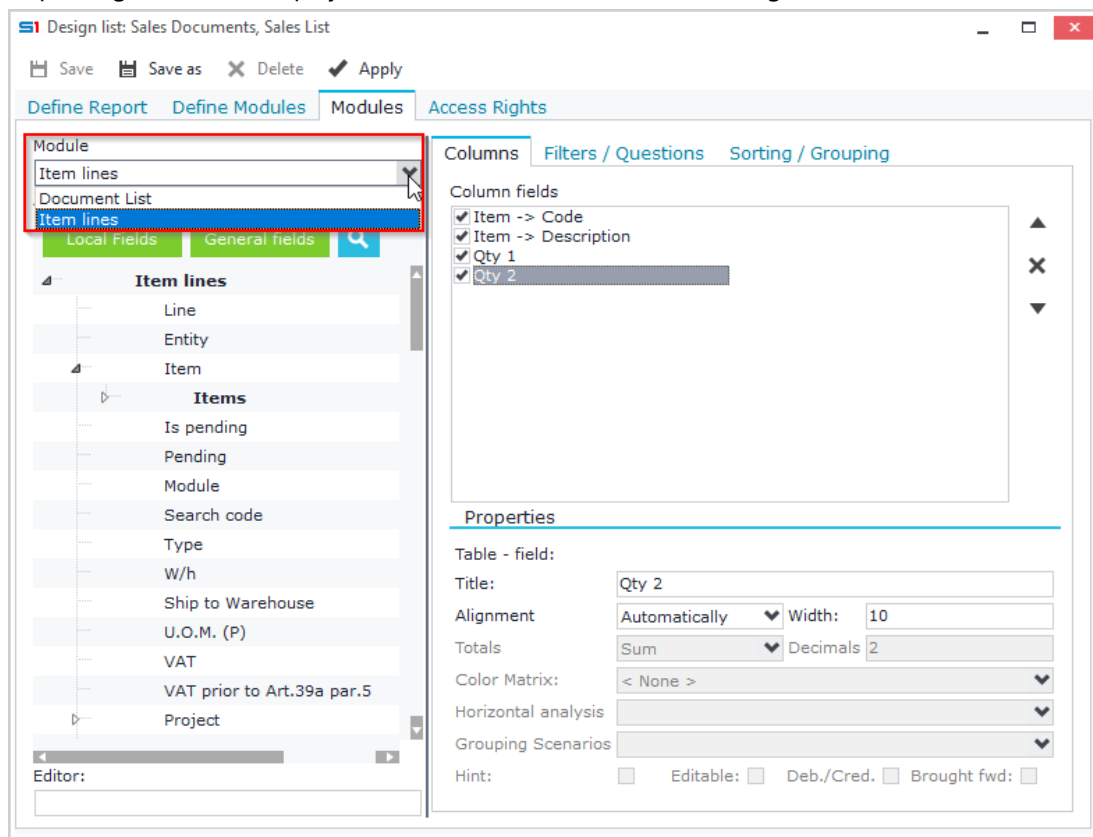




Figure F3

G. User-defined modules

The browsers provide the ability to design child bands (detail) using user-defined tables. That is, you can create SQL statements that will populate a datatable, that can be later used as child table of the browser.

This is done from the module “**Define Modules**” following the instructions below:

- Click the  button (next to button .
- Inside the window that opens, click on “**Add**” to create a new table (e.g. MyLines).
- Enter your sql statement in the right panel box “**SQL**” as in Figure G1. The SQL statement uses the expression **:ParentTable.ID** to link to the parent table of the browser, where ID is the primary key of the parent table.
- Click on “Create fields from SQL” inside “Fields” tab to create the fields and add captions (Figure G2).

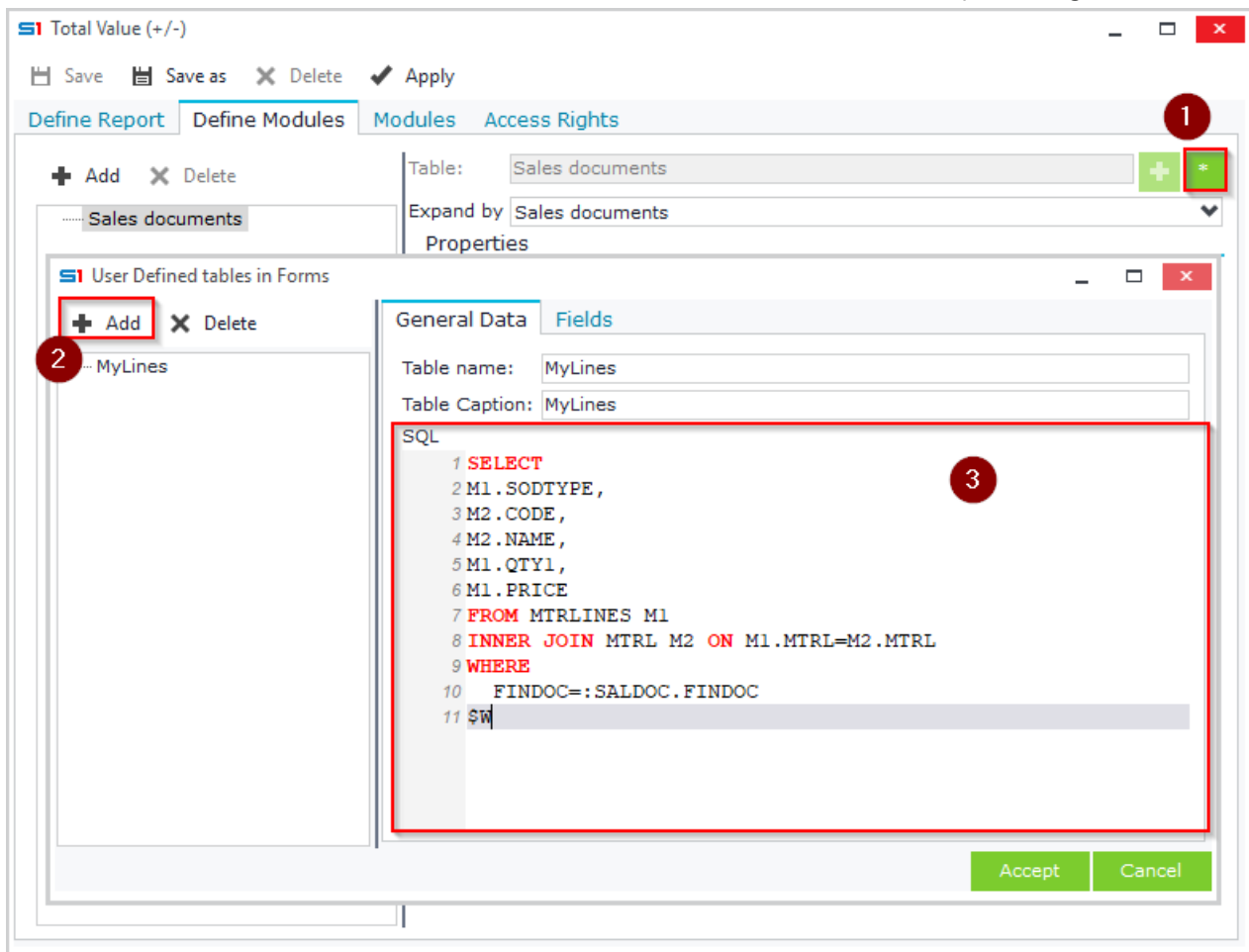


Figure G1

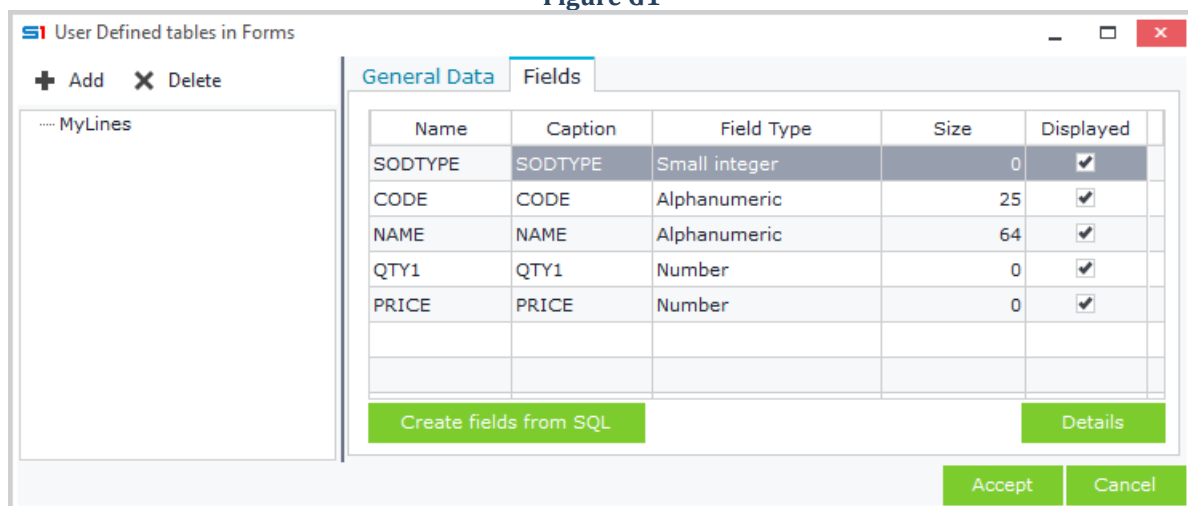


Figure G2

Add a <New module>, select the custom table from “*User-defined view tables*” (Figure G3) and follow the steps of the previous section to add new columns in the browser (Figure G4).

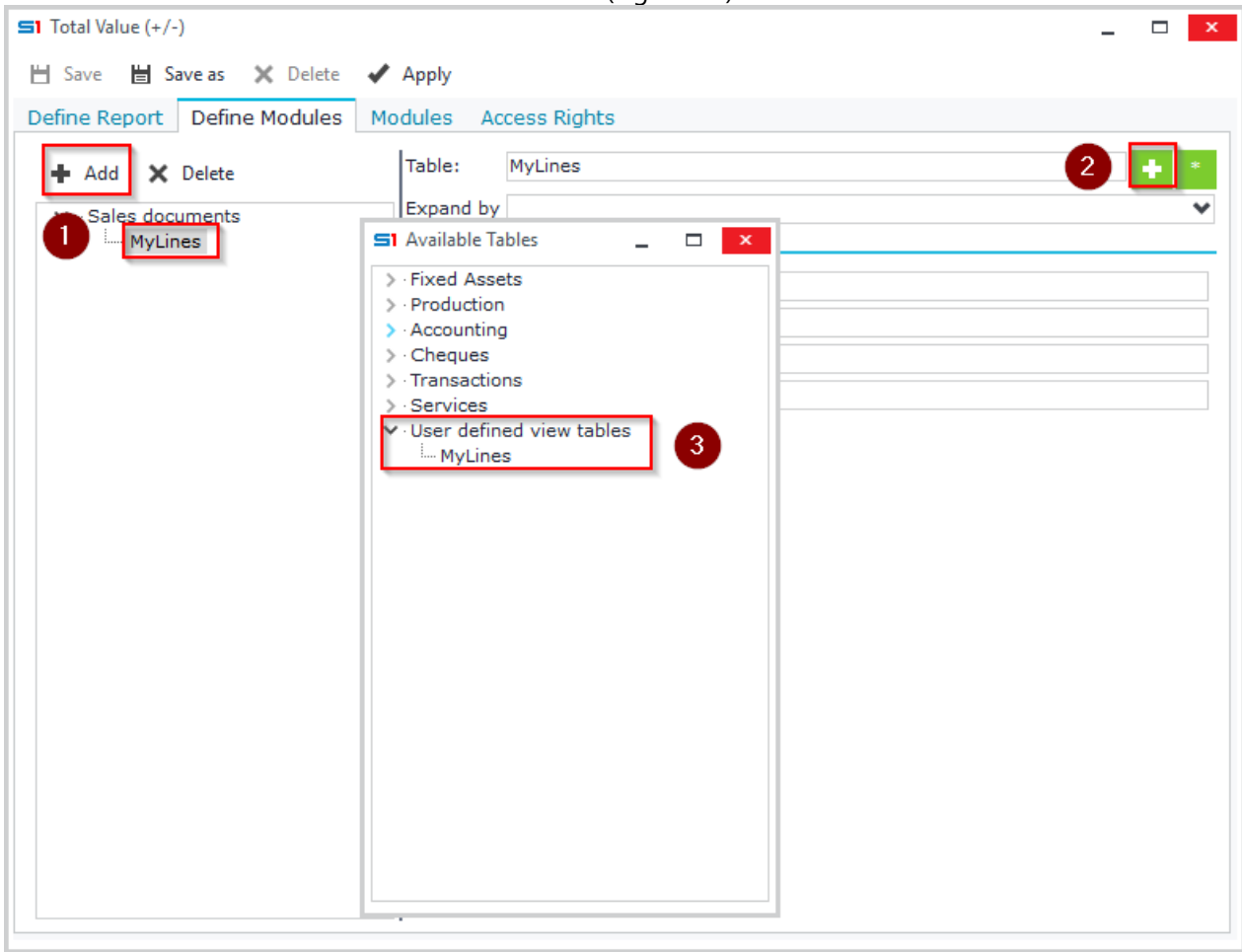


Figure G3

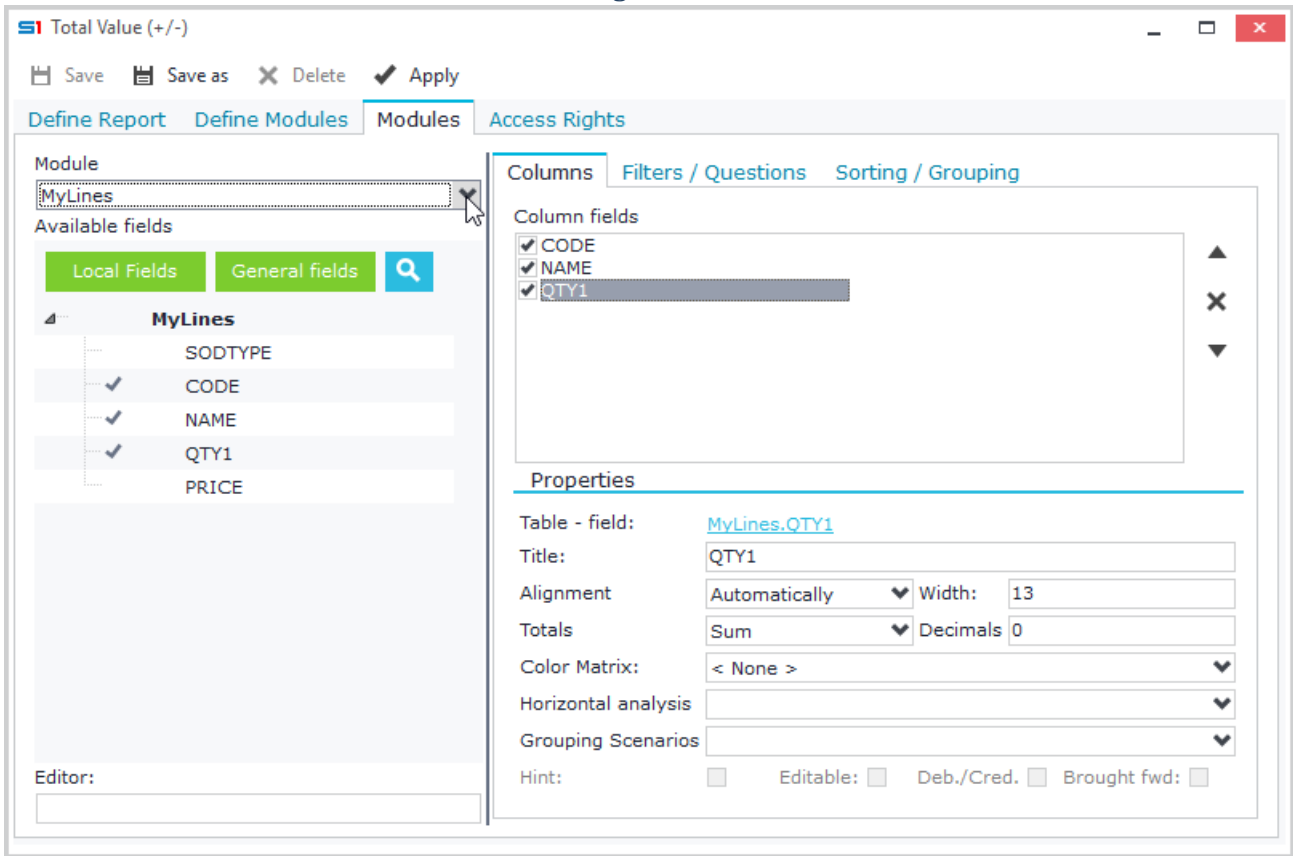


Figure G4

Example

Create a browser in customers object that will display in a 2nd band the financial data of customers per year and month.

Create a new table and enter the SQL statement as in Figure G5.

Then create the fields through the "**Fields**" Tab, enter the captions and the editors FISCPRD and PERIOD of the corresponding fields (Figure G6).

Insert a <New module> and select the new table through the "**User defined tables**".

In the "**Modules**" tab of the browser, select the second module from the combo box and add the fields in the right panel "**Columns**". Save the browser under a name of your choice.

Run the browser and the results will be displayed as in Figure G7.

User Defined tables in Forms

+ Add -X Delete

Table name: CusFinancialData

Table Caption: CusFinancialData

SQL

```
1 SELECT FISCPRD,
2 PERIOD,
3 LCREDIT,
4 LDEBIT,
5 LTURNOVR,
6 COMPANY
7 FROM TRDBALSHEET
8 WHERE TRDR=:CUSTOMER.TRDR
9
```

Accept Cancel

Figure G5

User Defined tables in Forms

+ Add

Name	Caption	Field Type	Size	Displayed
FISCPRD	FISCPRD	Small integer	0	<input checked="" type="checkbox"/>
PERIOD	PERIOD	Small integer	0	<input checked="" type="checkbox"/>
LCREDIT	LCREDIT	Number	0	<input checked="" type="checkbox"/>
LDEBIT	LDEBIT	Number	0	<input checked="" type="checkbox"/>
LTURNOVR	LTURNOVR	Number	0	<input checked="" type="checkbox"/>
COMPANY	COMPANY	Small integer	0	<input checked="" type="checkbox"/>

Editor: PERIOD

Formula:

Create fields from SQL Details

Accept Cancel

Figure G6

Customers

Code	Name	T.R.No.	Primary Address
101	Salesconsul GmbH	073586841	Rosa-Luxemburg-Strasse 7
102	Nick Panterben	23	124
103	Holding AE	345345	Switzerland 7
104	Oikoset AU		Solon 92
105	Thunder Tiger ABEE		Stegis 36
106	Shipping SA	140398745	THEodotou 1
107	SoftOne AE	999863881	Leoforos Posidonos 4
108	Kouskoukis George	111	Lykoudis 12

FISCPRD	PERIOD	LCREDIT	LDEBIT	LTURNOVR
2016	December		200	
2017	January		166	135
2018	January		539.613	438.710
2018	March		429.172	348.920
2018	April		258.603	209.352
2018	May		29.771	24.204
2018	June	1.287		
2018	November	120		
2018	December	1.250	3.561	2.895
		2.657	1.261.086	1.024.216

Figure G7

3. PRINTOUT FORMS

- A. [General Printing Features](#)
- B. [Internal Printout Forms](#)
- C. [Ms-Word Printout Forms](#)
- D. [Ms-Excel Printout Forms](#)
- E. [Label Printout Forms](#)
- F. [Crystal Reports Printout Forms](#)
- G. [Automated Jobs](#)

Overview

Printout form is a report tool that is used for printing data through SoftOne objects. This means that this tool auto inherits the data structure (tables, database views, etc.) of an object and makes the associated fields available for use in report by simple drag and drop. Printout forms run from the button "Print" while in object view of a record or through right click on a record from browser.

A. General Printing Features

A.1 Printer Settings

The printer mode (Image or Draft) is set through the "Printers" tab of system settings (Figure A1). Double click on a printer in order to change its printing mode from Image to Draft.

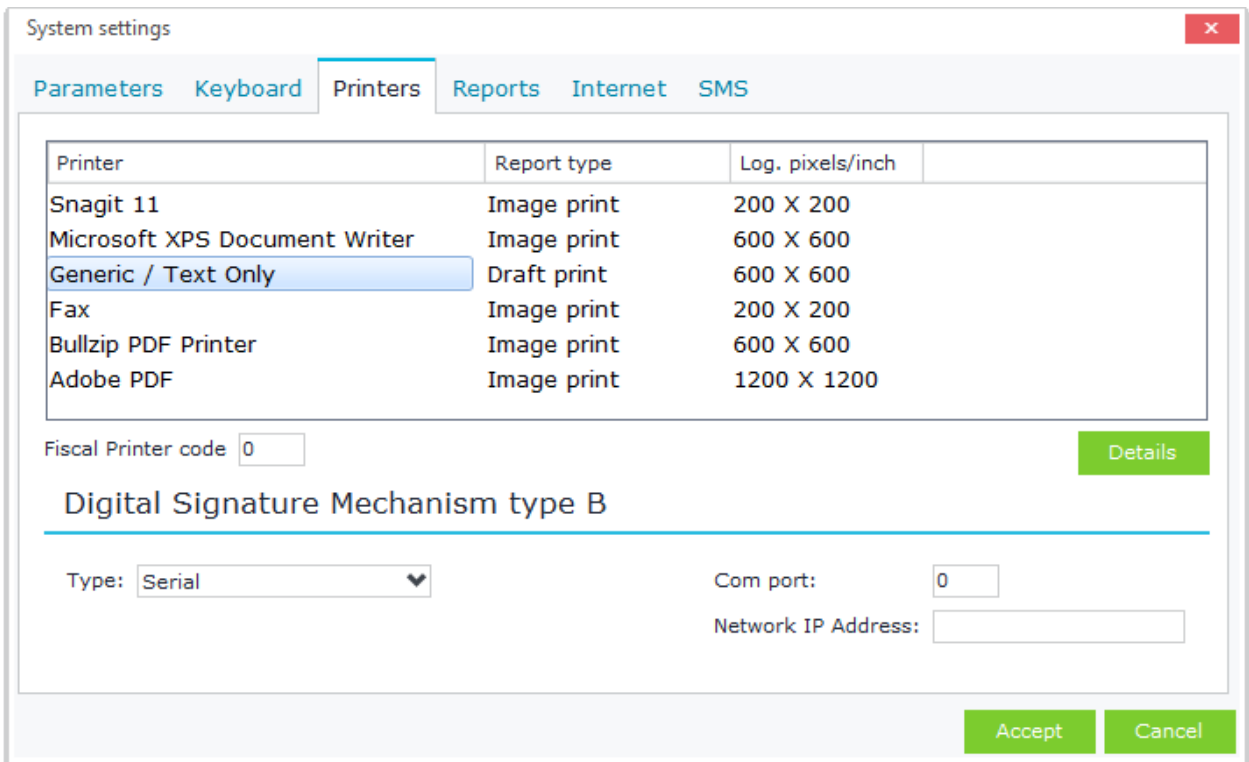


Figure A1

If you select the Draft mode, then the option button "Details" (in lower left corner) is activated. Inside the window that appears by clicking on the "Details" button, you can set the Dot Matrix printer settings. Default printing values are inserted through click on the button "Default values" (Figure A2). These settings are applied perworking station and are saved in the Xplorer.cfg file. This file is saved in the "User Profile Directory" folder. Link for this folder can be found under the "About" button.

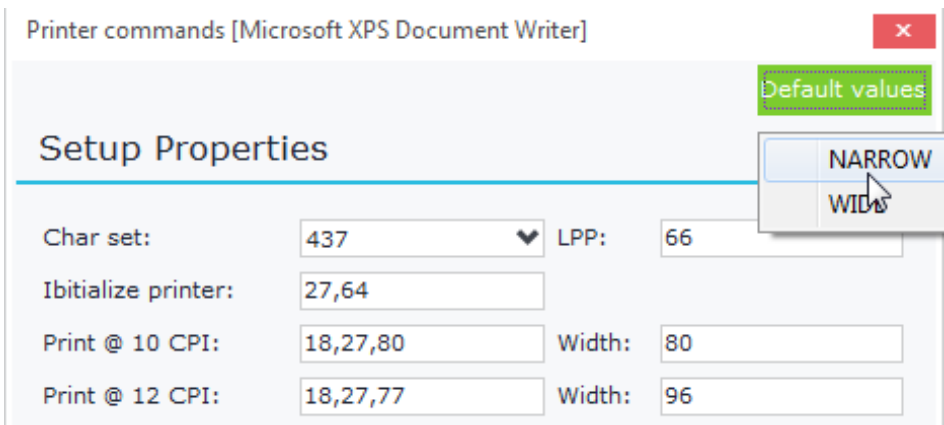


Figure A2

A.2 Available Printing Methods

Data viewed in screens can also be sent to the following media:

MS-Excel: Opens and prints data in new Microsoft Excel file. Microsoft Excel must be installed on the client computer.

MS-Word: Opens and prints data in new Microsoft Word file. Microsoft Word must be installed on the client computer.

E-mail: Sends email using the selected data. Through a settings screen that is displayed you can set the file format as well as whether the data will be sent in the email body or in an attachment file.

Spreadsheet File: Exports the data in xls format. It does not necessarily require the installation of Microsoft Excel, but any spreadsheet application.

Document File: Exports data in doc format. It does not necessarily require the installation of Microsoft Word, but any text processing application.

Ascii file Windows: Saves data in ASCII file using Windows 1253 encoding.

Ascii file DOS: Saves data in ASCII file using 437 character set.

Html format: Saves data html file.

SoftOne MetaFile: Saves data in xpr SoftOne format file. These files run through SoftOne application, directly from Windows Explorer through <double click>. If SoftOne is running, then the executed report is displayed on the application screen, else the operator is subjected to login process and immediately afterwards the report is displayed on the screen. In the last case, SoftOne runs in "viewer mode" and does not contain any other job.

A.3 Printout Form Types

The available printout form types (Figure A3) are the following:

Internal use: Design printout forms using the internal report tool.

External (Word): Design printout forms using MS Word.

External (Excel): Design printout forms using MS Excel.

Fiscal printer: Retail receipts printout forms that are printed through POS Fiscal Printers.

Label: Design printout forms for labels print using the internal report tool.

Crystal Report: Design printout forms using Crystal Reports. Crystal reports runtime must be installed.

Internal (Fast Report): Design forms using the embedded tool Fast Reports. No need to install extra application.

Label (Fast Report): Design label forms using the embedded tool Fast Reports. No need to install extra application.

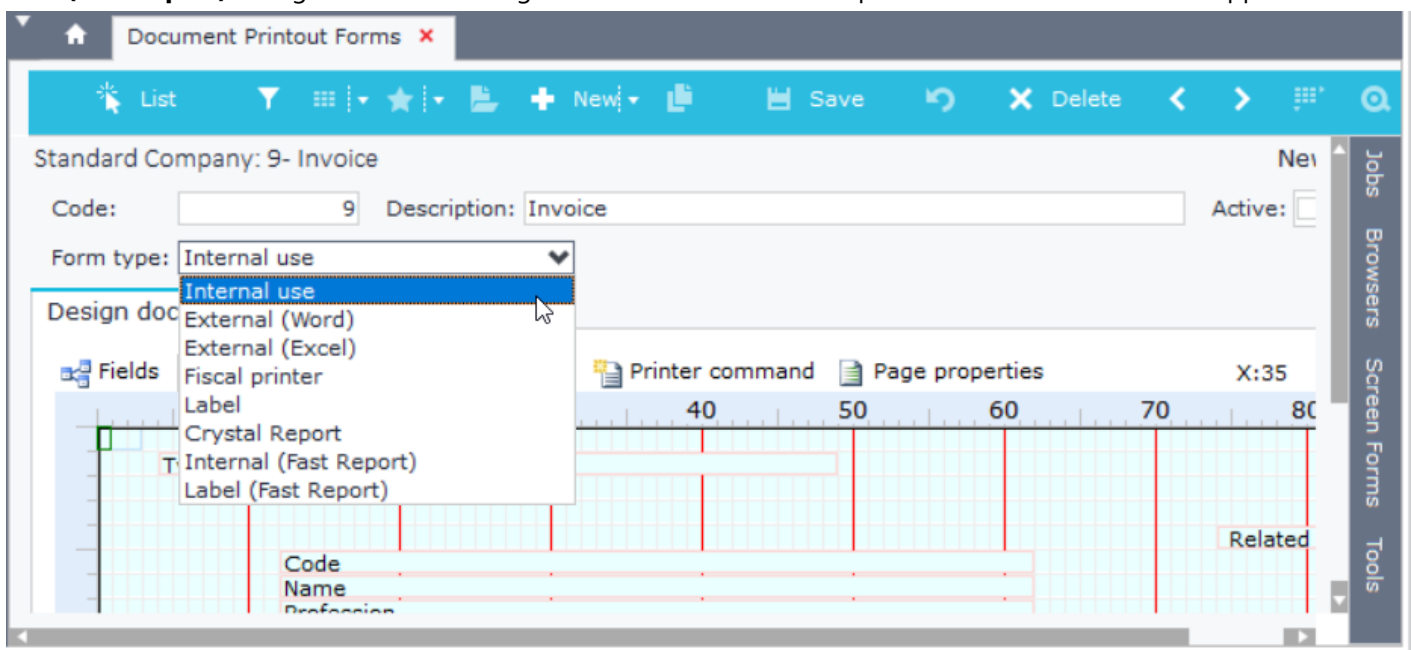


Figure A3

A.4 Import / Export Printout Forms

Printout forms can be exported and imported in other installations using XXF files.
For internal forms you can also use PTF file extension.

XXF File

This option saves the structure of the printout form along with the object that uses it, the code and the name.
A form is exported through the option **“Export to .xxf file”** of the browser list right click menu (Figure A4.1).
Forms are imported through the User or System option **“Import from .xxf file”** (Figure A4.2).

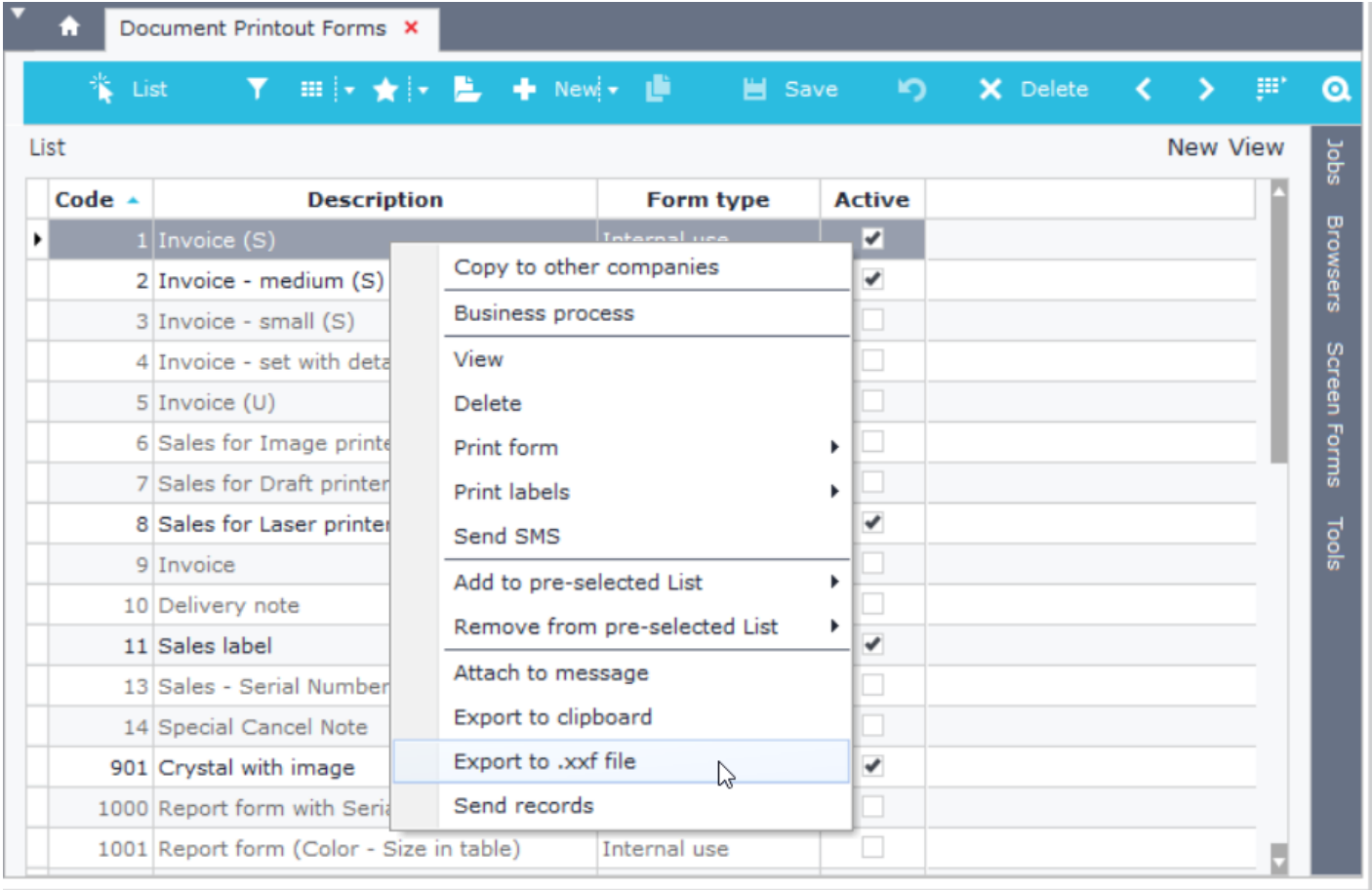


Figure A4.1

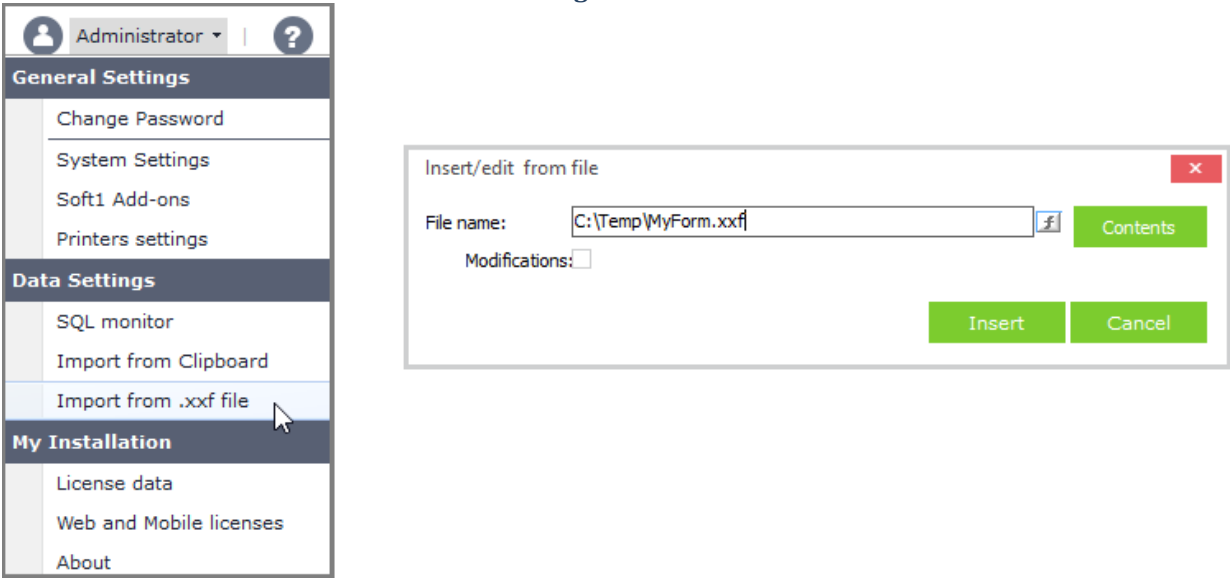


Figure A4.2

PTF File

This option is available in internal forms and saves only the structure of the printout form. Use this option when you need to copy a form from one object to another, that has similar fields (SALDOC to PURDOC). Edit the ptf file using a text editor, alter any table or field names and then import it in any other object. Notice though that background images are not saved.

Internal printout forms can be exported to ptf files through the right click option “**Save to file...**” that is found when you locate a printout form record (Figure A4.3).

Ptf files are imported into printout forms using the right click option “**Load from file...**” (Figure A4.3).

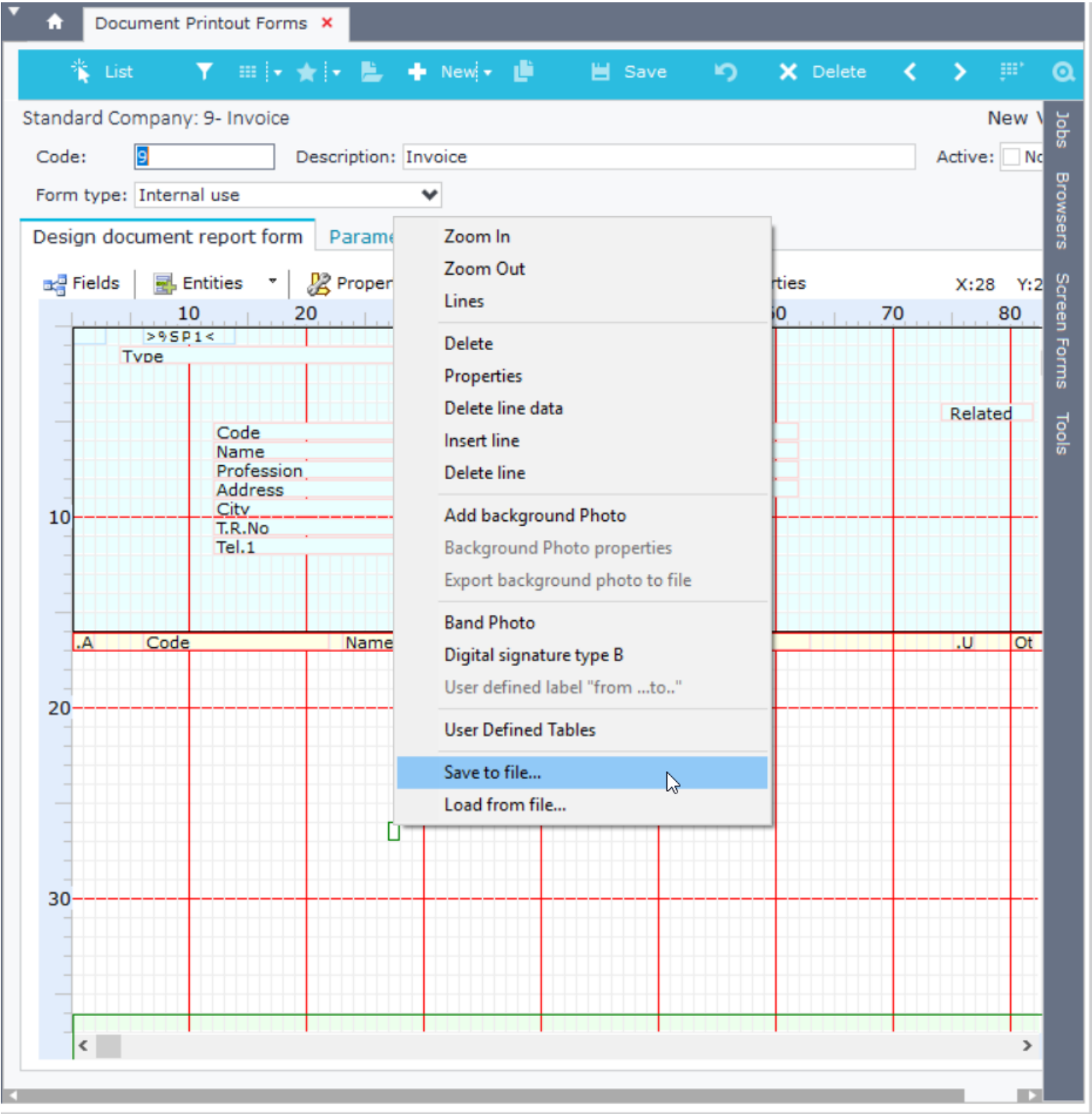


Figure A4.3

B. Internal Printout Forms

Printout forms can be designed using the following modes:

Draft: Reports printed on dot matrix printers. In this case the printer must necessarily be defined as draft and the "Printer commands" settings must be filled as in [Printer Settings](#).

Image: Reports printed on laser, inkjet printers.

The design environment is the same for both modes. What differ are the settings and printer commands that can be used.

B.1 Sections

Internal forms are divided in three sections (Figure B1):

Beginning of page - Header (light blue): It contains the header data of a report (e.g. customer data, date and number of a document, etc.).

Recurrent section (yellow): It contains the recurrent part of a report (e.g. item lines of a document).

End of page - Footer (light green): It contains the footer data of a report (e.g. prices-quantities totals, comments of a document).

Figure B1

B.2 Toolbar

The toolbar (Figure B2.1) which appears in the design of internal forms has the following commands:

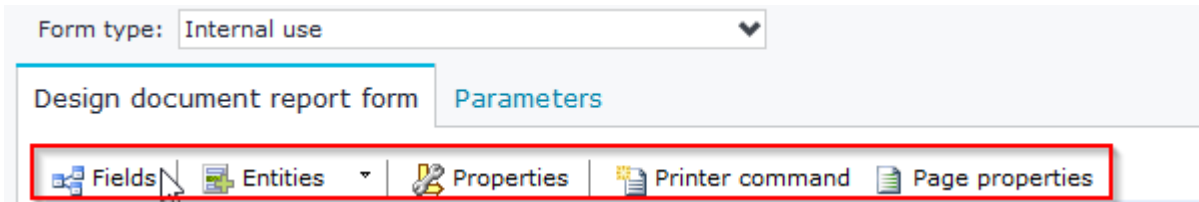


Figure B2.1

Fields: Displays the fields that can be inserted in the form through double click or drag and drop (Figure B2.2).

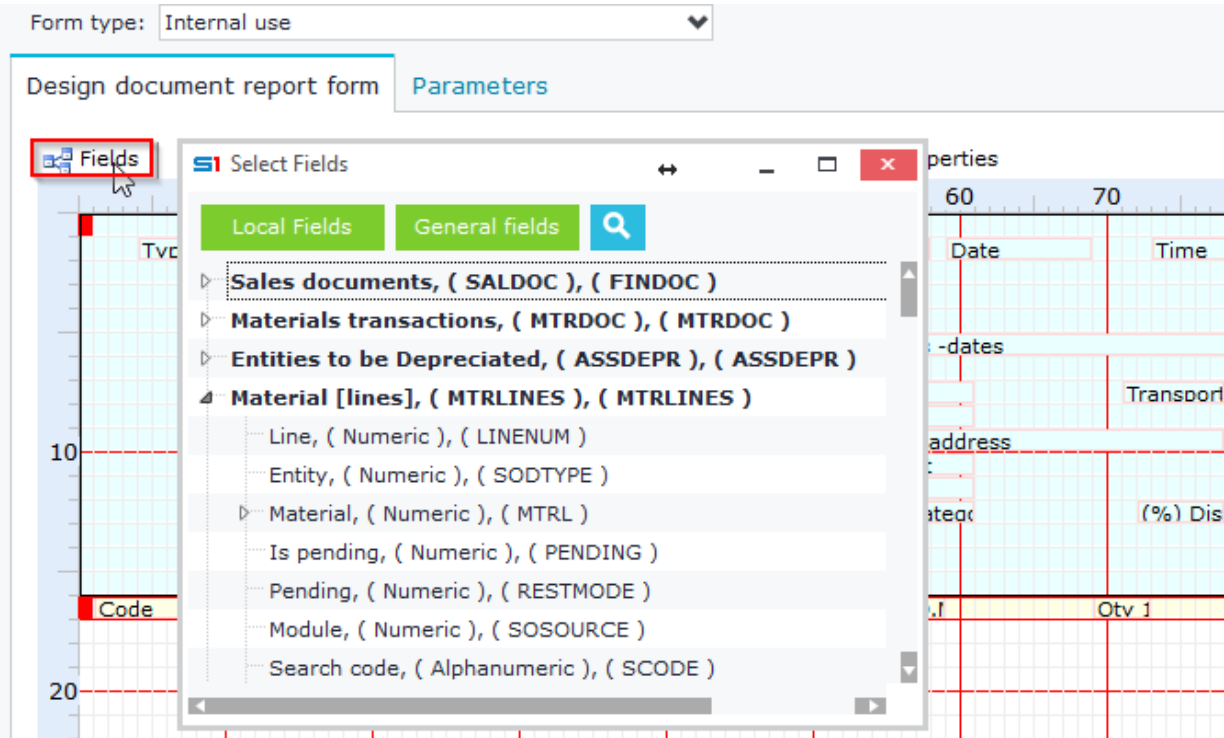


Figure B2.2

Entities: Displays the available sections of the form (Figure B2.3)

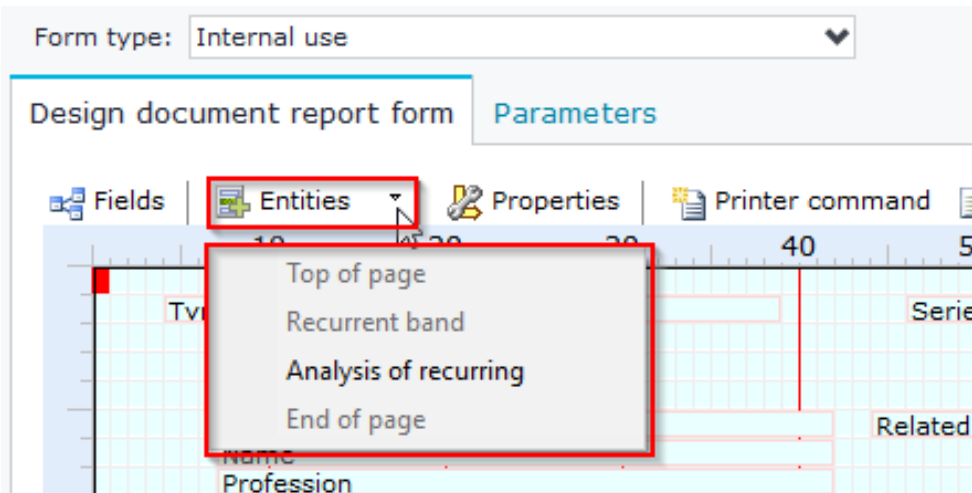


Figure B2.3

Properties: Displays the Properties window for the selected point on the form (the option is also available by right-clicking on the form). Figure B2.4 shows the properties of the field "MTRDOC.TRUCKS" of sales documents.

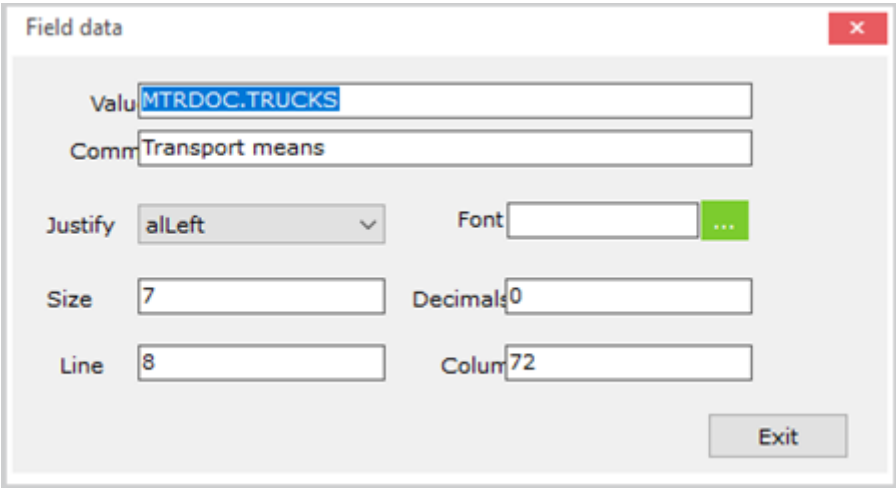


Figure B2.4

Printer command: Window for adding printer commands on the forms. (Figure B2.5)

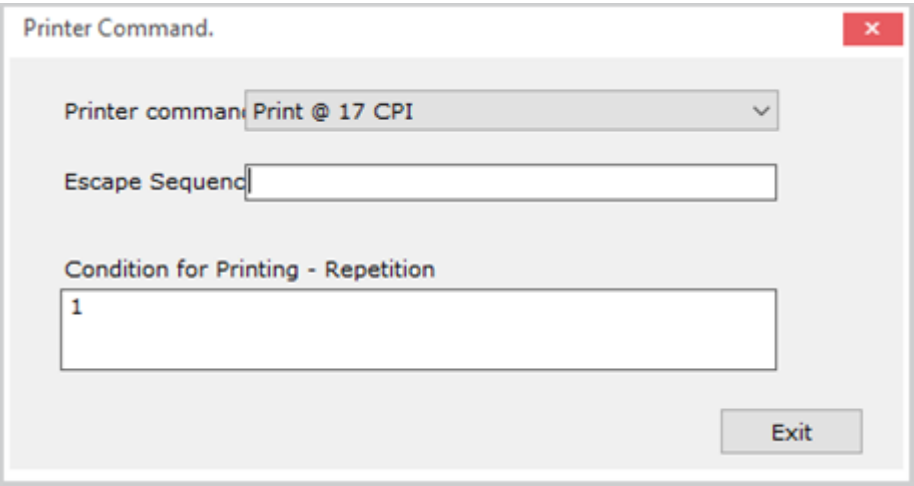


Figure B2.5

Page Properties: Printout form Properties window. The options "Blank lines above", "Blank characters on the left" and "Useful page lines" are used in label printout forms (Figure B2.6).

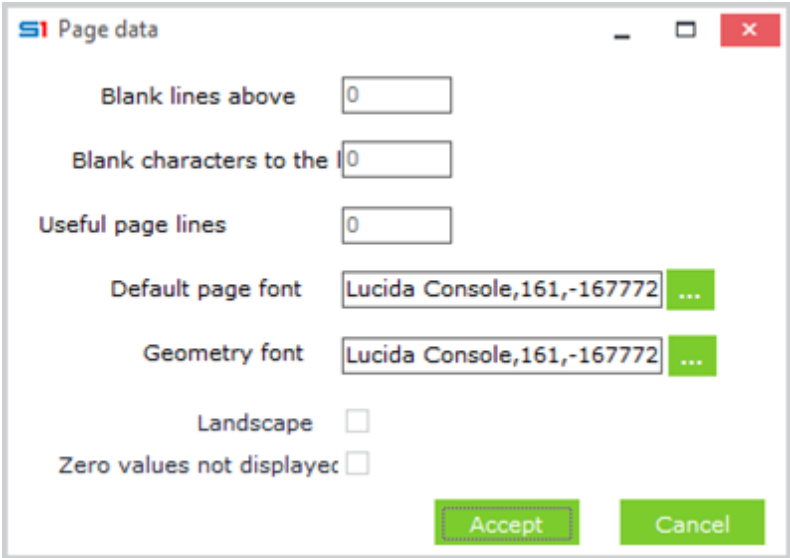


Figure B2.6

B.3 General Form Operations

Right clicking in any place of the form to display the list of available options (Figure B3.1):

Zoom In / Out: Zooms the form

Lines: Shows / Hides grid lines.

Delete: Deletes the selected field, zone, image etc.

Properties: Displays the properties of the selected field, zone, image etc. Same as the toolbar button "Properties".

Delete line data: Deletes all fields and texts of the selected line.

Insert line: Inserts a line at the selected point.

Delete line: Deletes the selected line.

Add background Photo: Inserts a background picture (jpg or bmp format).

Background Photo properties: Picture properties configuration

Export background photo to file: Exports image as file

Band Photo: Inserts an image at the selected point

Digital signature type B: Inserts a digital signature at the selected point. It works only in the footer and only in internal forms. (Greek Legislation)

User defined label "from ...to...": Inserts page numbers. Available only in footer.

User defined Tables: Inserts a User defined table to the available tables of the forms.

Save to file....: Saves form to a file in ptf extension, which can be edited using any text editor.

Load from file ...: Imports a form from ptf file.

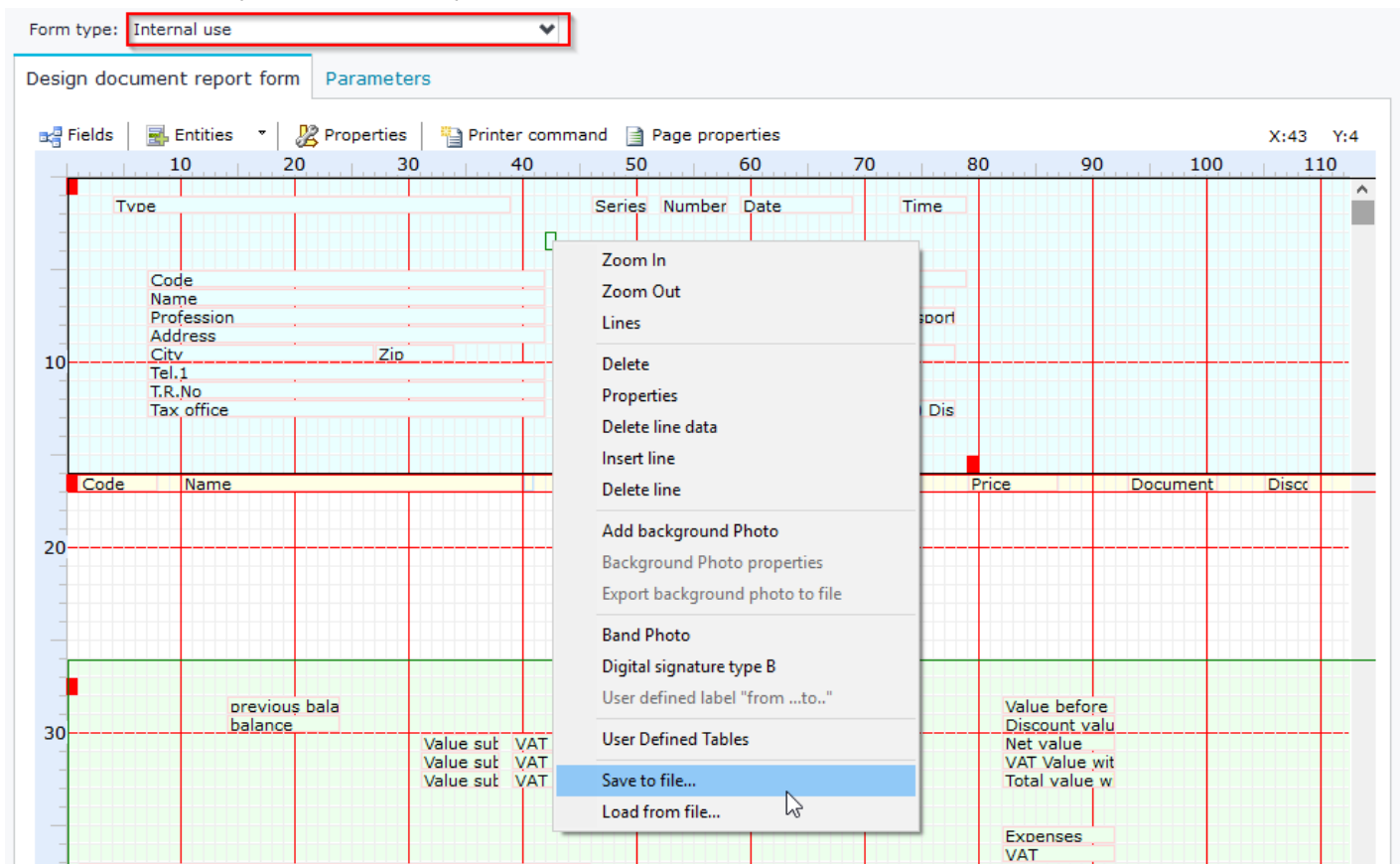


Figure B3.1

B.4 Page Size

The size of the form is set through the right click option "Properties" which is available at any point between the zone of the recurrent part (white color) and the footer of the form (Figure B4.1).

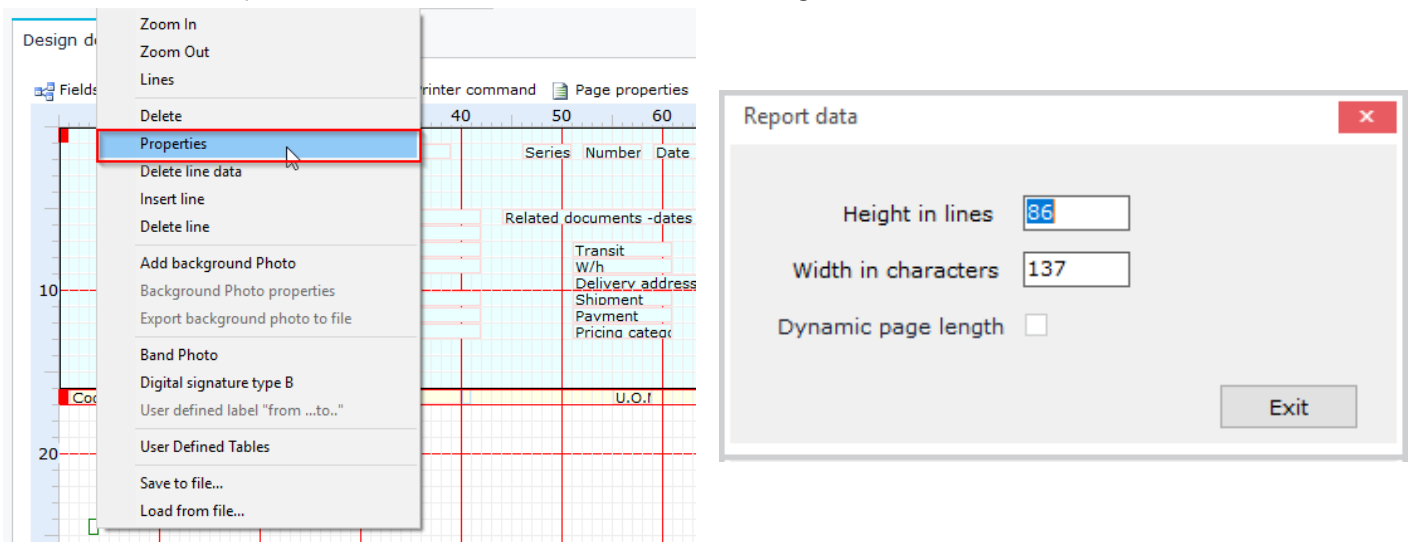


Figure B4.1

In case of using a document without distinctive paging (e.g. roll paper of a cash device), set the parameter "**Dynamic page length**" to "Yes" so that the footer is printed only once, at the end of the report.

Once page dimensions setting is completed, the ruler that surrounds the design screen of the form will be adjusted accordingly (Figure B4.2).

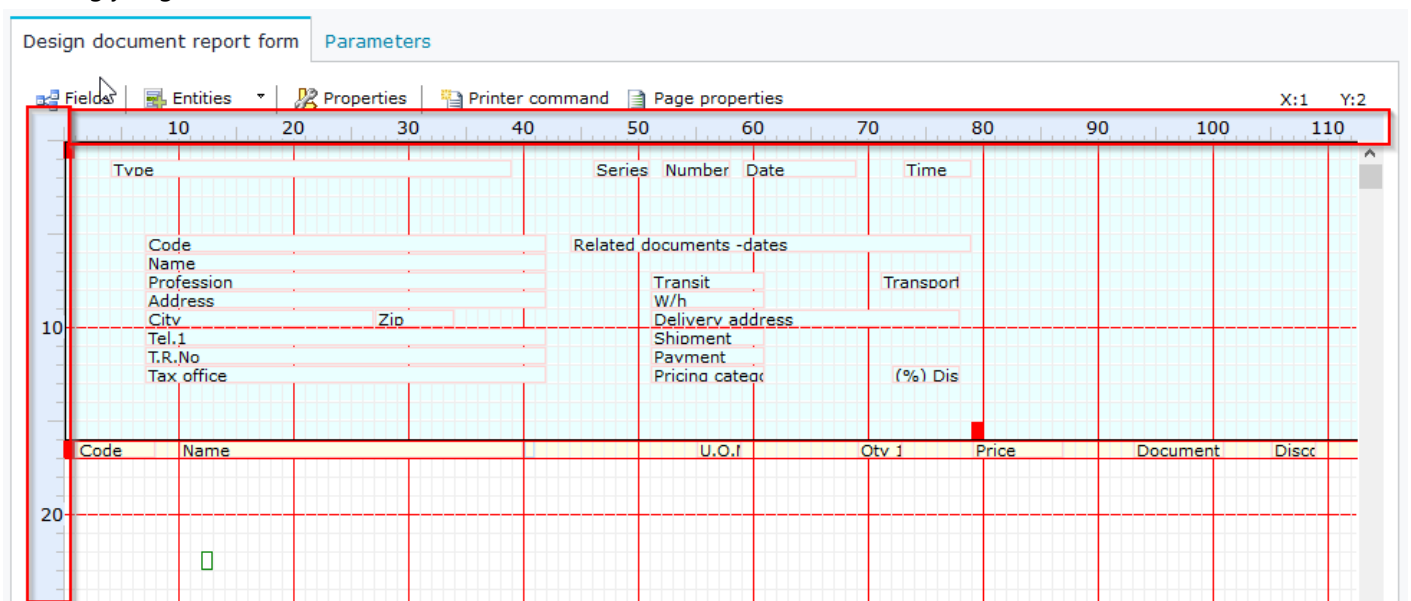


Figure B4.2

Basic Dimensions Parameters

Laser Printouts: Lines height: 85, Characters width: 137

Draft Printouts: Lines height: 66, Characters width: 137

Note : The actual dimensions of the printout in centimeters depends additionally on **geometry font** for image printouts (Figure B2.6), while for draft printouts it depends on the used **printer commands** (Figure B2.5).

B.5 Draft Printout Fonts

The font size in draft printouts is set by printer commands. A printout form can have one or more printer commands, which apply different font sizes per line or module. Printer commands (red squares) are added in forms by clicking on the button "Printer command" of the toolbar. Inside the properties window you can apply the appropriate setting (Figure B5.1). Printer command settings are applied in form until you use a new one. For example, if there is a printer command of 12CPI, then all elements of the form that follow will have this size. The size is applied in combination with the special settings that have been set in the draft printer.

The width of the line you have set at form settings determines the maximum number of characters per line. Suppose you set 12CPI size and the maximum width according to the settings, is 96 characters. If your data exceed this width, line folding will be applied, even if the overall width of the printout form is larger.

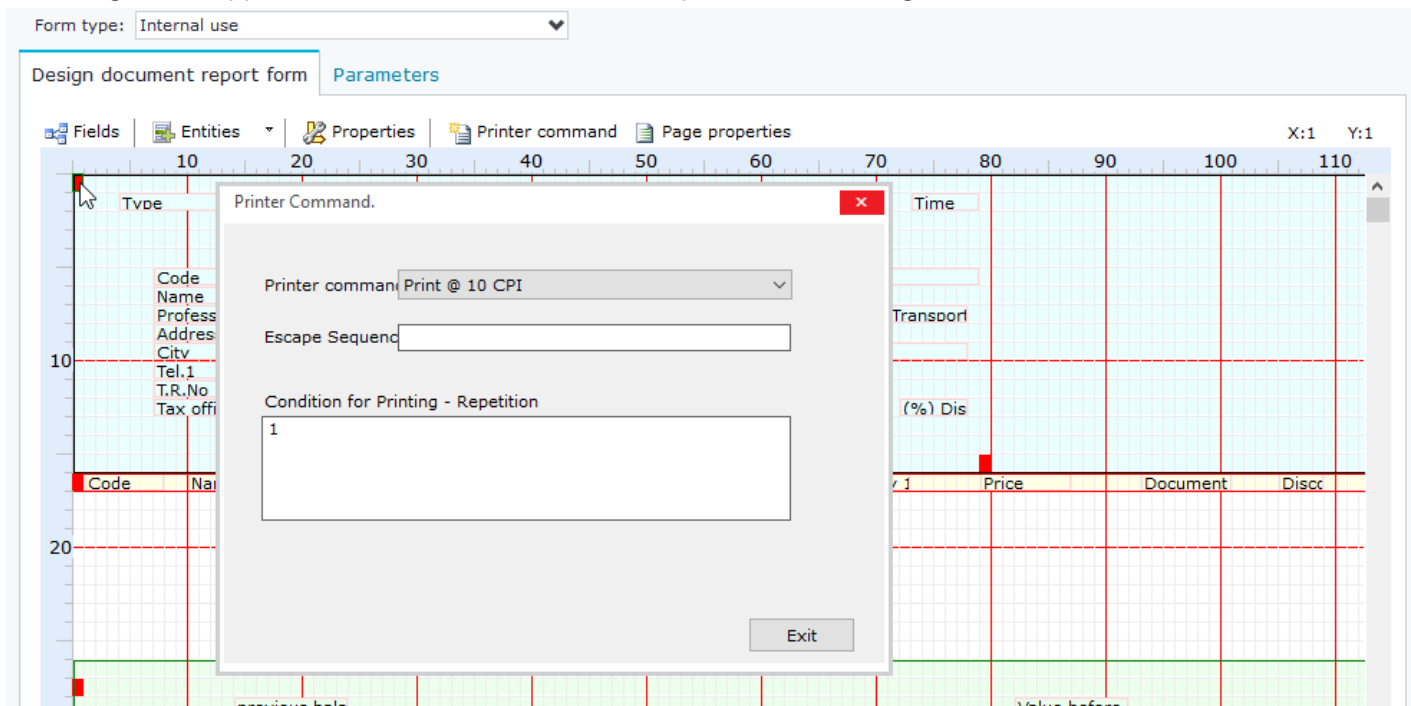


Figure B5.1

B.6 Image Printout Fonts

The font size in image printouts is initially set by "Page properties" button on the toolbar. Specifically, through page properties you can set the fonts to be used in the designed form.

The option "**Default Page Font**" determines the default font for each field or text you enter.

Special settings regarding the printing method (font type and characters size) for the different sections of the form or even for specific fields can be set through their "Properties".

The option "**Geometry font**" sets the font type and size that will determine the size of the grid cells of the form.

Note that the actual size of the printout is enlarged proportionally to the size of the selected geometry font.

Essentially, the geometry font is the key parameter that determines the space available for printing each character on the page. It is obvious that the default page font cannot be larger than the geometry font because this would lead to characters overlapping during printing.

Note: If these parameters are not entered, then the default fonts are used (Lucida Console 8pt).

B.7 Recurrent Zone

The first thing you have to define before inserting fields in the recurrent part of the form is the table that will be used in this section. Right click the recurrent part (yellow line), click on "Properties" button to display the menu of Figure B7.1. The recurrent section contains prints the data entered in the yellow line.

Section properties are the following:

Lines: Indicates the section number of lines

Table: Name of the table that will be used for displaying data / fields in this section

Filter: If used, it filters the data of the table according to the expression. Expression must be entered using only the name of the field (not the table). For example, if you want to filter item lines that have quantity larger than zero, you have to enter **QTY1>0**.

Sorting: If used, it sorts the data of the table according to the selected field. The field must be added in the same way as it is displayed when you add it inside a section of the form.

Pack fields: If used, it groups the data according to the selected fields. Fields are separated with semicolons.

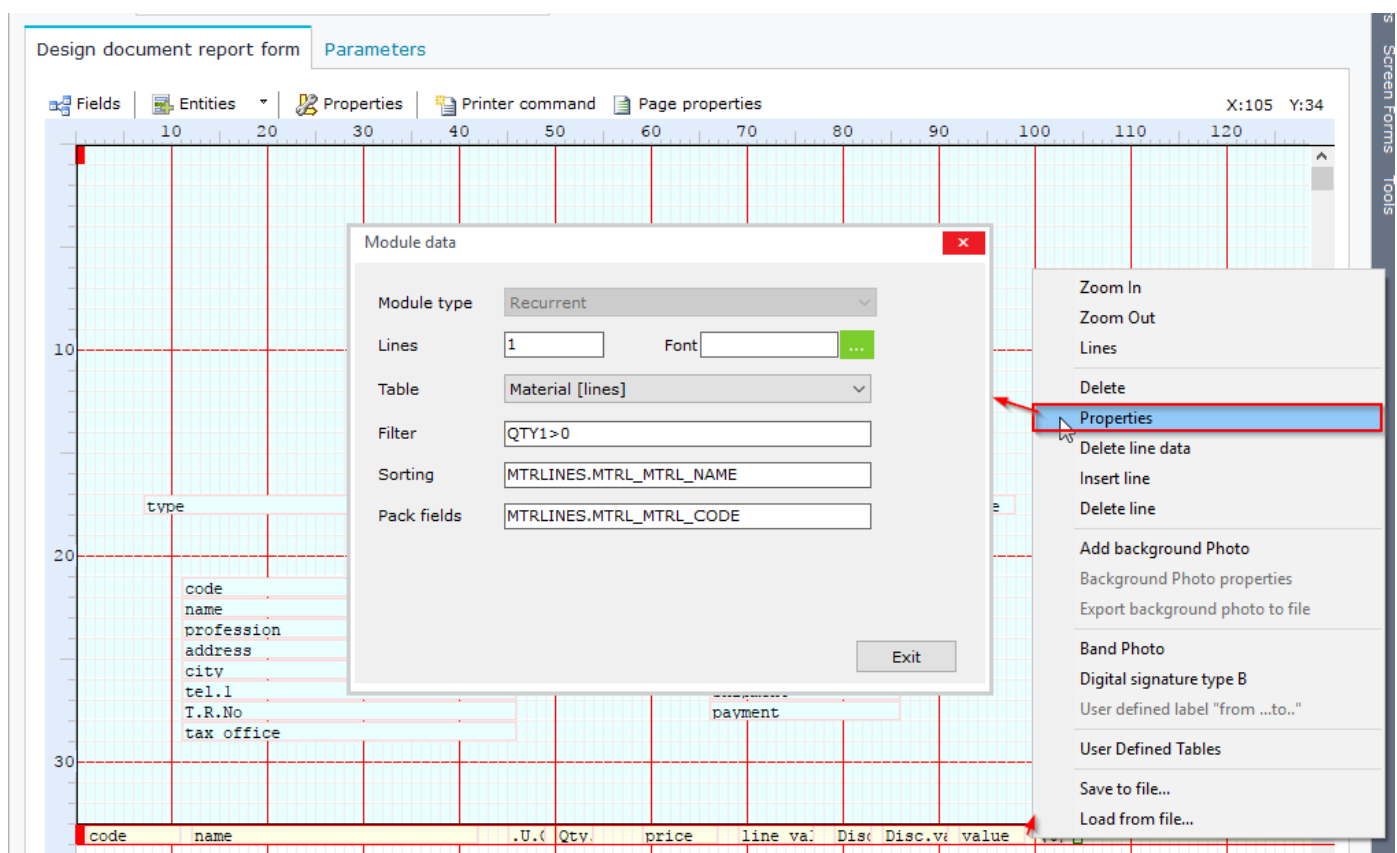


Figure B7.1

B.8 Band Image

Images are added into form sections through the right click option "Band Photo". Inside the properties of the image (Figure B8.1) you can define the path and the filename of the image in the Value textbox.

Item Images are referenced using the command **\$MTRLINES.MTRL;0** while images from item related files are displayed using the command **\$MTRLINES.MTRL;**.

Figure B8.1

B.9 QR Code

QR Codes are added into form sections through the right click option "Band Photo". Inside the properties of the image (Figure B9.1) enter the name of the field that contains the QR code using the prefix character ~ .

The example of Figure B9.1 will display the data of the field MTRDOC.SOSIGNQR as QR Code.

Figure B9.1

C. Ms-Word Printout Forms

C.1 General

MS Word printout forms are designed through Microsoft Word application using the option "External (Word)" in the form type. (Figure C1.1).

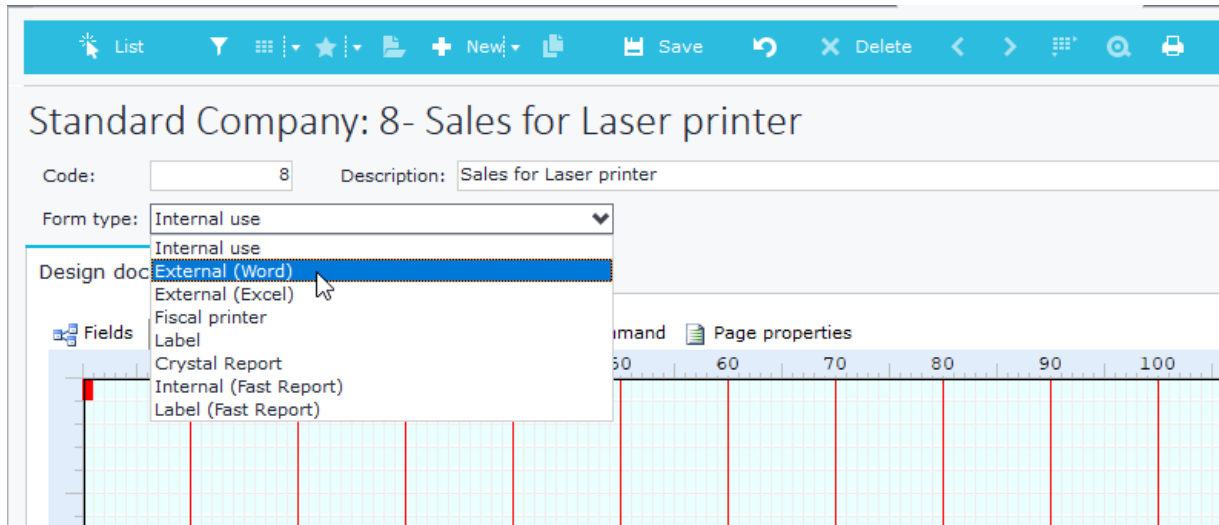


Figure C2.1

A different toolbar is displayed (Figure C1.2) that lets you create a new word document, modify or delete the existing one and insert Fields from the current object to your MS-Word form.

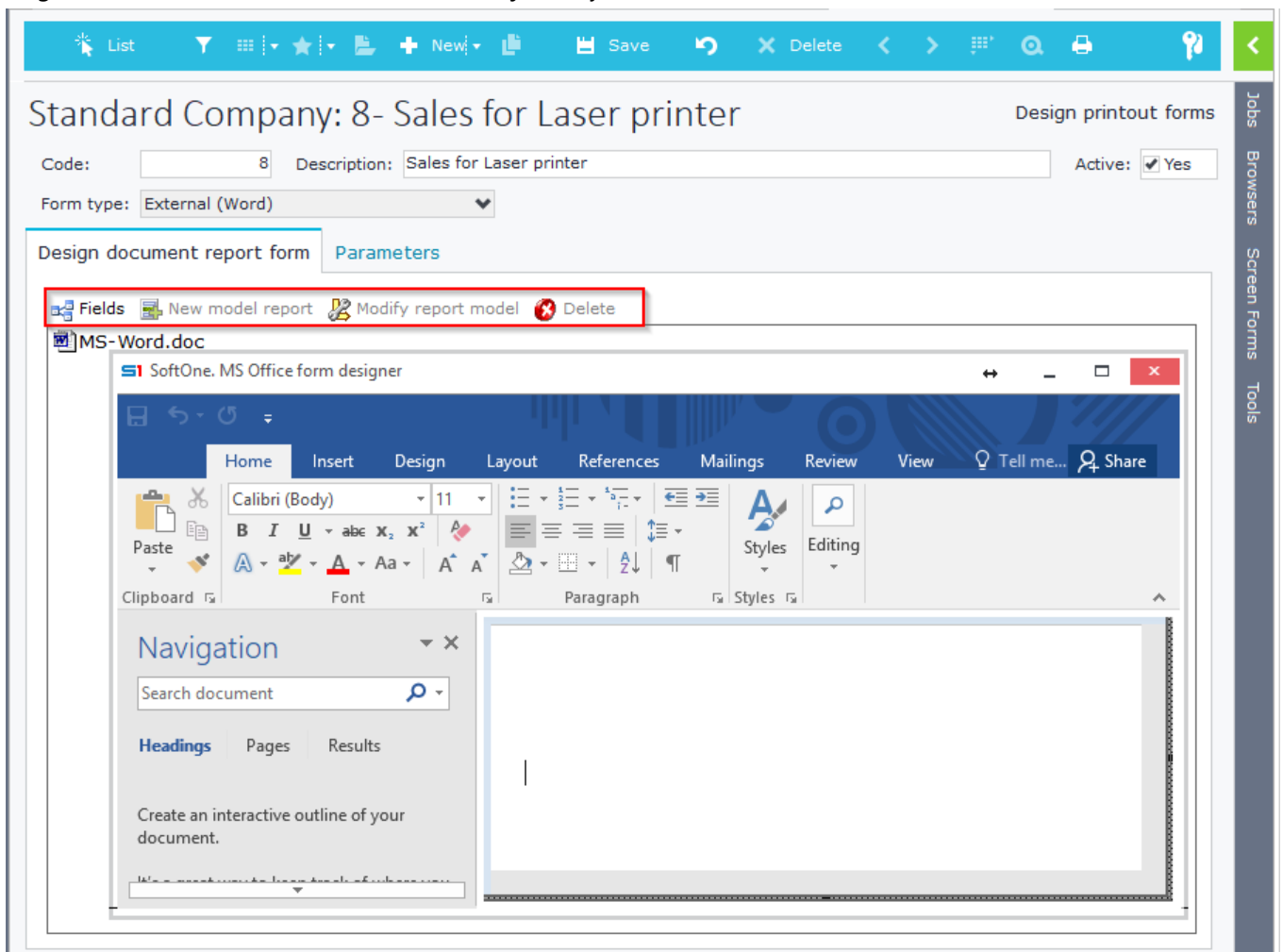


Figure C2.2

C.2 Form Design

Fields are inserted inside the form by double-clicking on them from the Fields window.

Word Forms, are divided in the following sections:

- Page Header**
The fields of this section must be inserted inside the document header (Figure C2.1).
- Recurrent**
The fields of this section must be inserted in a MS-Word table (Figure C2.2).
The first column of the table must contain the symbols { @ and the last column the symbols @ } .
The SoftOne table that will be used for recurrency is entered inside the hyperlink Address box of the first two symbols. The command is: **Table:"TableName"** (Figure C2.3).
The last columns symbols @} of the table do not need a hyperlink.
- Page Footer**
Page Footer is defined in MS Word form by inserting a new textbox at the bottom (not document footer) of the document page and grouping the items inside it.
- Report Footer**
Definition of this section is done by adding a hyperlink to a Page Footer (not document footer) using the command "**XTR: LP_FOOTER**" in the Address field.

Company Info

COMPANY INFO

CODE: Code
NAME: Name
PROFESSION Profession
ADDRESS: Address

TRANSIT: Transit
SHIPMENT: Shipment
DELIVERY ADDRESS: Delivery address Delivery town
PAYMENT: Payment
RELATED DOCUMENTS: Related documents

CODE	NAME	U.O.M	QTY	PRICE	VALUE (NO DISCOUNTS)	DISC. %	DISCOUNT VALUE	VALUE	VAT %
{@Code	Name	U.O.M (P)	Qty 1	Price	Document line value (no discounts)	Disc. 1(%)	Disc. 1 value	Value	Percent ass (%)(@}

Figure C2.1

{@Code	Name	Qty 1	Price	Disc. 1(%)	Disc. 1 value	Value	@}
--------	------	-------	-------	------------	---------------	-------	----

Figure C2.2

Edit Hyperlink

Link to: Text to display: {@

ScreenTip...

Look in: INetCache

Content.MSO
Content.Word
Low
Virtualized

Bookmark...

Target Frame...

Address: TABLE:MTRLINES

Remove Link

OK

Cancel

Figure C2.3

C.3 Filters – Sorting – Grouping

Data of the recurrent part of a Word can be filtered, sorted, and grouped, by adding extra commands to the hyperlink of the first column ({@}). The syntax is the following (Figure C3.1):

TABLE:<Table>,<FilterField 1> AND<FilterField 2> OR<FilterField 3>,<SortField>, <GroupField>

The fields used in the above command must also be inserted in the form, in order to work properly. If you do not add the fields in the form, then in runtime, an exception message will be displayed informing you that a field is not found.

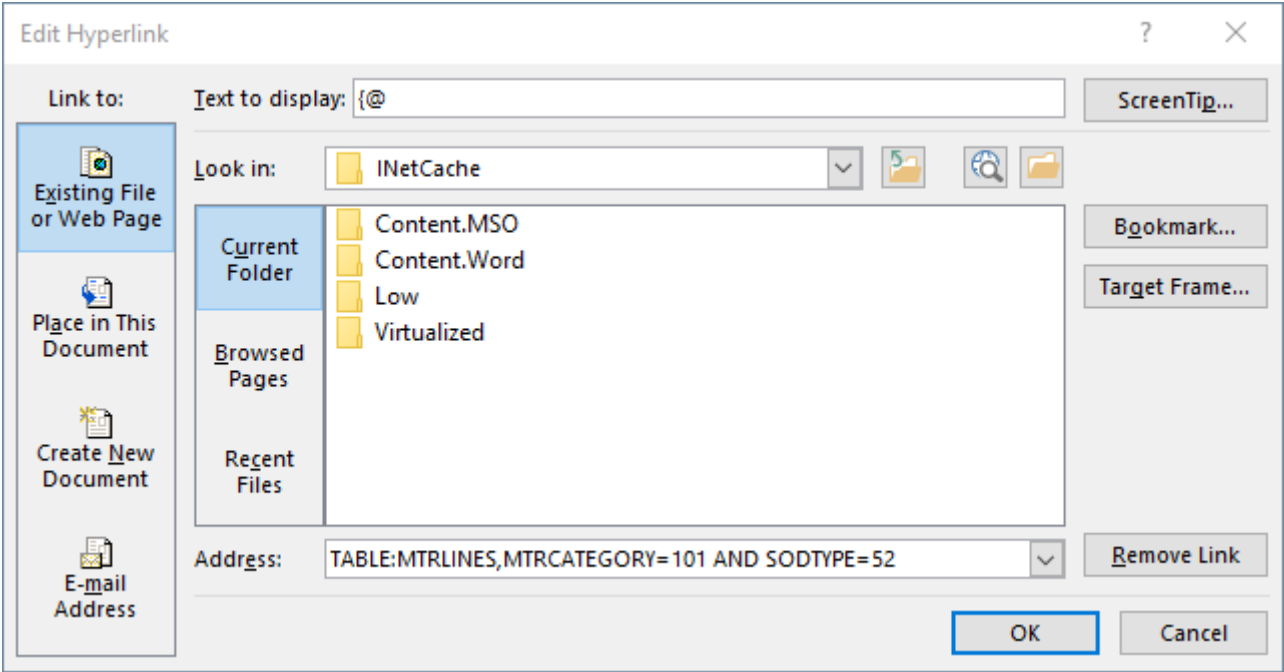


Figure C3.1

C.4 Field decimals

Decimal places can be added to numeric fields, through editing the field hyperlink and adding the following command in the address textbox (Figure C4.1):

FIELD:<Field>;n where n is the number of decimal places.

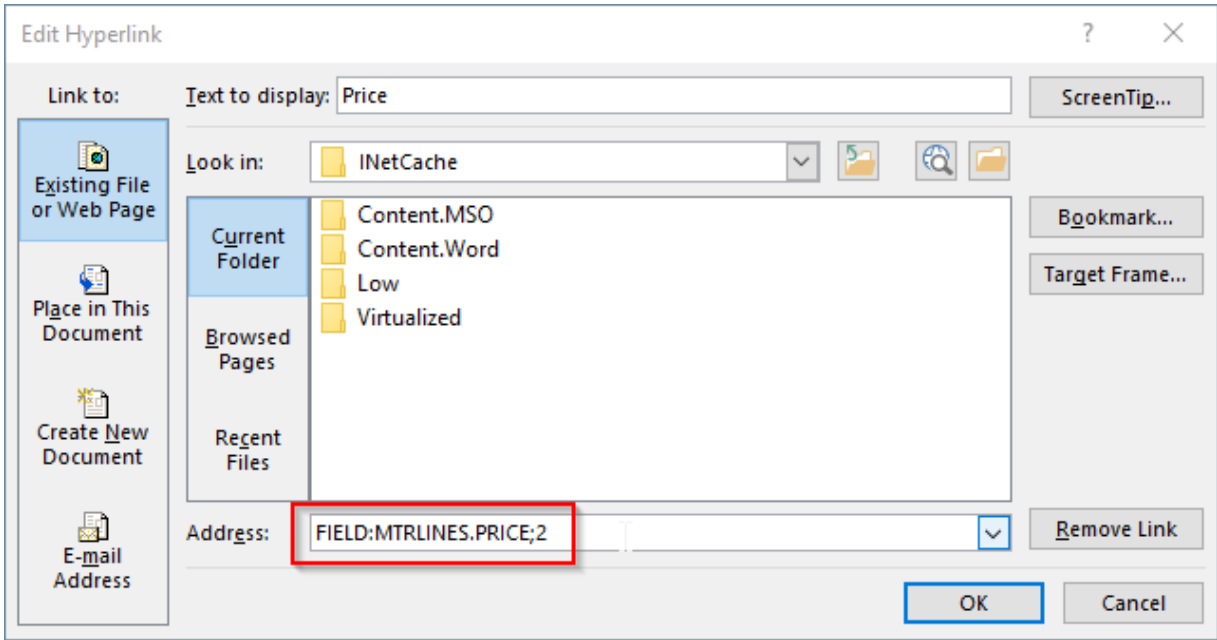


Figure C4.1

C.5 Report Footer

In order to add data in the report footer you need to use the following steps:

- Insert text boxes at the bottom of the page and enter the text you want to be displayed.
- Add a text box outside the above text boxes. (Figure C5.1)
- Group all text boxes.
- Select the grouped box and add a hyperlink, entering the command "**XTR:LP_FOOTER**" in Address textbox. (Figure C5.2)

NEW BALANCE:	PREVIOUS BALANCE:	TOTAL QTY
Balance	Previous customer balance	Total qty

NET VALUE	VAT%	VAT VALUE
Value subject	VAT	VAT value
Value subject	VAT	VAT value
Value subject	VAT	VAT value

VALUE BEFORE DISCOUNT	DISCOUNT	VALUE AFTER DISCOUNT	VAT
Val w/o disc-exp	Total discount	Net value	VAT Amnt
TOTAL VALUE			Total

Figure C5.1

Link to: Text to display: <<Selection in Document>>

Look in: INetCache

Current Folder: Content.MSO, Content.Word, Low, Virtualized

Browsed Pages:

Recent Files:

Address: **XTR:LP_FOOTER**

OK

Figure C5.2

C.6 Item Image

Image fields are added inside the lines of the MS-Word document, using the following steps:

- Insert a Rectangle Shape (Insert - Shape - Rectangle) (Figure C6.1).

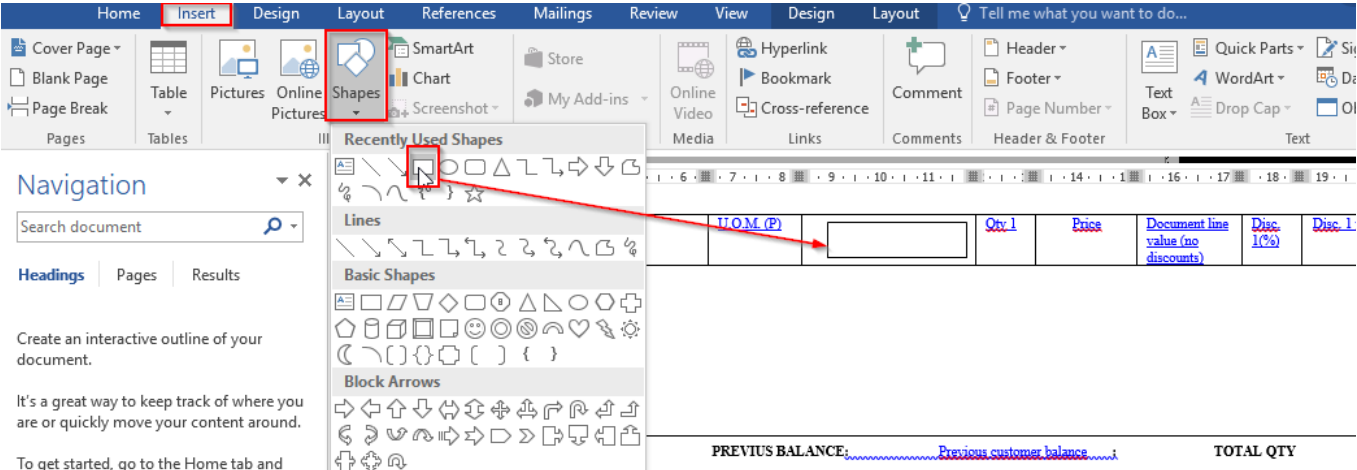


Figure C6.1

- Right click the shape and insert a hyperlink. Add the following command inside the Address box: **IMAGE:\$MTRLINES.MTRL;0** (Figure C6.2)

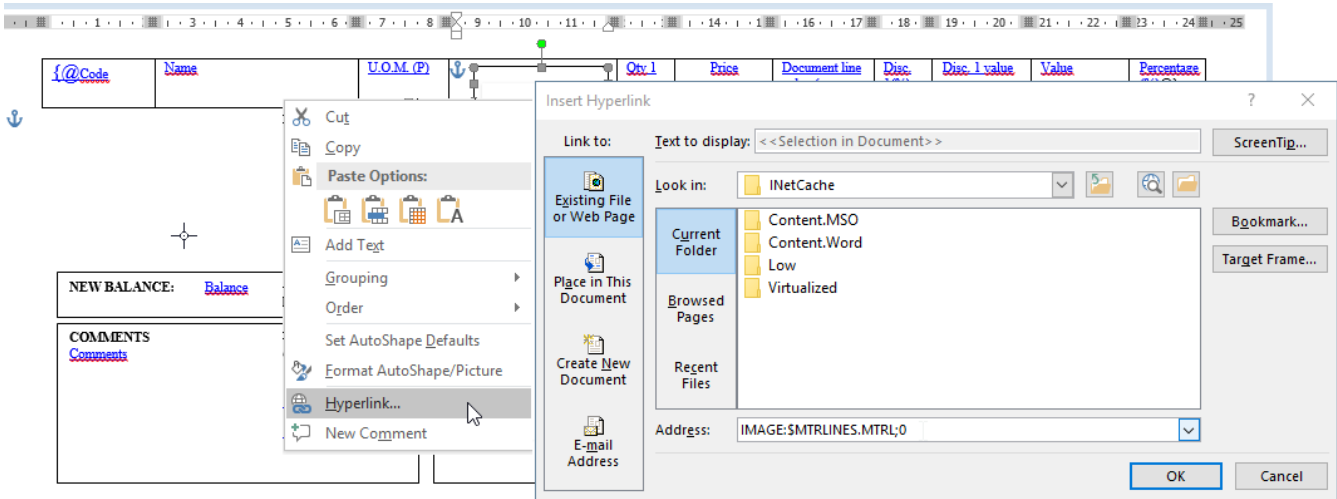


Figure C6.2

- Notice that the shape must be inside a column of the detail table. In order to achieve that, you can use the formatting marks to resize and move it (Figure C6.3).

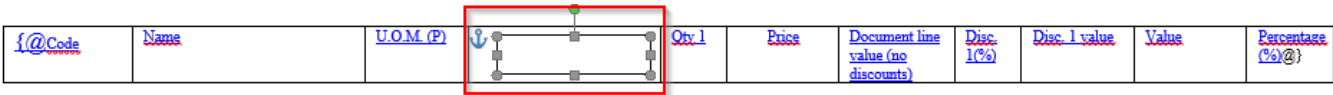


Figure C6.3

C.7 Text wrap

MS-Word forms that use multi-line memo fields (i.e. customer remarks) are displayed incorrectly (with line-break character symbols). Moreover, they perform a line break if the field is inserted directly into the main text, while they do not insert a line break if the field is inserted into a table cell (Figure C7.1)

Code	101
Name	John Economopoulos
VAT Number	073586841
Address	Distomo
City	Distomo Viotia
☐comments line 1 ☐comments line 2 ☐comments line 3 ☐comments line 4	
Remarks: comments line 1☐comments line 2☐comments line 3☐comments line 4	

Figure C7.1

This can be easily fixed by using a defined field which will replace the line break characters #13 & #10 with the character #11. The following example uses the field vRemarks instead of the field Remarks of Sales documents. **REPLACE(REPLACE(A.REMARKS,char(13),''),char(10),char(11))** (Figure C7.2)

The result of using the user-defined field on the form is shown in Figure C7.3.

Local Fields. Table : Sales documents

Available functions

Fields

Name	Caption	Operation	Field Type	Calculation Type	Display
vRemarks	RemarksWrap	SQL command	Alphanumeric	REPLACE(REPLACE(A.REMARKS,char(13),''),cha	<input checked="" type="checkbox"/>

1 of 1

Calculation formula (Detailed)

1 REPLACE REPLACE (A.REMARKS, char (13) , '') , char (10) , char (11)

1: 58

CAPS NUM SCRL INS

OK Cancel

Figure C7.2

Code	101
Name	John Economopoulos
VAT Number	073586841
Address	Distomo
City	Distomo Viotia
comments line 1 comments line 2 comments line 3 comments line 4	
Remarks: comments line 1 comments line 2 comments line 3 comments line 4	

Figure C7.3

D. Ms-Excel Printout Forms

D.1 General

MS Excel printout forms are designed through Microsoft Excel, using the option "External (Excel)" (Figure D1.1).

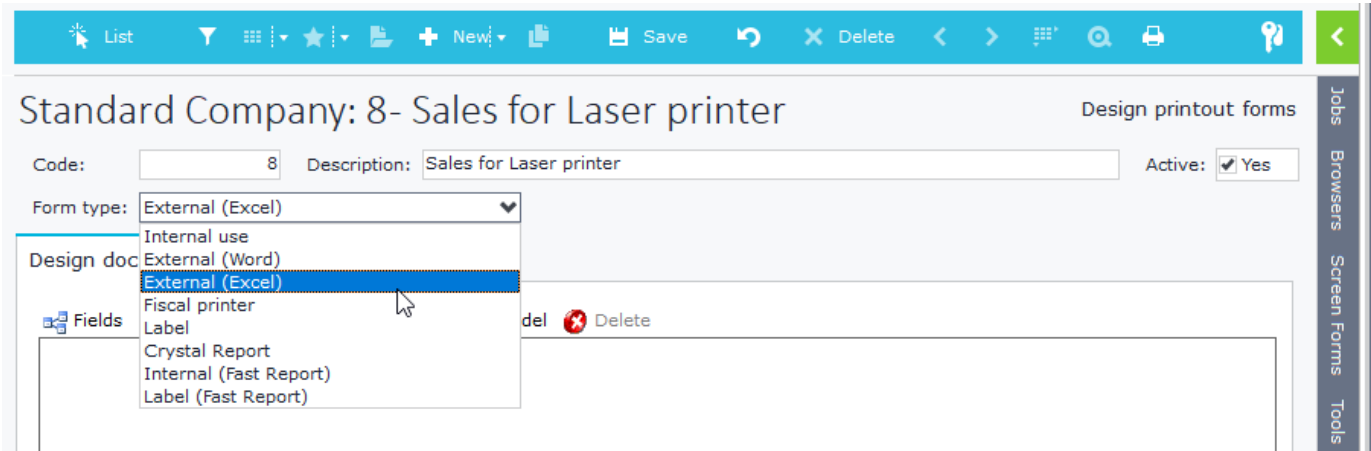


Figure D1.1

Through the toolbar that is displayed (Figure D1.2) you can create a new excel document, modify or delete the existing one and insert Fields from the current object to your MS-Excel form.

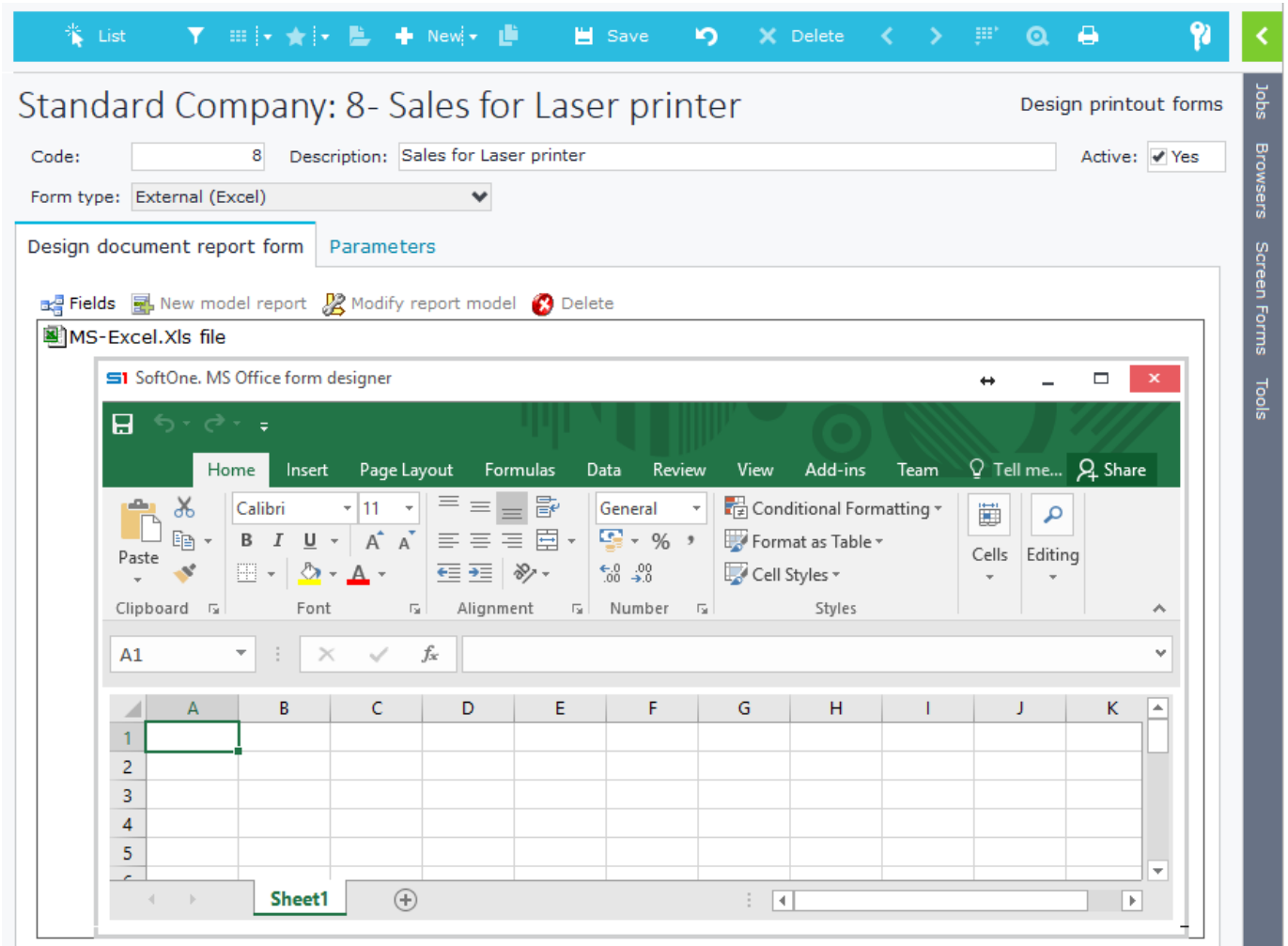


Figure D1.2

D.2 Form Design

Excel forms have a recurring section but do not support pagination. Fields are inserted through **double click**. Select a cell inside Excel and then double click on a field from SoftOne "Fields" window.

Recurrent section must have as first column the {> symbol signs and as last column the symbols <}. To set the SoftOne table from which to retrieve data of the recurrent section, select the {> symbol in the first column and create a hyperlink entering in the address box the command "**Table: table name**" (Figure D2.1). You do not need to insert a hyperlink in the <} symbol of the last column cell.

Note: Fields in recurrent section must be entered after you insert the {> symbol of the first column in order for the form to run properly.

The syntax for filtering, sorting or grouping data is the same as in MS-Word printout forms:

TABLE:<Table>, <FilterField 1> AND<FilterField 2> OR<FilterField 3>, <SortField>, <GroupField>

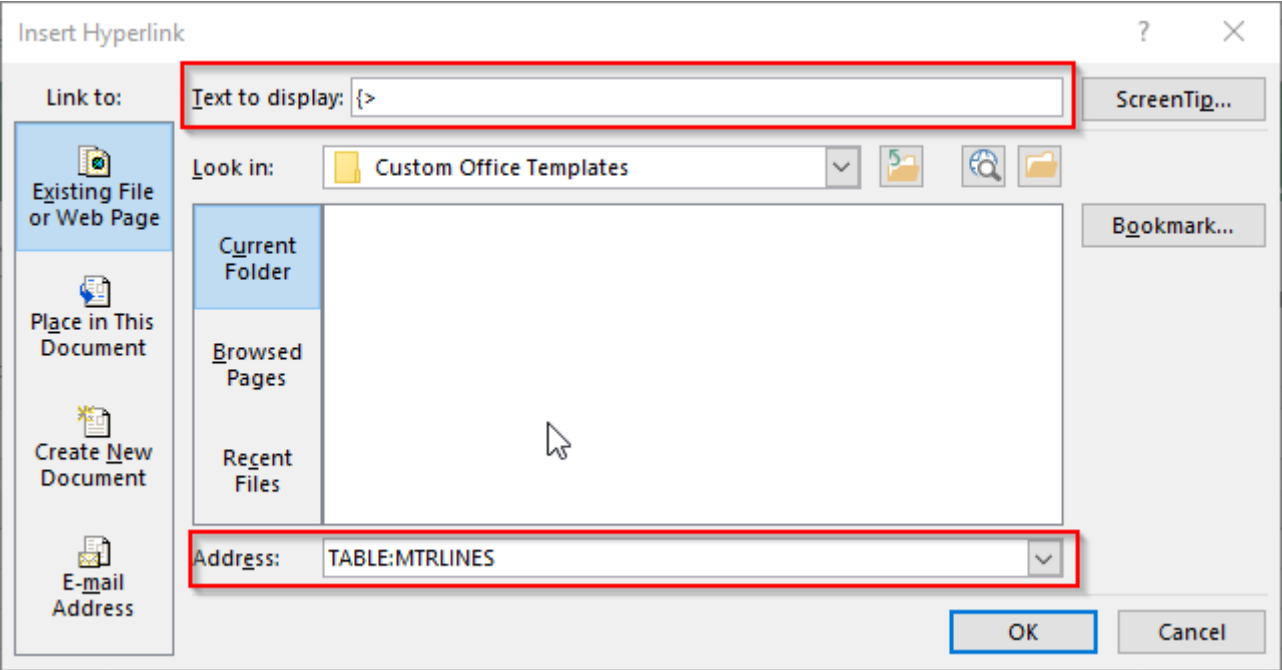


Figure D2.1

E. Label Printout Forms

Labels are designed through the option “**Label**”. The tool for designing labels is the same as the one used for internal printout forms (Figure E1).

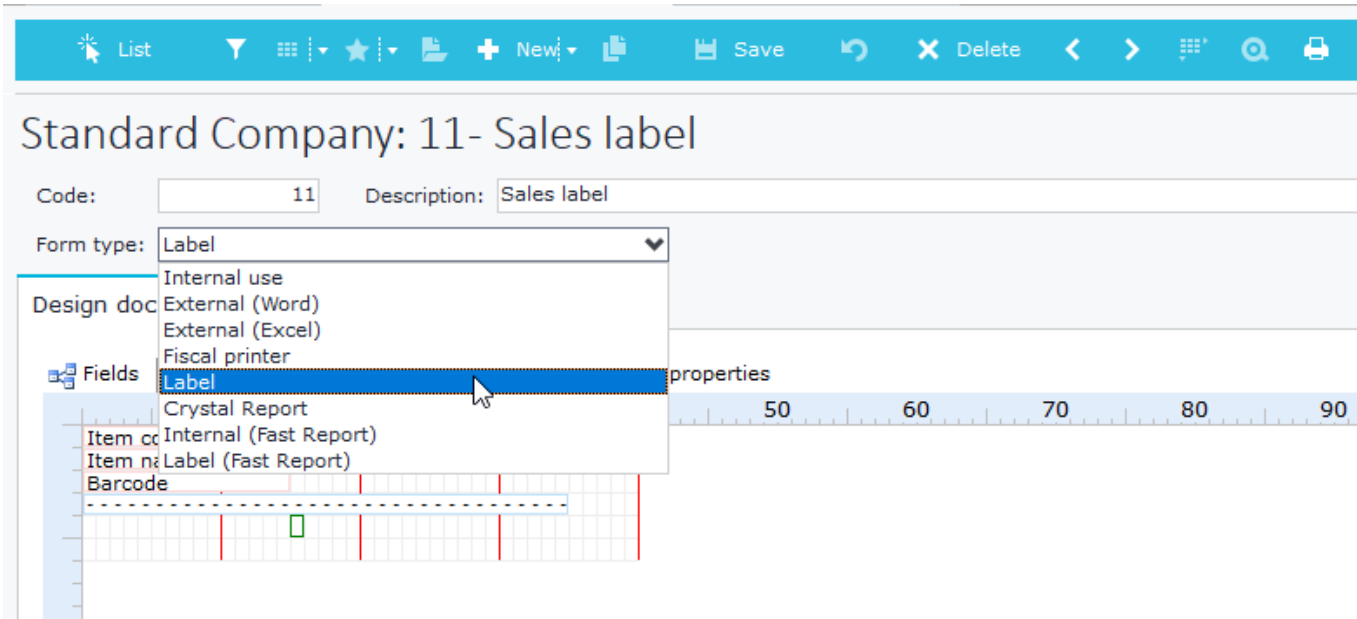


Figure E1

The page properties button allows you to change the page data properties as shown in Figure E2.

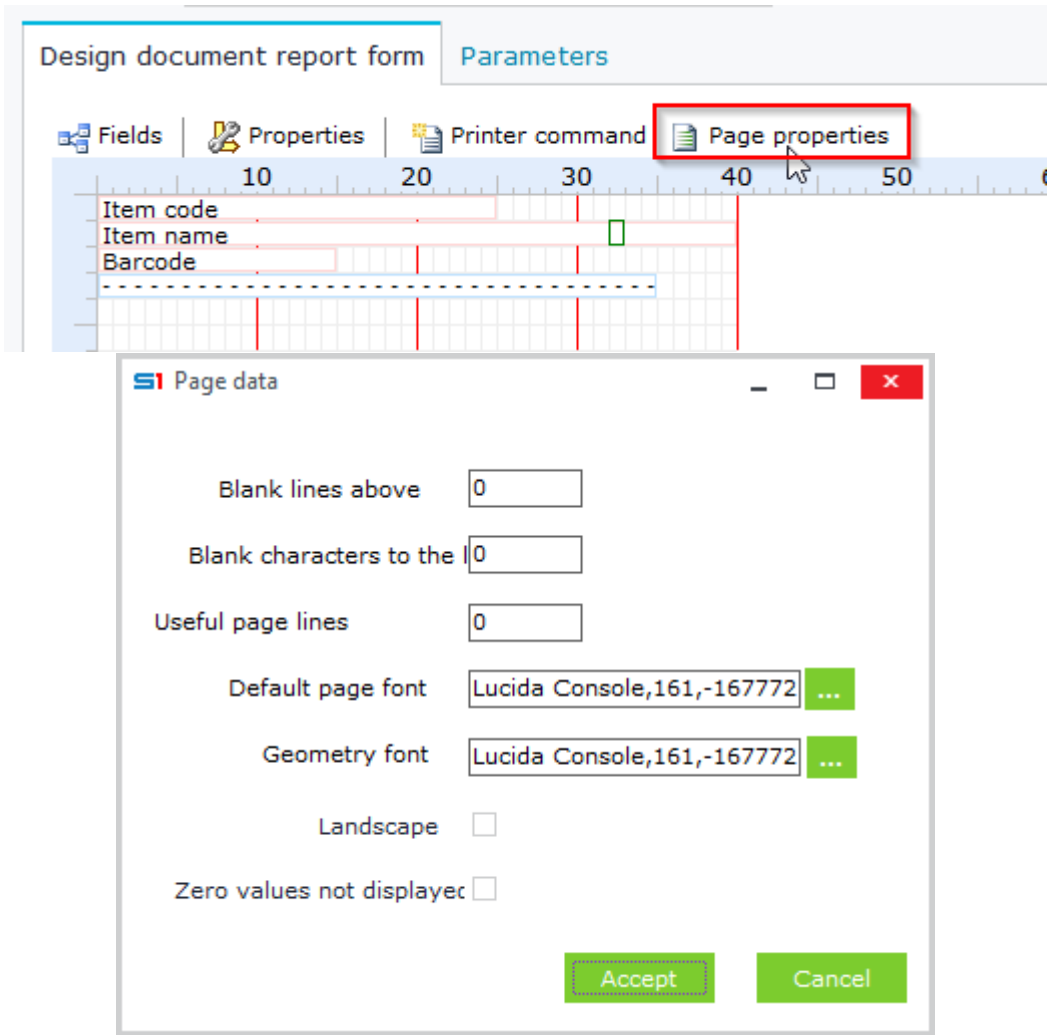


Figure E2

Through form properties you can define the label size, as well as the table that will be used for retrieving data. From the same window you can set the number of copies that a label will be printed (Figure E3).

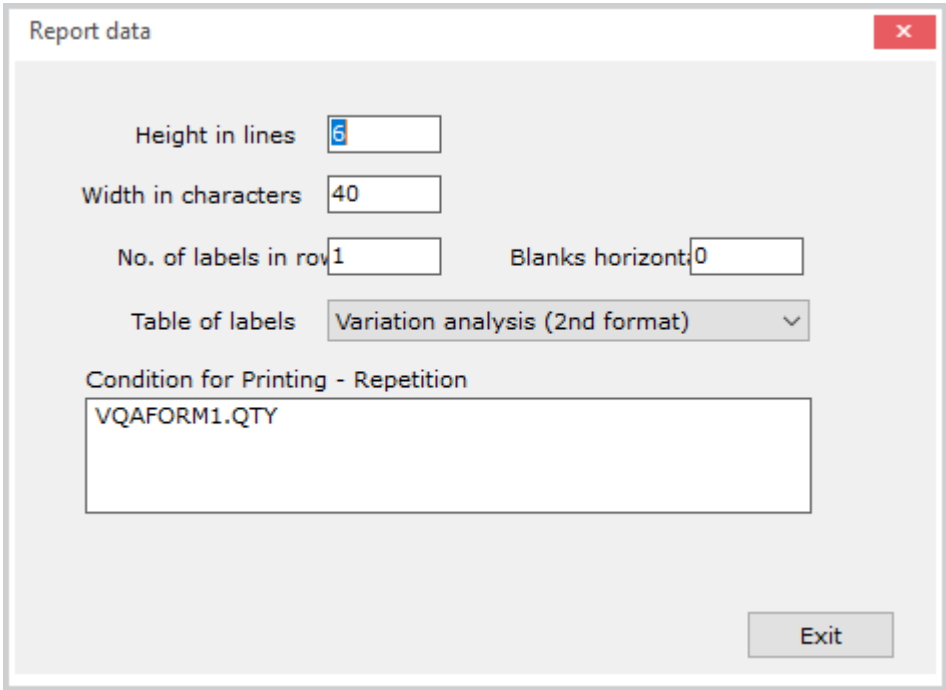
A dialog box titled "Report data" with a close button (X) in the top right corner. It contains several input fields and a dropdown menu. The fields are: "Height in lines" with a value of 6, "Width in characters" with a value of 40, "No. of labels in row" with a value of 1, and "Blanks horizontal" with a value of 0. There is a dropdown menu for "Table of labels" currently showing "Variation analysis (2nd format)". Below these is a text area labeled "Condition for Printing - Repetition" containing the text "VQAFORM1.QTY". At the bottom right is an "Exit" button.

Figure E3

Fields are added in the label in the same way as in internal forms (drag and drop or double click). Barcodes can be displayed by choosing the appropriate font.

In runtime, labels are printed through the right click option **"Print labels"**. Before printing labels, the window of figure E4 is displayed, allowing you to change the printing settings. Enable the option **"Automatically from form"** for activating the commands you have entered in the **"Printing – repetition condition"** property of the label.

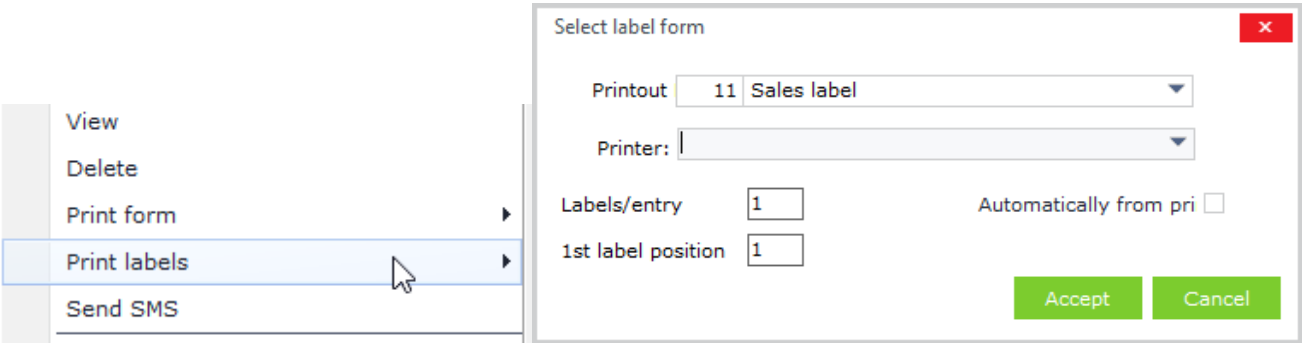
A screenshot showing a context menu on the left with options: View, Delete, Print form, Print labels (highlighted with a mouse cursor), and Send SMS. To the right is a dialog box titled "Select label form" with a close button (X) in the top right corner. It contains: a "Printout" dropdown menu showing "11 Sales label", a "Printer:" dropdown menu, "Labels/entry" input field with value 1, "1st label position" input field with value 1, and a checkbox labeled "Automatically from pri" which is unchecked. At the bottom right are "Accept" and "Cancel" buttons.

Figure E4

F. Crystal Reports Printout Forms

F.1 Basics

The forms in Crystal Reports are designed exclusively from within the Crystal Reports application. Printout forms designed using crystal reports can only be printed on a physical printer or in pdf file.

Crystal Reports forms can run from computers where Crystal Reports is not installed, as long as you have installed the Crystal Reports Runtime application, which is located in the SoftOne common area.

Create a new form, using the option "**Crystal Report**" in the form type. (Figure F1.1)

Next, create a new Crystal Reports printout form through the right click option or by clicking on the "New model report" button (Figure F1.2)

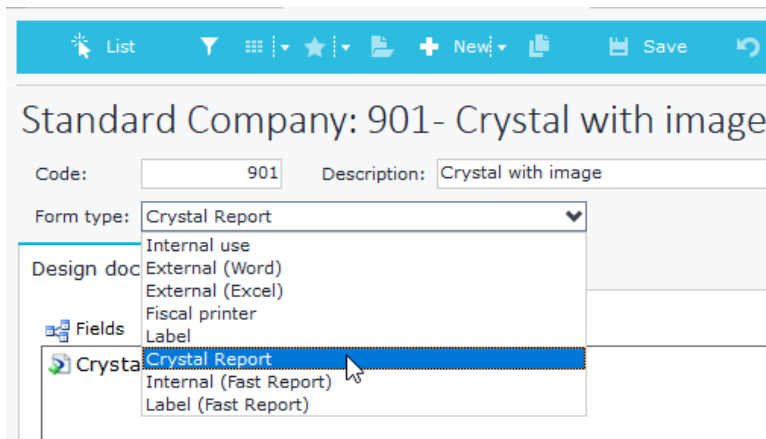


Figure F1.1

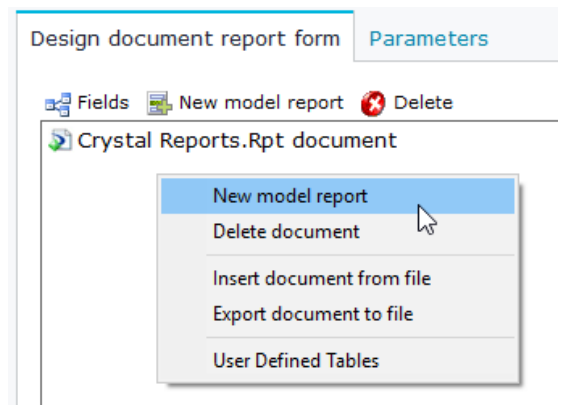


Figure F1.2

Upon creating report forms in Crystal Reports, you need to know the following:

- The database tables that contain the data you want to display.
- The field names of the above tables.
- The primary keys of the tables that will be used to link them.
- If the SoftOne installation is on-premises or on Azure.
To create a crystal report in a Windows Azure environment it is mandatory to use a specific command.
- The key fields of the object tables through which SoftOne will connect to the Crystal report.
These fields will be matched through SoftOne local fields.

Designing a form in Crystal Reports application can be done in two ways; either using tables or using commands. Commands use SQL queries to define the data and the structure (tables and fields) that will be used in the report.

F.2 Crystal Report Design

Create a new report from the File - New - Standard report menu and then from the window that appears choose OLE DB(ADO). In the window displayed choose "Microsoft OLE DB Provider for SQL Server", if the database is on an SQL Server (Figure F2.1). The next window (Figure F2.2) asks for the Server credentials and the database name that will be used in the report.

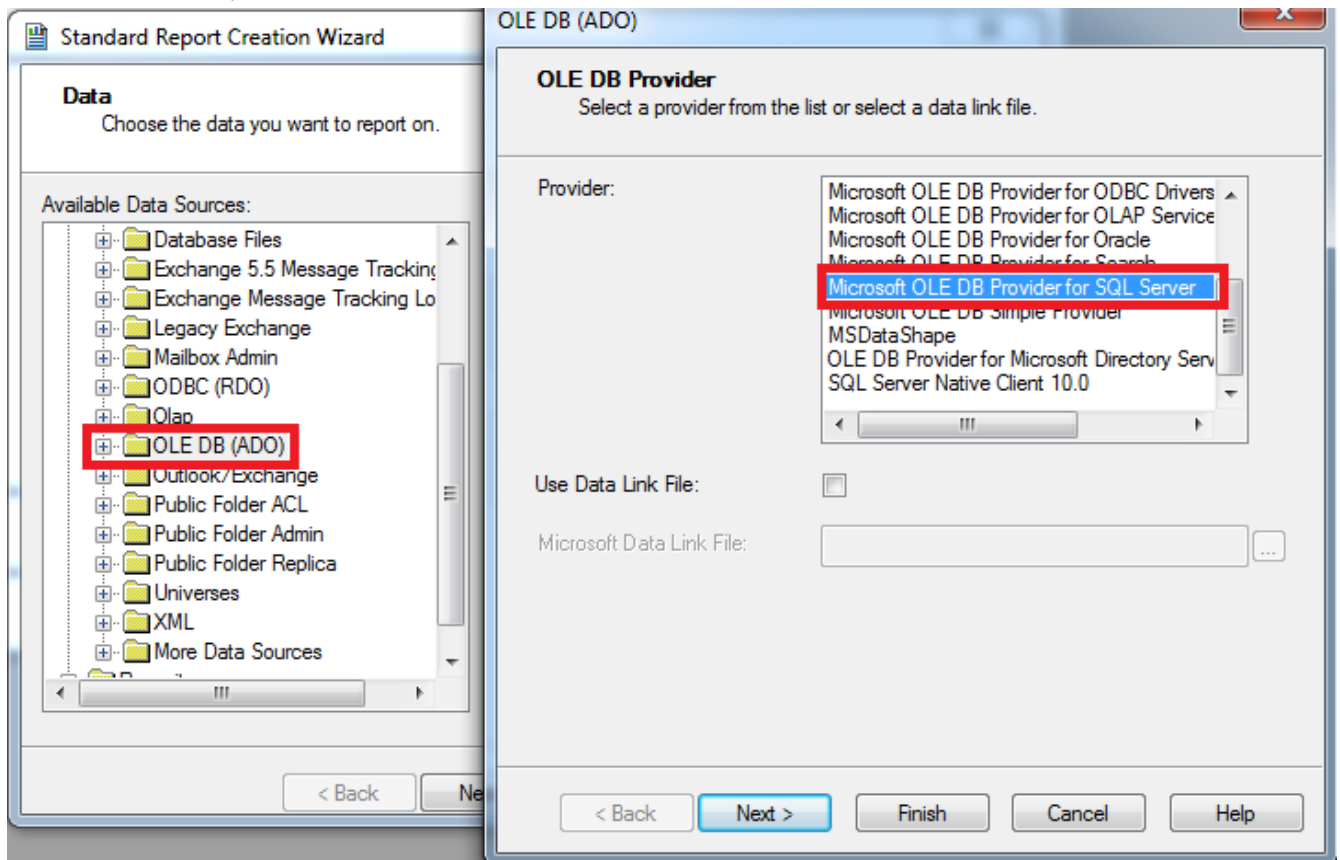


Figure F2.1

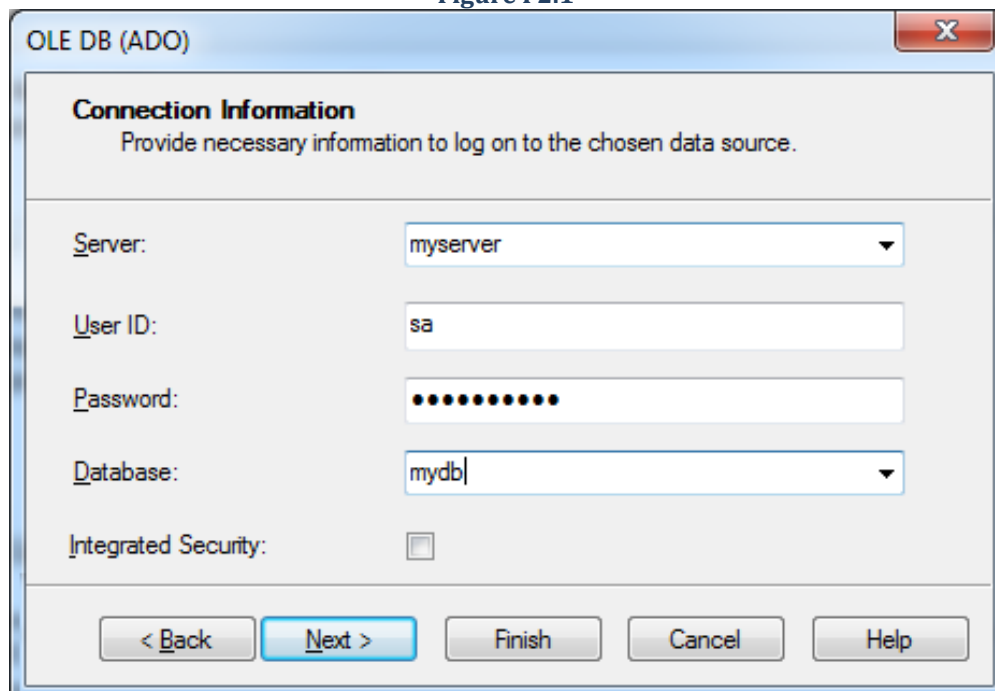


Figure F2.2

F.3 Reports using tables

For creating reports using tables, you need to add the tables that will be later used in the report from the tab “Data” (Figure F3.1). The next step is to link the tables (tab “Links”) based on their primary keys (Figure F3.2).

The example below uses tables for creating a crystal report that will be inserted in Sales documents printout forms.

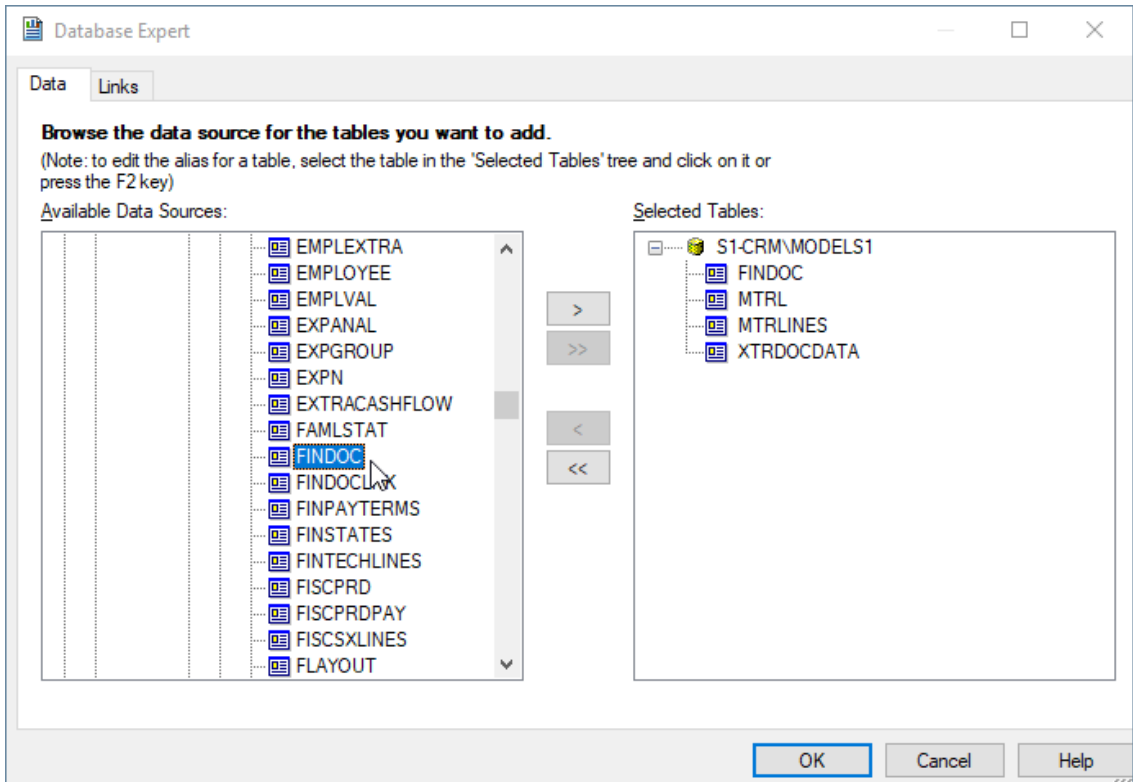


Figure F3.1

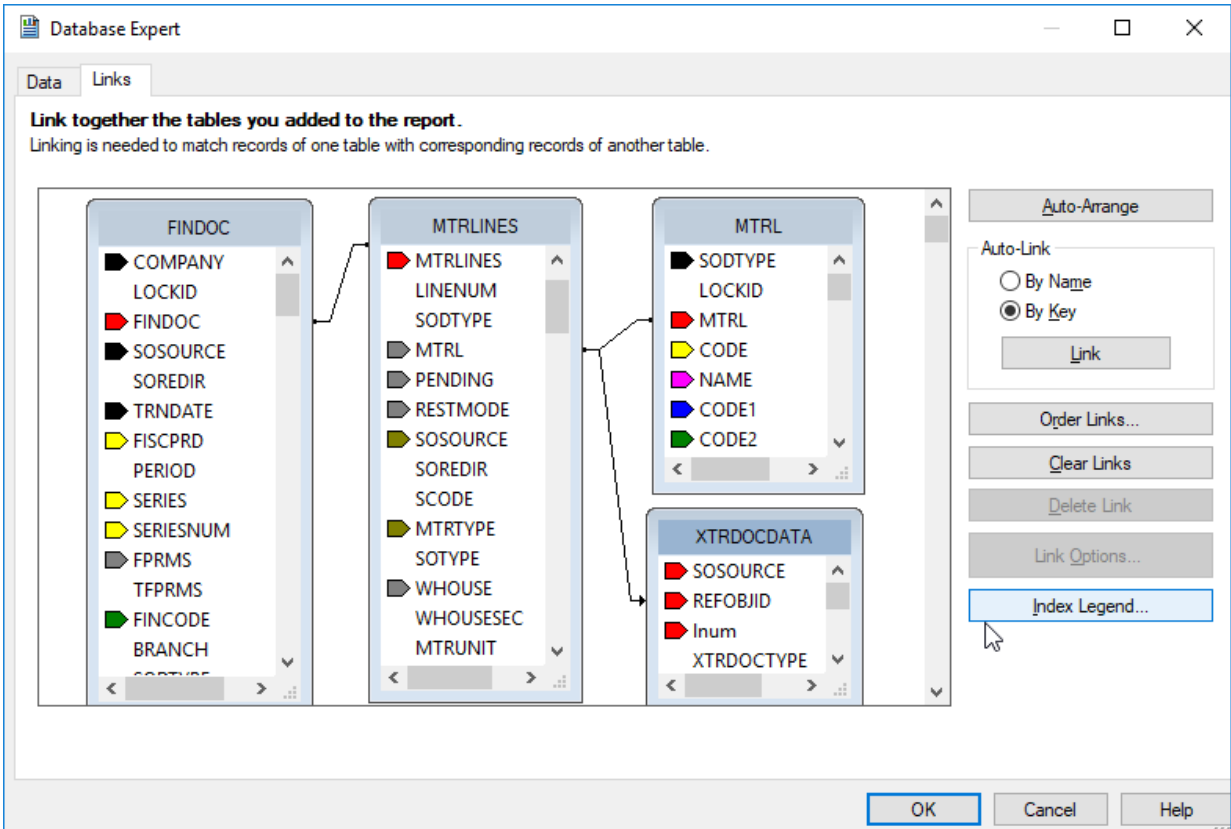


Figure F3.2

From the main report Design screen, design the report by selecting the fields you want display (Figure F3.3).

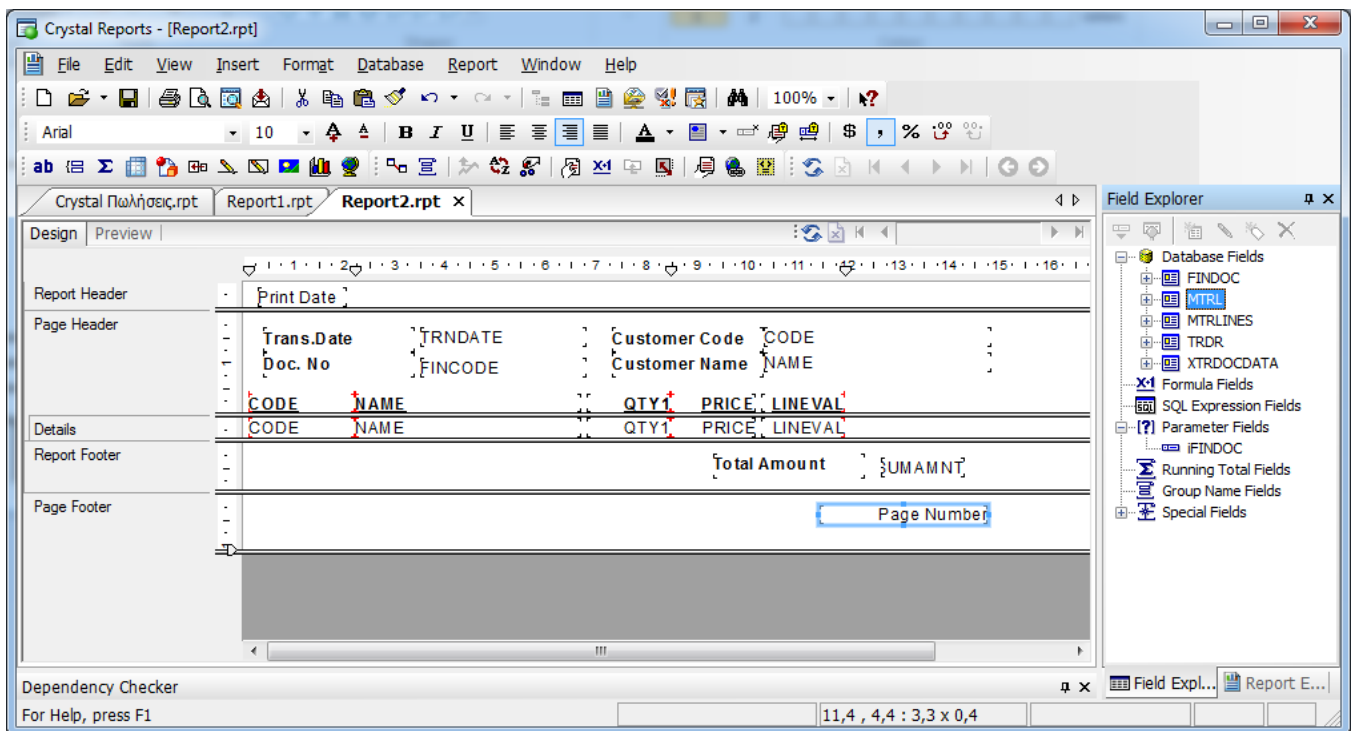


Figure F3.3

To link the report to a SoftOne entry (e.g. the FINDOC field) you must set a Parameter field which will be linked to the primary field of the SoftOne entry.

The example uses the parameter iFINDOC which links to the FINDOC field of documents (Figure F3.4).

When creating the Parameter field, you should always pay attention to the field type. In our example the link field FINDOC is an integer, so the parameter type must be Number.

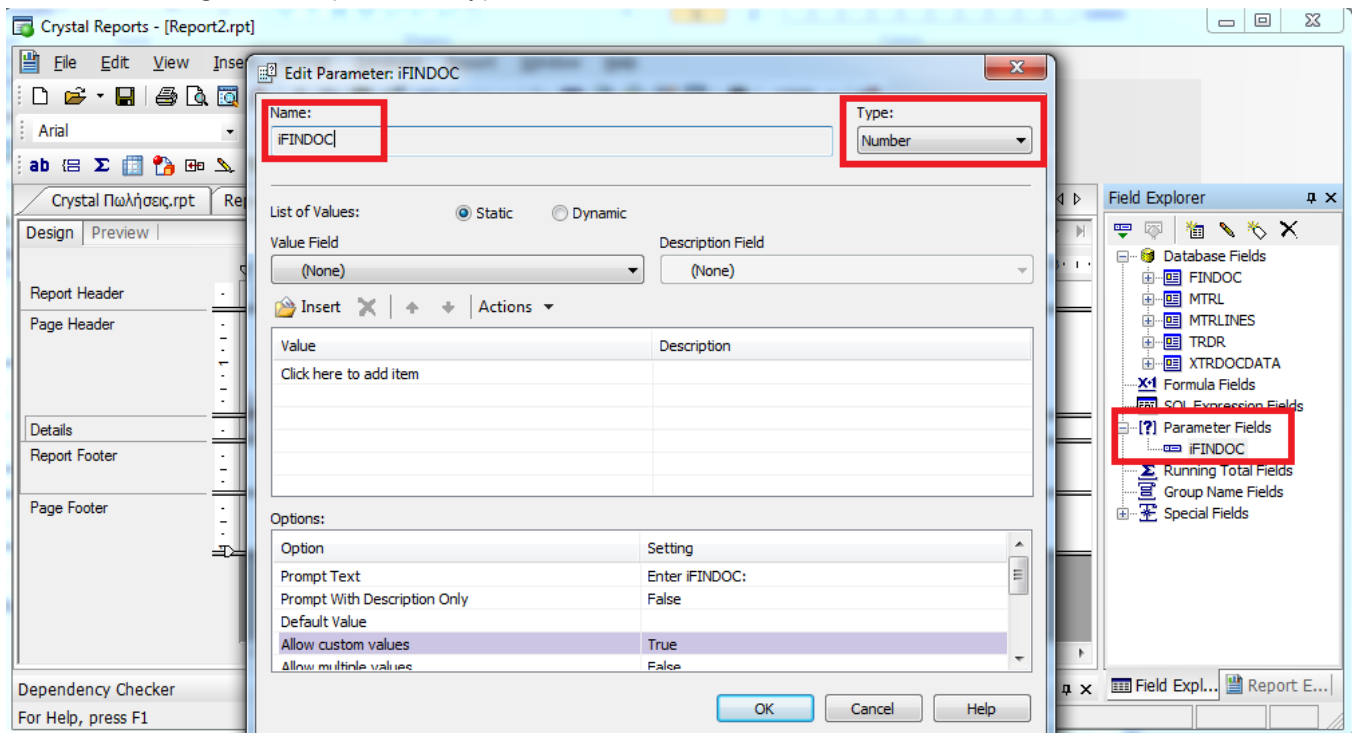


Figure F3.4

The next step is to create a filter for the report entries based on the Parameter field.

From the menu choose Report - Selection Formulas - Record and in the window that appears "link" the iFindoc parameter filter to the field FINDOC.FINDOC as shown in Figure F3.5.

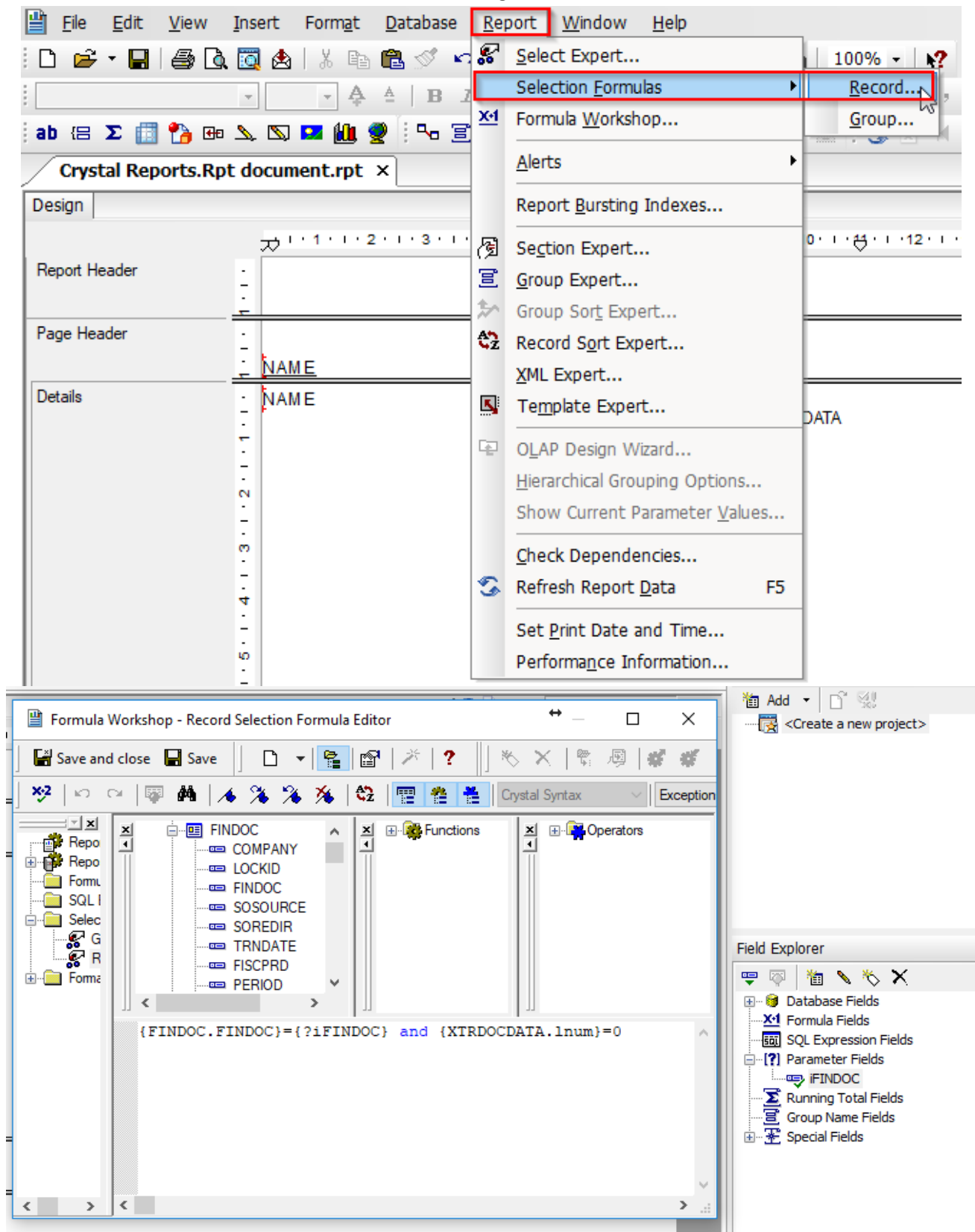


Figure F3.5

Instructions for important Crystal Reports to SoftOne can be found in Section F.5 .

F.4 Reports using SQL statements

Reports can be created using SQL statements. Notice that this way of creating Crystal reports **is mandatory for Azure installations**.

The first thing you need to do is to enter the SQL statement inside the “Add command” windows. The parameter fields are also defined here (Command Parameters) which will link your crystal report to a SoftOne record.

The example in Figure F4.1 displays the design of a crystal report through a SQL query command. This report can be used in Sales documents printout forms, since the parameter (iFINDOC) can be linked with a document record (FINDOC) through a defined field inside the printout form.

Design your report as in the previous section omitting the part of matching the Parameter Field with a table field, because it is already defined in the SQL command (Figure F4.2).

To import the report into SoftOne as a printout form, follow the steps of the next section.

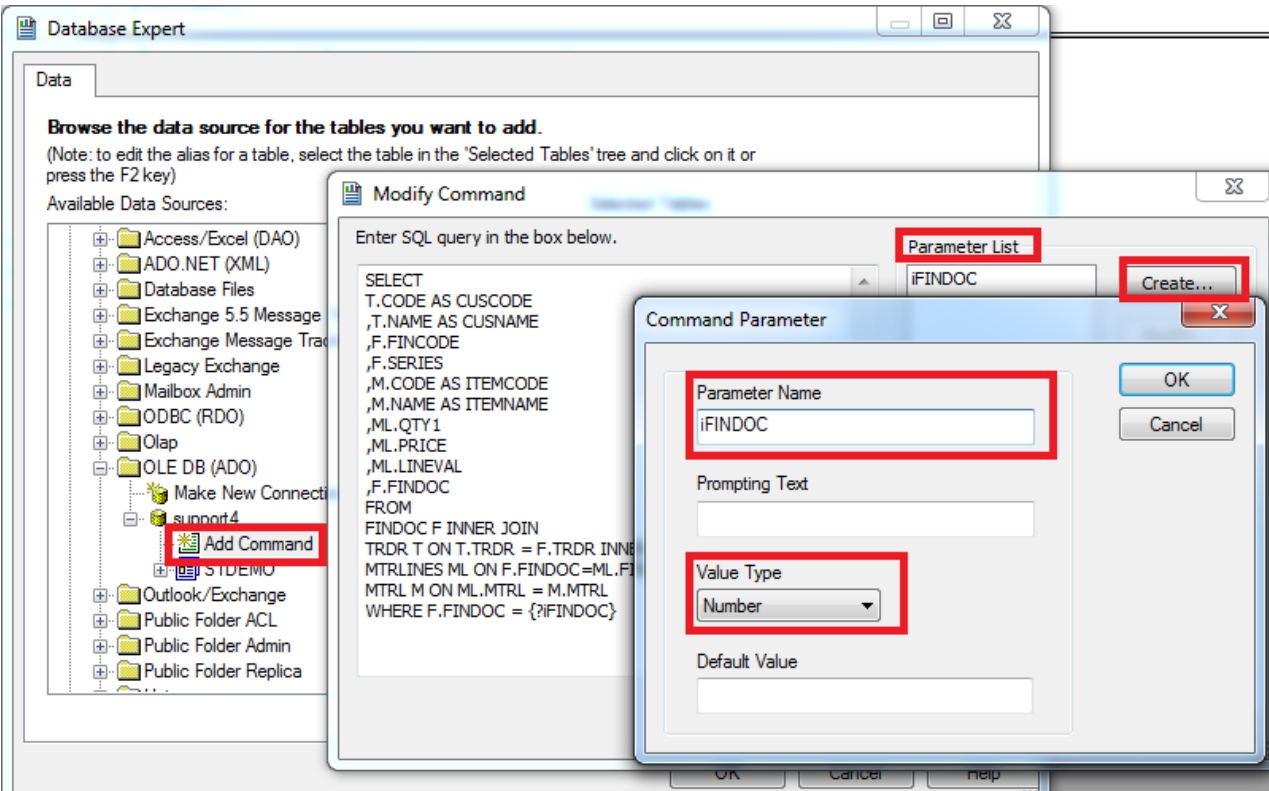


Figure F4.1

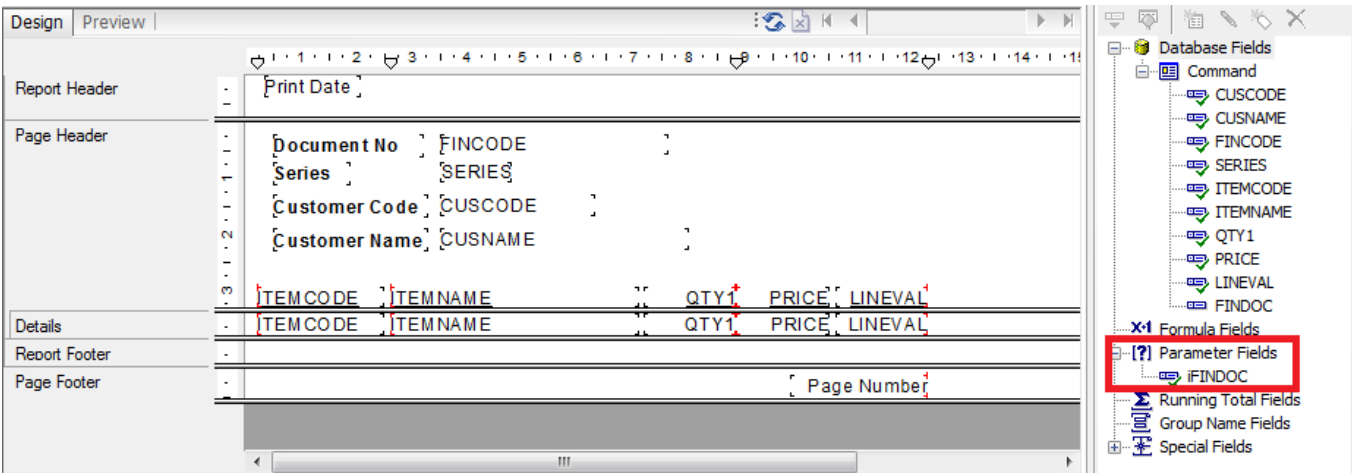


Figure F4.2

F.5 Import Crystal Report into SoftOne

In order to import a crystal report inside SoftOne as a printout form, just follow the steps below.

Create a new report form, select "Crystal Reports" as form type and then right click to select "Insert document from file".

Select the rpt file you have created and the window shown in Figure F5.1 will be displayed. If you choose "Yes" then the Parameter fields will be automatically imported into the "User defined fields" section of the report form defined fields (Figure F5.2).

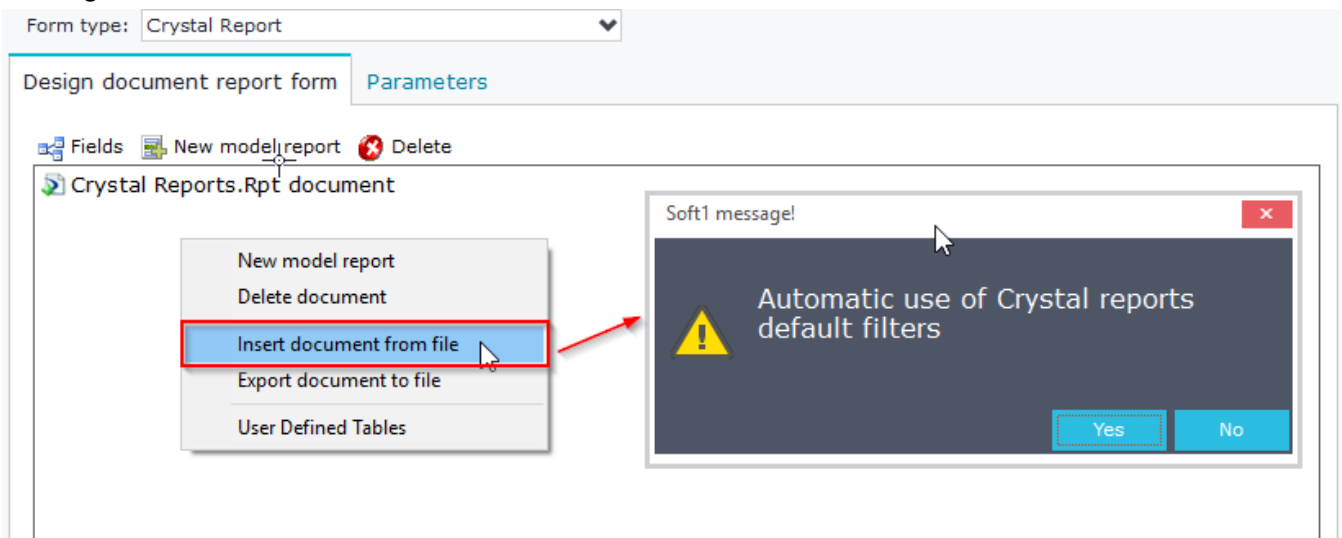


Figure F5.1

Notice that, "**Auto Insert Filters**" from Crystal parameters recreates the filters inside SoftOne user-defined fields. This means that if you insert a modified Crystal report, and do not want to lose the field matching (Crystal – SoftOne parameters) you should choose **NO** and not let SoftOne automatically insert filters, since otherwise the matches you have created will be deleted and then you will need to recreate them.

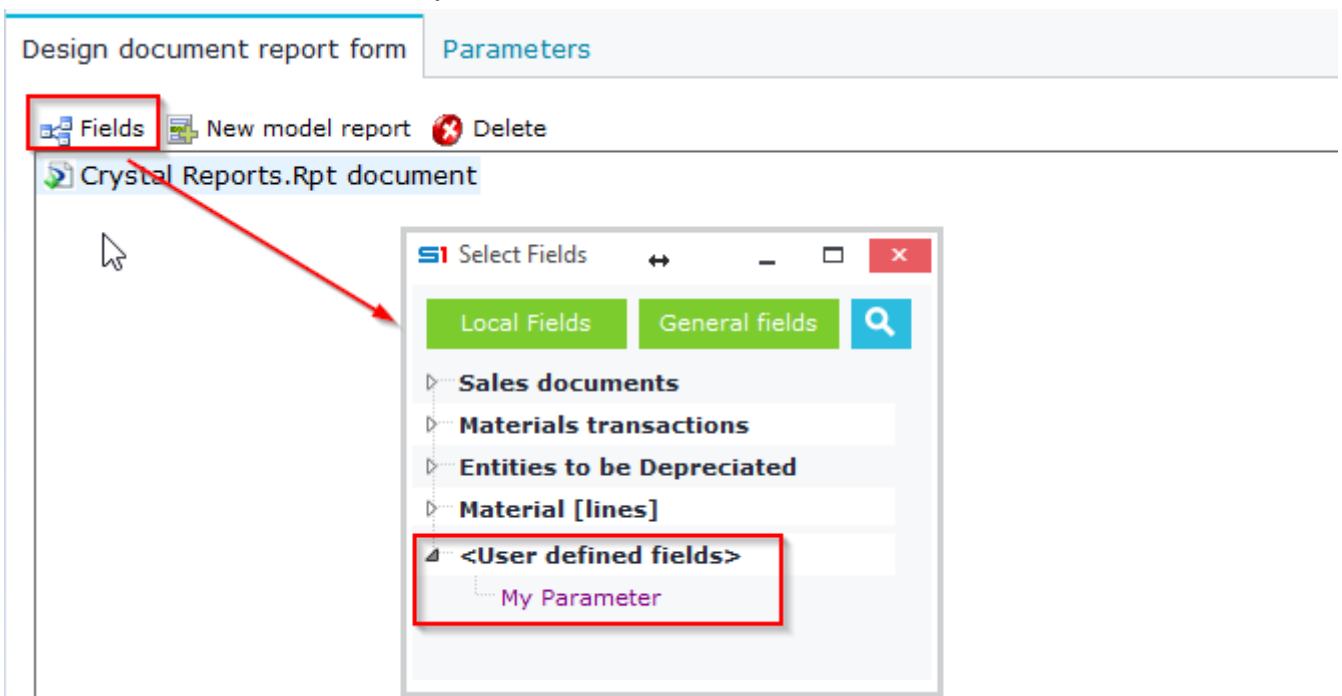


Figure F5.2

The following example demonstrates how to import a crystal report into sales documents:

- Click on “Fields” button of the printout form, select the «User defined fields” section.
- Insert a new user-defined field named Ifindoc, which is the name as the parameter name field of crystal reports.
- In the calculation formula enter the SoftOne matching field, which in our example is SALDOC.FINDOC (Figure F5.3).

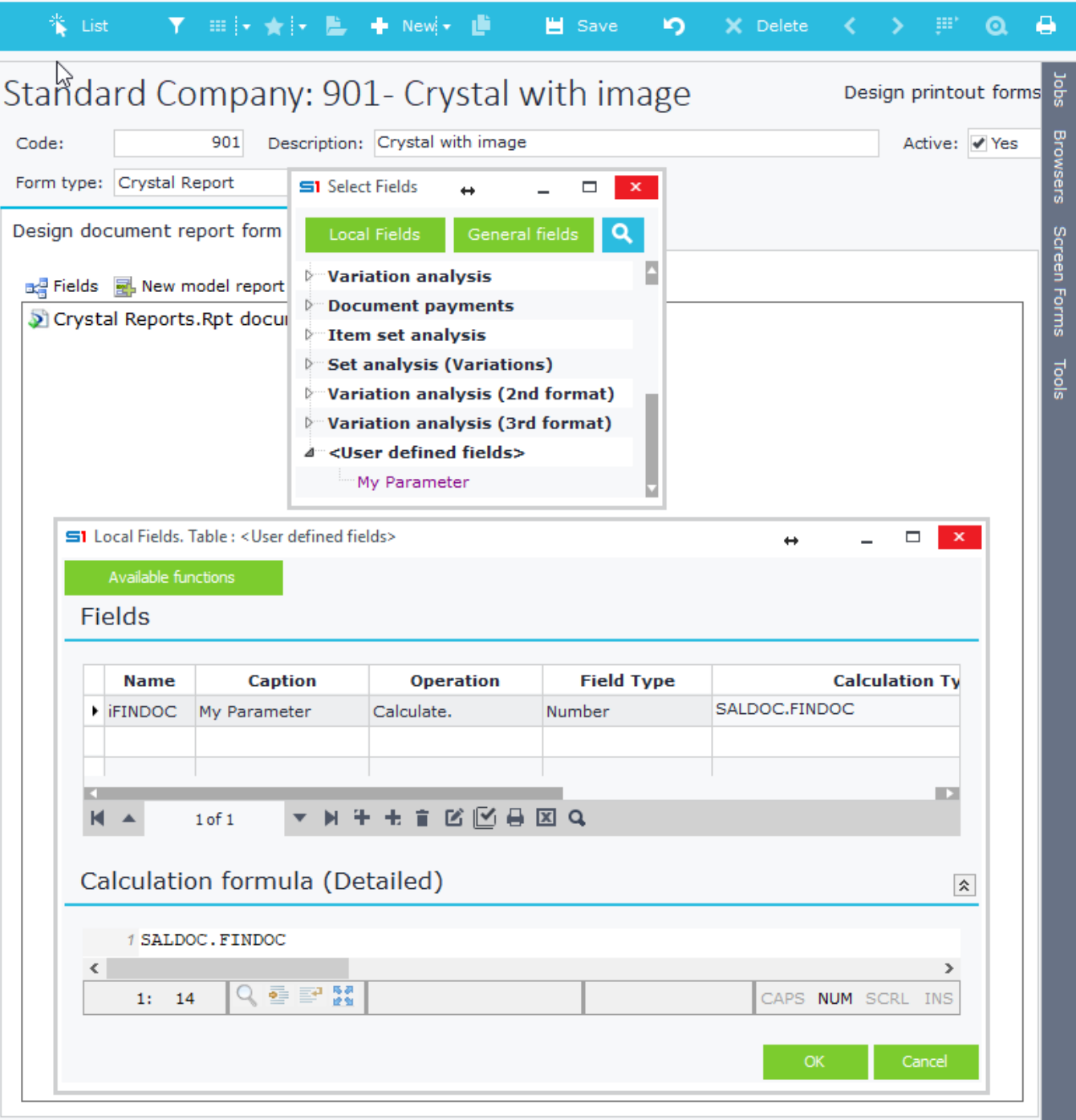


Figure F5.3

G. Automated Jobs

Printout forms can be used to automatically send reports via email in pdf format. There is also the option of auto saving the report in a file and path.

Note: If you have tax mechanism of B type, then the email files sent include the two txt files produced, that have the Special Secure Tax Layout of Data Marking.

G.1 Auto Email

Automatic routing (email sending) is performed by activating the corresponding option in the "Parameters" Tab (Figure G1.1). Using this option, the document will be sent directly in the chosen format, upon usage of the specific form.

The screenshot shows a software interface for configuring printout forms. The title bar indicates 'Standard Company: 2002- e-INVOICE'. The 'Parameters' tab is selected, showing various configuration options. The 'Layout' is set to 'PDF', 'Automatic routing' is 'Yes', and 'Automatic filing' is 'Yes'. The 'Fiscal signature device administration (type A)' is set to 'No'. The 'eMail' section is expanded, showing fields for 'Formula To', 'Formula CC', 'Formula BCC', 'Formula Subject', and 'Formula Body'. The 'Files' section is also expanded, showing fields for 'Formula for path' and 'Formula for file name'. The 'PDF layout codes' section is partially visible at the bottom.

Field	Value
Code	2002
Description	e-INVOICE
Form type	Internal use
Layout	PDF
Automatic routing	Yes
Automatic filing	Yes
Fiscal signature device administration (type A)	No
Formula To	SALDOC.TRDR_CUSTOMER_EMAIL
Formula CC	
Formula BCC	
Formula Subject	SALDOC.vSubject+' '+SALDOC.FINCODE+' from SoftOne'
Formula Body	SALDOC.vBody
Formula for path	'C:\DocFile\' + String(Login)
Formula for file name	SALDOC.FINCODE+' '+SALDOC.TRDR_CUSTOMER

Figure G1.1

The email parameters are the following:

- Formula To:** Email Recipient. In this box, enter the recipient's email in single quotes, or the email field from the corresponding table of the form.
 - Formula CC:** Notification. Completed similarly to "Formula To"
 - Formula BCC:** Bcc. Completed similarly to "Formula To"
 - Formula Subject:** Email subject. Enter in single quotations the text of the subject.
 - Formula Body:** Email text. Enter in single quotations the main text of the email. You can also use html code, which must be entered in a defined field (Figure G1.2).
- Note:** You can add any form fields to all the above-mentioned boxes.

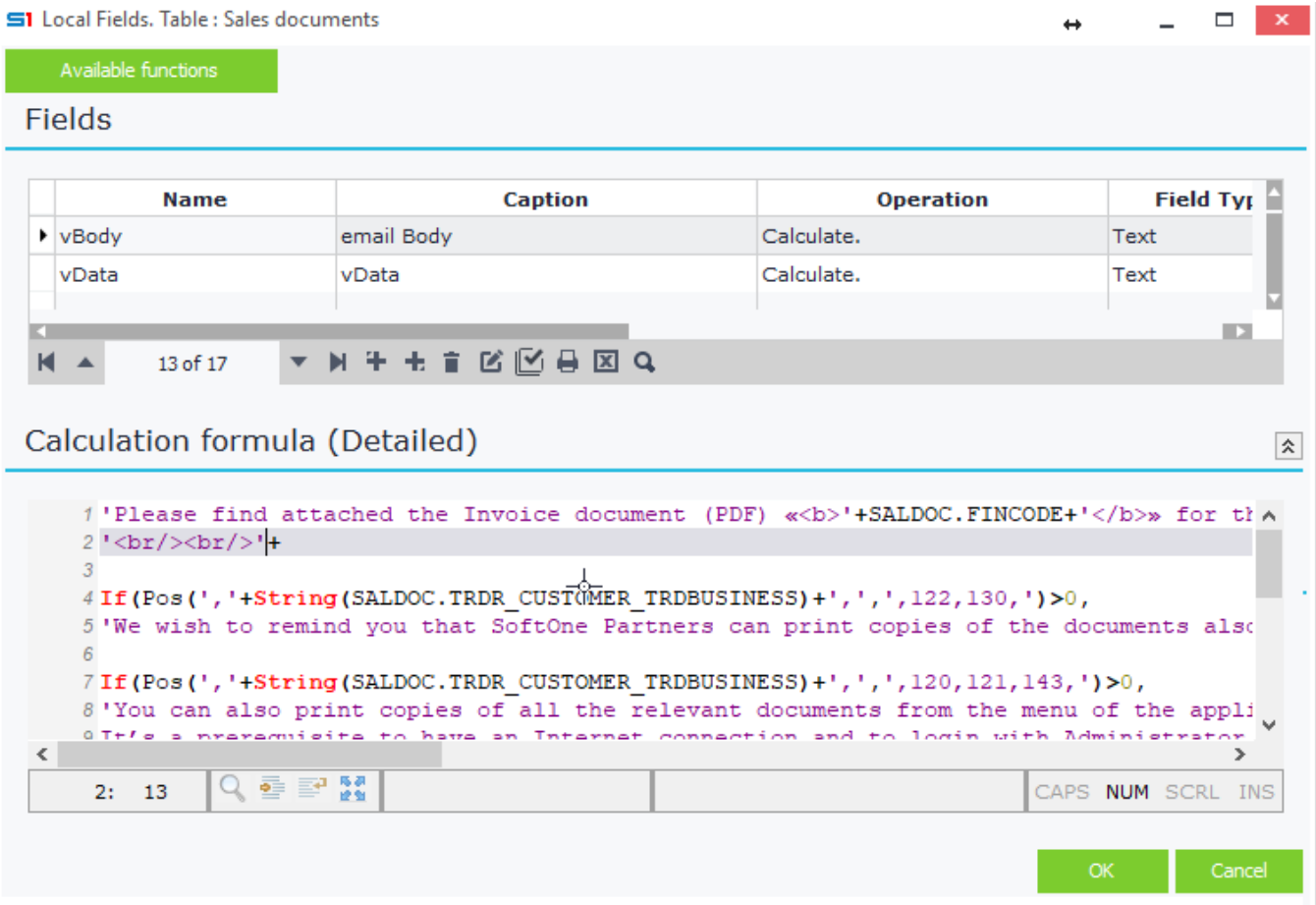


Figure G1.2

Figure G1.3 provides a simple example that uses text and fields inside the email properties. At runtime, after printing the specific sales document form, the (vBody) text will be sent to the email address of the customer (in Sales) (**SALDOC.TRDR_CUSTOMER_EMAIL**).

The subject of the email is a combination of text and fields **SALDOC.vSubject**, **SALDOC.FINCODE**.

The main text of the email is the one entered inside the user-defined field **SALDOC.vBody**.

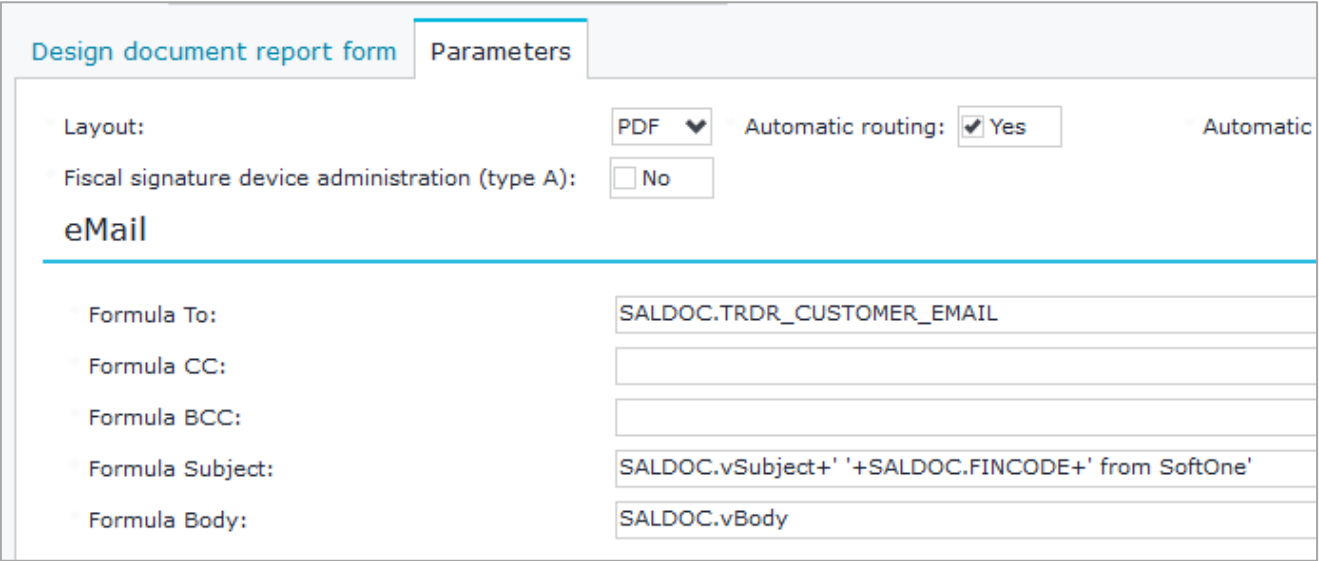


Figure G1.3

G.2 Auto Save to file

Automatic filing (file saving) is performed by activating the corresponding option in the "Parameters" Tab. This option can be used either individually or in combination with automatic routing.

In the "Files" panel fill in the "Formula for path" and "Formula for file" textboxes.

Formula for path: File path. Use single quotes to specify the path where the file will be saved. The path can be parametric, i.e. can include form fields (Figure G2.1).

Formula for file name: Filename. Use single quotes to specify the file name. You must necessarily indicate the extension of the file.

Note: If you encounter problems when printing multiple printout forms, remove the last backslash from the "Formula for path" textbox and place it at the beginning of "Formula for file name" ('\'+filename).

The screenshot shows a software interface with a 'Parameters' tab. The 'Layout' section has a 'PDF' dropdown, 'Automatic routing' checked 'Yes', and 'Automatic filing' checked. The 'Fiscal signature device administration (type A)' is set to 'No'. Below this is the 'eMail' section with fields for 'Formula To', 'Formula CC', 'Formula BCC', 'Formula Subject', and 'Formula Body'. The 'Files' section is highlighted with a red box and contains two fields: 'Formula for path' with the value 'C:\DocFile\' + String(LoginYear) + \'\' and 'Formula for file name' with the value 'SALDOC.FINCODE+', '+SALDOC.TRDR_CUSTOMER_CODE+'.PDF'.

Design document report form Parameters	
Layout:	PDF
Automatic routing:	<input checked="" type="checkbox"/> Yes
Automatic filing:	<input checked="" type="checkbox"/>
Fiscal signature device administration (type A):	<input type="checkbox"/> No
eMail	
Formula To:	SALDOC.TRDR_CUSTOMER_EMAIL
Formula CC:	
Formula BCC:	
Formula Subject:	SALDOC.vSubject+' '+SALDOC.FINCODE+' from SoftOne'
Formula Body:	SALDOC.vBody
Files	
Formula for path:	'C:\DocFile\' +String(LoginYear)+'\'
Formula for file name:	SALDOC.FINCODE+', '+SALDOC.TRDR_CUSTOMER_CODE+'.PDF'

Figure G2.1

4. ALERTS – EDA

EVENT DRIVEN ACTIONS

A. [Events & Conditions](#)

B. [Actions](#)

C. [Case Studies](#)

Overview

Event Driven Actions (Alerts) is a very powerful tool that allows you to add new functionality to any built-in or custom object in a way that it meets your needs. All the events, built in commands and internal functions are available through a very friendly interface that even amateur developers can use.

Jobs, like custom messages, field and record validation, SMS and email sending, field value calculations and many others can be easily developed with simple clicks or drag and drop actions.

More experienced developers can also find this tool very useful, because apart from the above, it fully supports SQL statements and coding in JavaScript or VBScript.

One thing that you must always keep in mind is that EDA affects by default all the forms of an object.

Examples of EDA are the following: display a message when text is changed in a field, calculate and set the data of a table field when a new row is inserted in a datagrid or even send email and SMS when a record is posted and certain conditions are met.

Alerts are saved as blob data inside the table CSTINFO (CSTTYPE=6) and can be easily transferred through different installations as .cst or .auv files using the "[Custom Administration](#)" tool.

A. Events & Conditions

EDA tool allows you to set the events and the conditions where actions will take place. When such conditions are met, a series of actions can be executed. In order to use events, you need to know the basic structural components (tables, virtual tables, etc.) that compose the objects of the application, e.g. object Customers. More specifically, different events occur in fields, datagrids and objects and can be used to create new rules and jobs. Multiple rules can be defined in a single EDA.

A.1 Field Rule (On Change)

This rule is used if you need to perform an action (display message, change field value etc.), when users change the data of a field. The following table shows the available rule properties:

PROPERTY	DESCRIPTION
Type	Type of Event (Field / Table / Module / Browser)
Description	Name of the Rule
Formula	Condition Formula that defines if the rule will fire and the actions will execute.
Field	Field that triggers the rule
Formula Action	Type of action that will be executed Send Message – Message display – Error display – Run – Reminder
Formula Type	Formula Type that depends on the formula action.
Formula Recipient	Recipient (used in message and email actions)
Formula	Calculations that will be performed

The following example (Figure A1) displays an alert created in 'Items' entity, that changes the value of the field ITEM.PRICEW (Wholesale price) when user changes the data of the field ITEM.PRICER (Retail price).

The operation is performed only when 'Retail price' (ITEM.PRICER) is greater than zero and 'Accounting category' (ITEM.MTRACN) is equal to 100.

Note: **Formula conditions** with more than one argument, that are separated with operators AND & OR should always be placed in **parentheses** (ITEM.PRICER>0) AND (ITEM.MTRACN=100).

The function that changes the data of the field is 'SETVALUE' and its syntax can be found inside the window Fields – <Event fields> – Local Fields – Available functions (Figure A2).

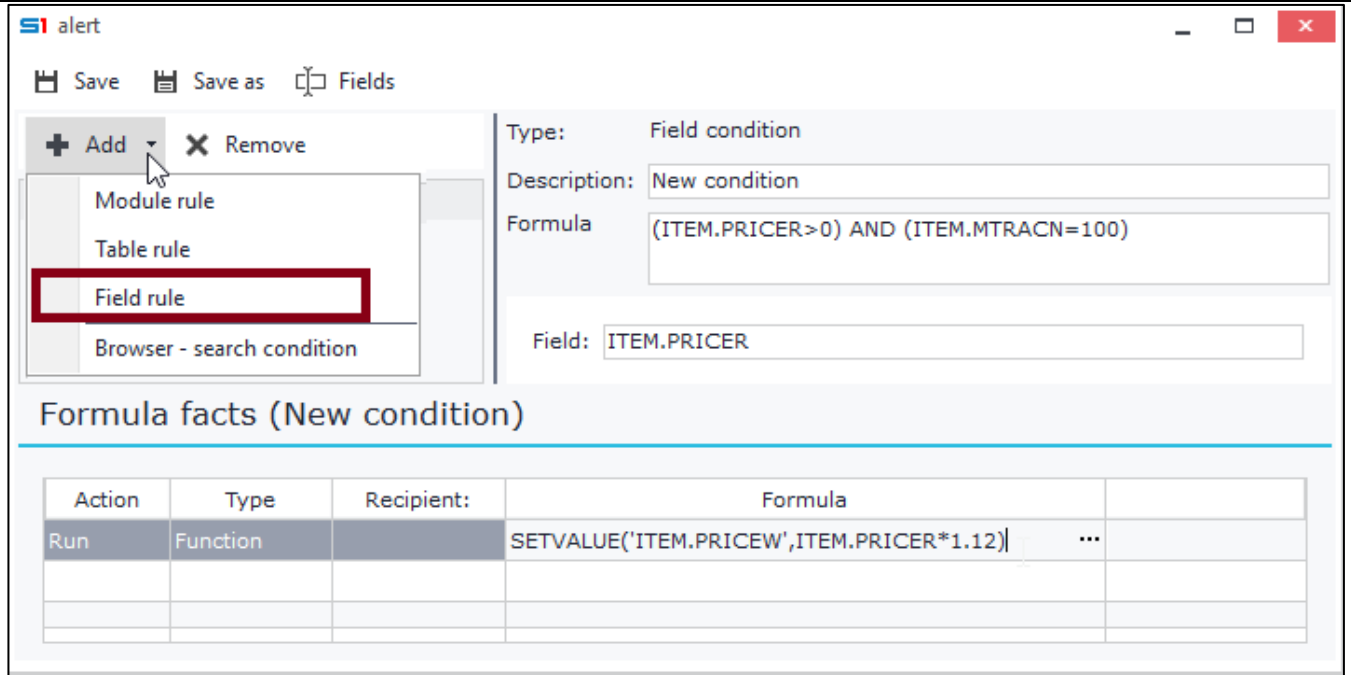


Figure A1

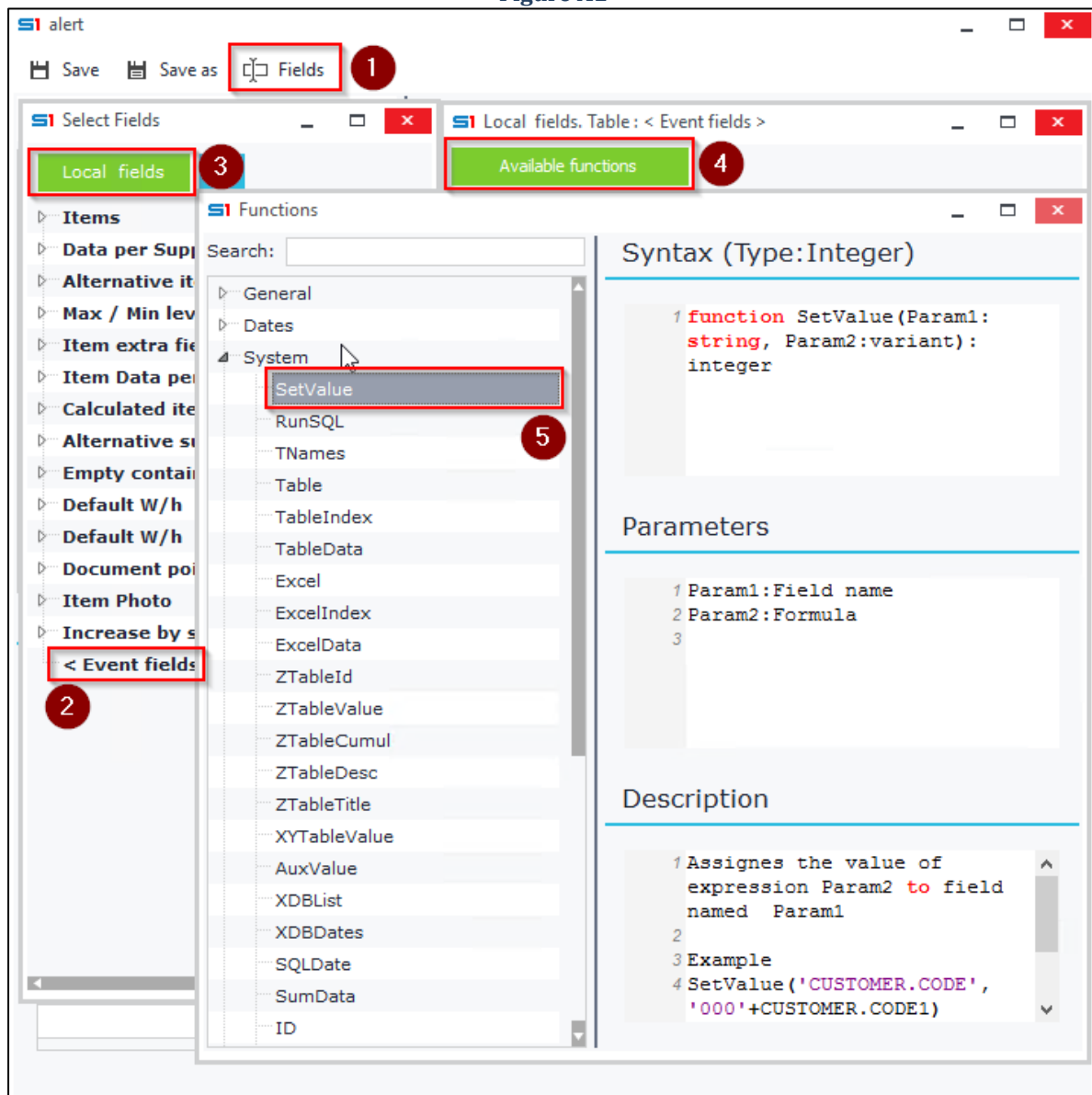


Figure A2

A.2 Table Rule

Table rules are triggered when table events occur. These events are summarized in the table below.

Be careful when enabling more than one events because they may fire twice. For example, if you enable both “Before Post” and “After Post” and choose to display a message, then this message will be displayed twice when you change the data of a datagrid and move to the next record.

EVENT	DESCRIPTION
On Insert	Fires when a new record is added in the dataset. Usually used for setting default values.
Before Post	Fires before changes to a dataset record are posted. Use it to perform actions that you want to occur before a dataset record is posted. It is commonly used for updating values, performing validation or raising custom error messages.
After Post	Fires after changes to a dataset record have been posted. Usually used to display / send a message or for calculating purposes in order to update values.
Before Delete	Fires before a dataset record is deleted. Use it to perform validation or to raise custom error messages.
After Delete	Fires after a dataset record is deleted. Use it to raise custom messages or perform any other updating actions.

Example

The EDA rule of the example in Figure A3 operates in the customers' object as follows:

When users add a new branch in table CUSBRANCH then the function SETVALUE('CUSBRANCH.SALESMAN', CUSTOMER.SALESMAN) is executed and sets as the default salesman of the branches the customer's salesman.

The screenshot shows the EDA(Alerts) window with the following configuration:

- Type:** Table rule
- Description:** Set Branch Salesman
- Formula:** 1
- Table:** CUSBRANCH
- Events:**
 - On Insert: ☒
 - Before Post: ☐
 - After Post: ☐
 - Before Delete: ☐
 - After Delete: ☐
- Formula facts (Set Branch Salesman):**

Action	Type	Recipient:	Formula
Run	Function		SETVALUE('CUSBRANCH.SALESMAN',CUSTOMER.SALESMAN)

Figure A3

A.3 Module Rule

This rule is used if you need to perform actions when object events occur. Notice that same actions can be executed in more than one events (multi check), but always be careful of accidentally running the same job twice.

The available events are the following:

EVENT	DESCRIPTION
On Insert	Occurs when a new record is created (click on toolbar button "New"). Usually used for setting default values.
On Locate	Occurs after locating a record in an object; display the record in screen form. Usually used for displaying messages or changing the layout of the form (hide elements).
Before DB Insert	Occurs before inserting a new record in an object For example, it fires when users click on "Save" button for a newly created record of an object, before the record is saved in the database.
After DB Insert	Occurs after inserting a new record in an object For example, it fires when users click on "Save" button for a newly created record of an object, after the record is saved in the database.
Before DB Update	Occurs before updating a record in an object For example, it fires when users edit / change the contents of a record and then click on "Save" button. The event rises before the record is saved in the database.
After DB Update	Occurs after updating a record in an object For example, it fires when users edit / change the contents of a record and then click on "Save" button. The event rises after the record is saved in the database.
Before DB Post	Occurs before saving record data of an object It merges the events "Before DB Insert" and "Before DB Update", so be careful not to combine it with these events because it will fire twice.
After DB Post	Occurs after saving record data of an object. It merges the events "After DB Insert" and "After DB Update", so be careful not to combine it with these events because it will fire twice.
Before DB Delete	Occurs before deleting a record of an object. Usually used when you need to prevent users from deleting a record, because it fires before the changes take effect.
After DB Delete	Occurs after deleting a record of an object. Usually used when you need to perform an action (send message / add log) when users delete a record, because it is triggered after the changes take effect.

Example

The EDA rule of the following example (Figure A4) is applied in sales documents when a record is saved. It prevents the record from being saved in the database if the transaction's currency is different from the customer's currency and the document is not an order (SALDOC.TFPRMS<>201).

If the above conditions are met, then it displays an exception message (Figure A5).

- Design a new Alert and add a "Module Rule" with any description (e.g. Currency check)
- Add the condition inside the formula text, using drag and drop from "Fields":
(SALDOC.SOCURRENCY<>SALDOC.TRDR_CUSTOMER_SOCURRENCY) AND (SALDOC.TFPRMS<>201).
- Select the event "Before DB Post"
- Add the message that will be displayed when the conditions are met, using "Error display" action.

CustomerCurrency

Save Save as Fields

+ Add - Remove

Currency check

Type: Module rule

Description: Currency check

Formula: (SALDOC.SOCURRENCY<>SALDOC.TRDR_CUSTOMER_SOCURRENCY) AND (SALDOC.TFPRMS<>201)

Events

On Insert: ☐ On Locate: ☐

Before DB Insert: ☐ After DB Insert: ☐

Before DB Update: ☐ After DB Update: ☐

Before DB Post: ☒ After DB Post: ☐

Before DB Delete: ☐ After DB Delete: ☐

Formula facts (Currency check)

Action	Type	Recipient:	Formula
Error display	On screen		Transaction Currency is different from Customer's

Figure A4

List

Standard Company: SISN000021- 05/04/18

Series: 7062 SISN Tvne: 7062 Sales Invoice - Delivery Note

Date: 05/04/18

General data

Customer: 266

Cust. branch:

Salesperson: 00012

Currency: 101 US Dollar Exchange rate: .3186 TRP exchange rate: 1

Comment:

Items

	Code	Description	Qty 1	Price	Disc. ...	Value
1	20019	Sofa 2 pcs.	15	800,00		12.000,00

Soft1 error!

Transaction Currency is different from Customer's

OK

Figure A5

A.4 Browser – Search Condition

The browser – search conditions operate as filters in the running browsers of the objects.

To create a browser condition, set the run condition (Formula) and the caption (Indication) that will appear in the browser as title.

In the "Formula facts" panel, select "Run" as Fact, "SQL filter" as Type and in the Text field enter the SQL filter, which will be inserted in the "where clause" of the SQL query of the browser when the condition is met.

Example

The following example (Figure A6) applies to object Meetings.

The application user with code 262 will be able to see only the meeting entries, where he is defined as the operator.

Moreover, the meeting lists' title will display the caption "Locked for 262" (Figure A7).

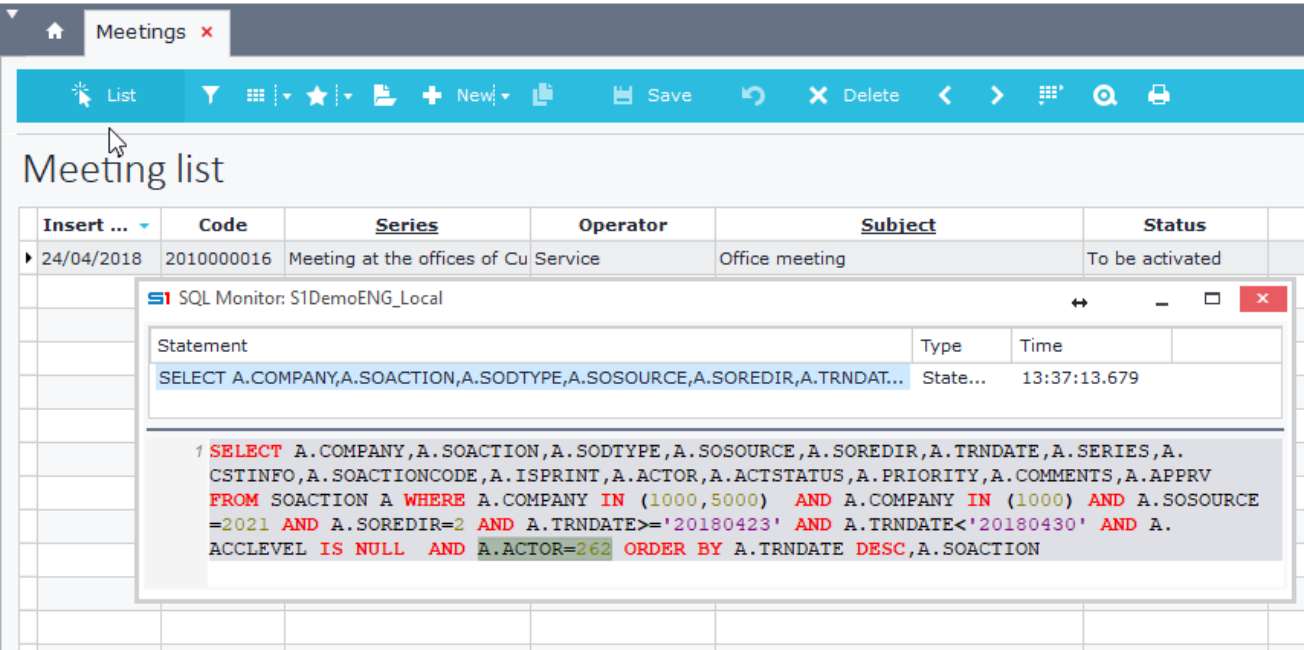


Figure A6

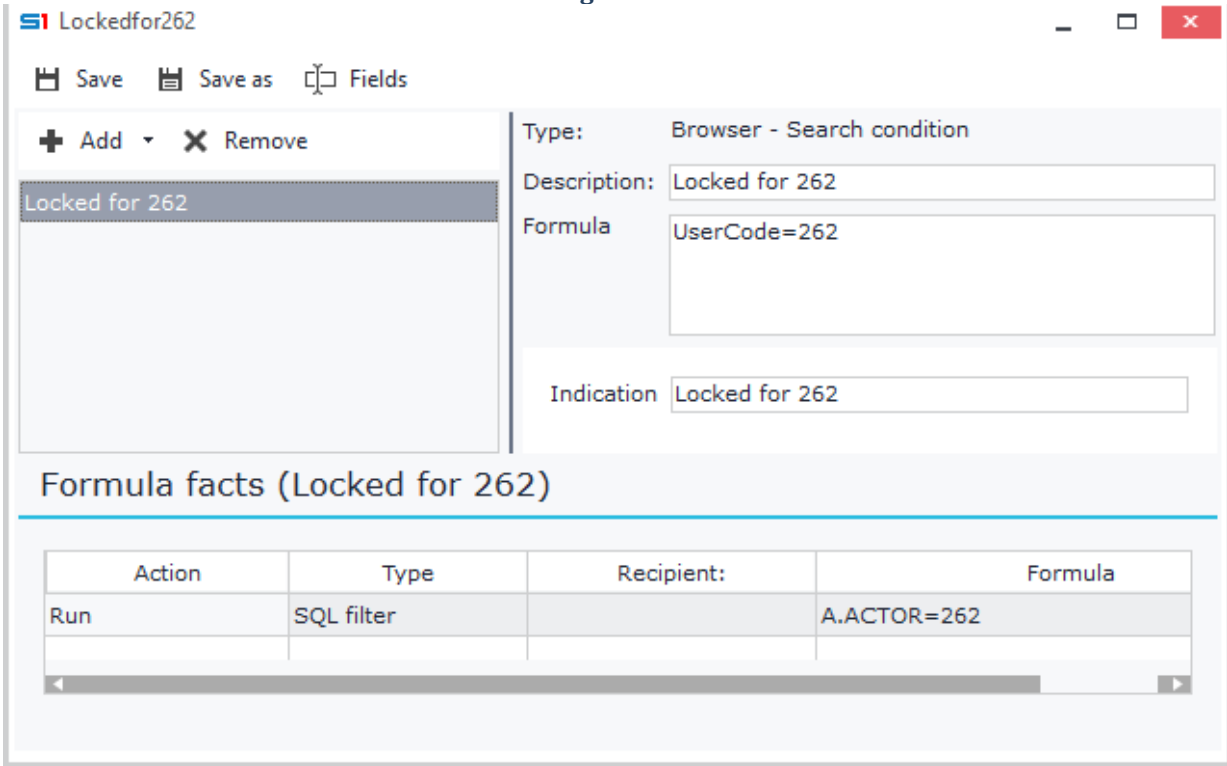


Figure A7

B. Actions

The datagrid “Actions” contains all the jobs that can be executed whenever the selected events fire and certain conditions are met. The available actions are the following (Figure B1.1):

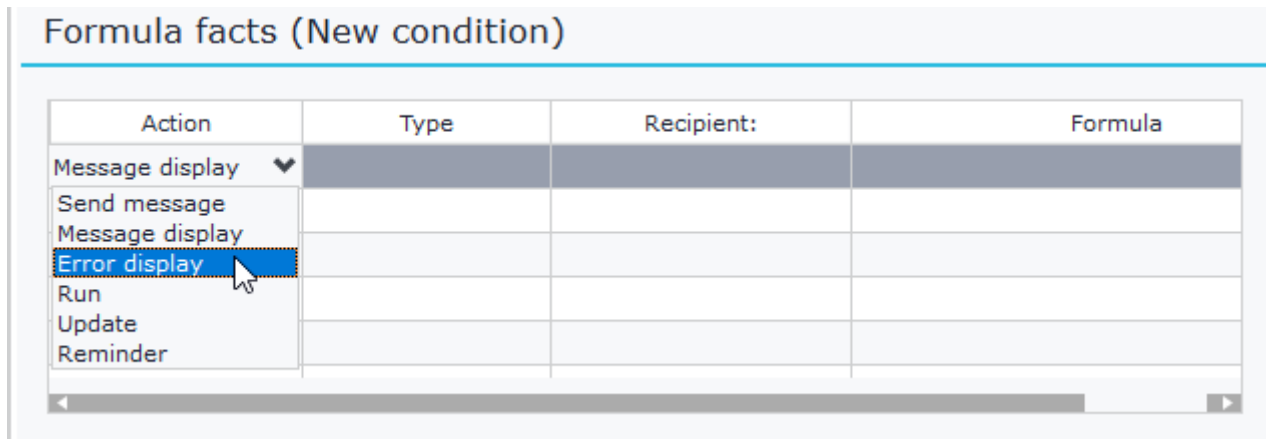


Figure B1.1

B.1 Send Message

Sends a message specified in the “Formula” column to the recipient of the column “Recipient”.

Available types are (Figure B1.2):

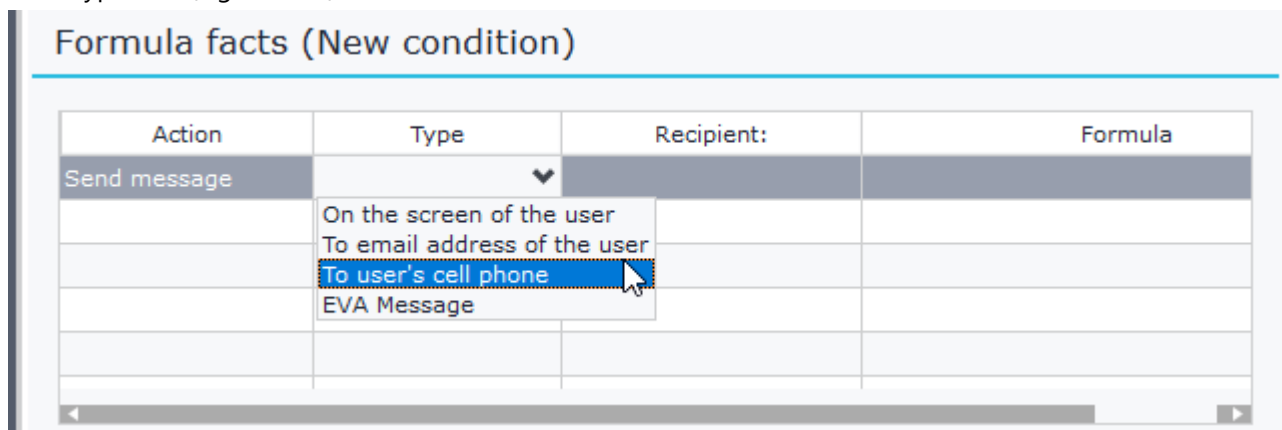


Figure B1.2

On the screen of the user

Sends the message to a user of the application. Select the user from the column “Recipient”. The message is saved and as soon as the selected user is available (logs in the application), the message will be displayed on the screen.

To email address of the user

Sends the message to the email address specified in the “Recipient” column. The recipient may be entered through a formula or a table field (e.g. TRDR.EMAIL). In order to use this type, you need to setup the outgoing email from the Settings window or from the SoftOne Remote Server.

To user's cell phone

Sends the message to the cell phone specified in the “Recipient” column. The recipient may be entered through a formula or a table field (e.g. TRDR.PHONE01). In order to use this type, you need to setup the cell phone from the SoftOne Remote Server or use a Web SMS provider from the Settings window.

B.2 Message Display (Notification)

Displays a message box on the screen of the login user. The message is entered inside the "Formula" column. Module and user-defined fields are also allowed to use.

Action	Type	Recipient:	Formula
Send message		SALDOC.TRDR_CUSTOM	'Your Quote Is Ready-
Send message		SALDOC.TRDR_CUSTOM	'Your Quote Is Ready-
Message display	On screen		'Your Quote Is Ready-Order Code: '+SALDOC.CMPFINCODE ...

Figure B2

B.3 Error Display (Exception)

Raises an exception message on the screen of the login user and stops the current process.

The message is entered inside the "Formula" column. Module and user-defined fields are also allowed to use.

Action	Type	Recipi...	Formula
Error display	On screen		Transaction Currency is different from Customer's

Figure B3

B.4 Run

Apart from the display or sending messages, EDA allows you to run various commands. The available types / run commands are (Figure B4):

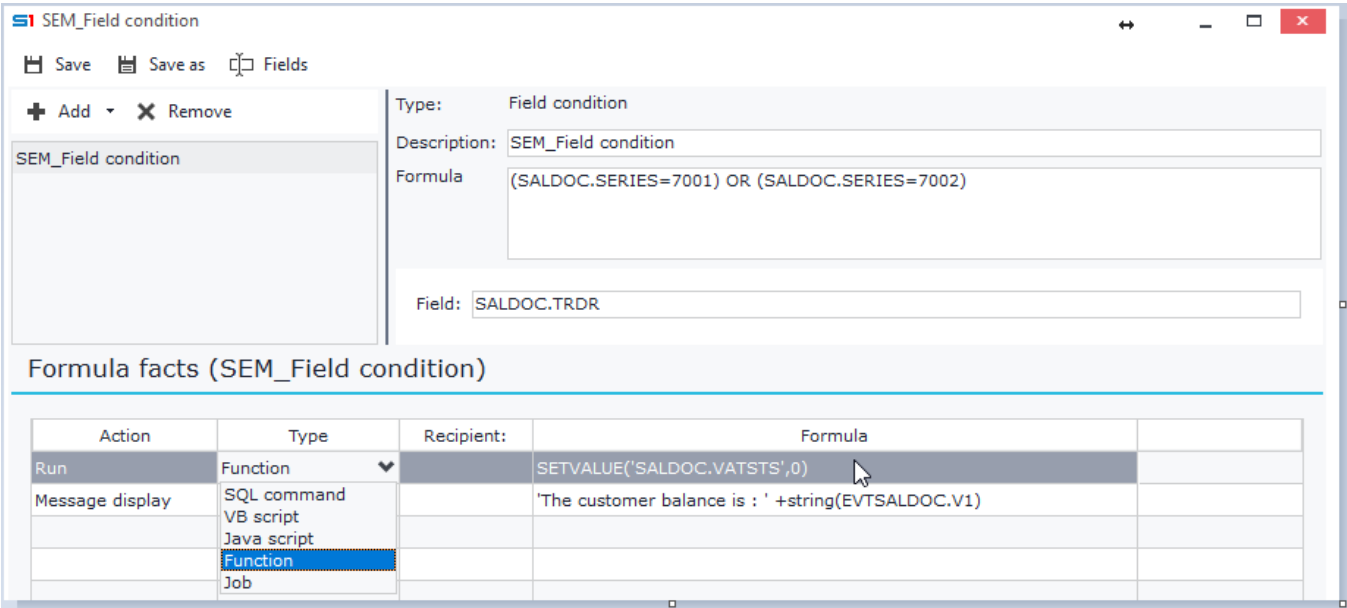


Figure B4

SQL Command

Runs the SQL statement entered in the "Formula" column.

VB Script

Runs the VB script entered inside the "Formula" column.

Java Script

Runs the Java script entered inside in the "Formula" column.

Function

Runs an internal function of the program, which is entered inside the "Formula" column.

Job

Runs the SQL statement entered in the "Formula" column. It can only be used for object rules and only for the events related to the database. On "before Event" actions, an event confirmation question is displayed. On "after Event " actions, an event completed message is displayed. In this case, you don't need to set anything but the formula as all the rest elements is calculated automatically.

B.5 Update

This type of action can be used only in object rules and it displays a validation message box before committing the changes to the database.

B.6 Reminder

This type of action is the same as sending a message to a user of the application and it only differs to the fact that the message is saved and displayed as a reminder.

C. Case Studies

C.1 Field Rule – Message display

Each time the log in user changes the address of a customer and the turnover of the customer is greater than zero, then the operator's screen will display the message **"Changed the address of the customer [customer's name]"** (Figure C1.2).

- Create an EDA inside Customers' module
- Add a Field Rule
- Drag and drop **Turnover** inside Formula and enter: **CUSTOMER.TRDR_CUSFINDATA_TTURNOVR>0**
- Drag and drop **Address** (Figure C1.1) in **Field** textbox.

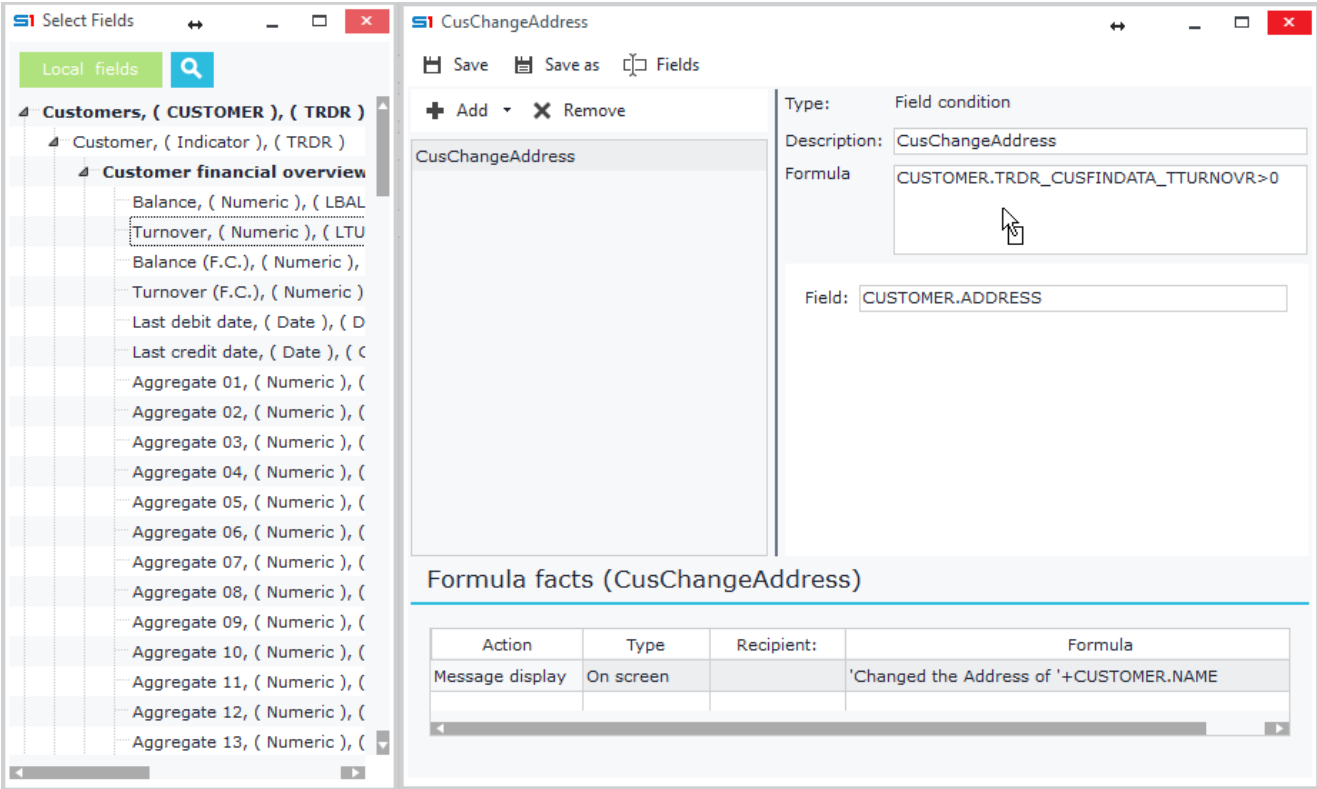


Figure C1.1

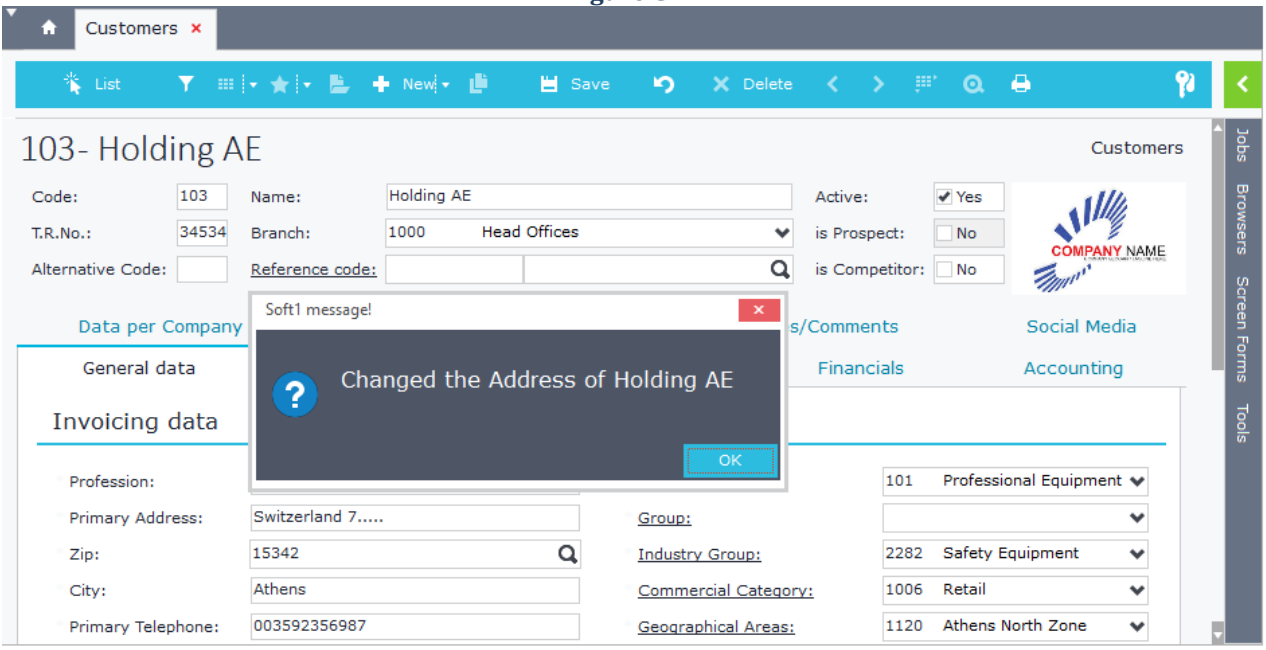


Figure C1.2

C.2 Table Rule – Error Message Display

Prohibition on deletion of a customer's branch when the customer's turnover is positive, displaying the message "Deletion is aborted" (Figure C2.2).

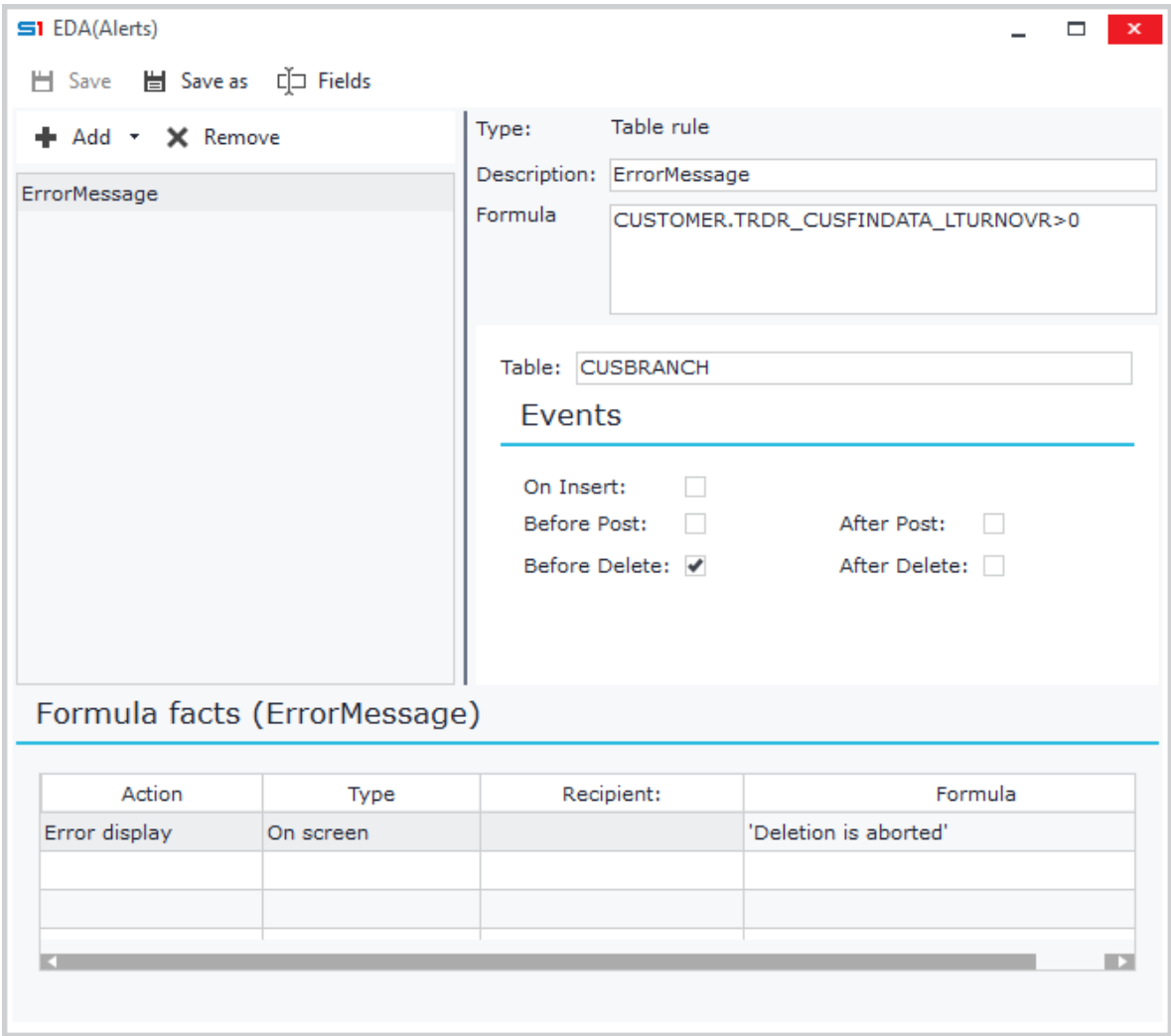


Figure C2.1

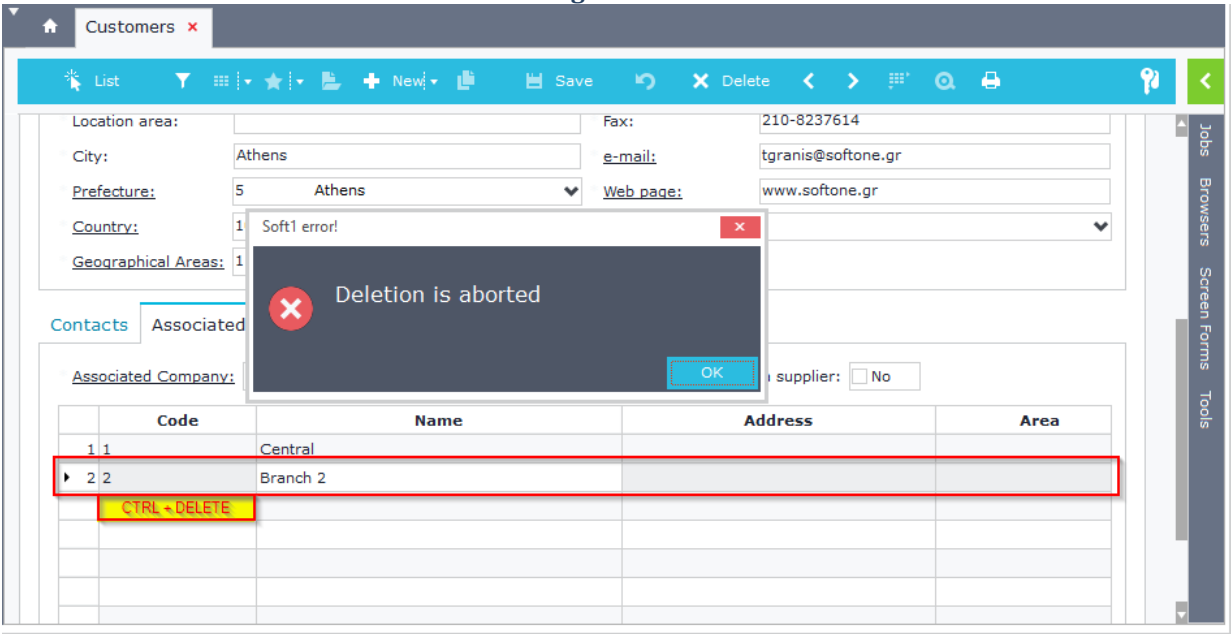


Figure C2.2

C.3 Object Rule – Send Message

Once the operator chooses to delete a customer with a positive turnover, the application will stop the process showing a message to his screen. The message will also be sent and displayed to the “Administrator’s” screen (Figure C3).

The screenshot shows the EDA(Alerts) window with the following configuration:

- Type:** Table rule
- Description:** ErrorMessage
- Formula:** CUSTOMER.TRDR_CUSFINDATA_LTURNVR>0
- Table:** CUSBRANCH
- Events:**
 - On Insert: ☐
 - Before Post: ☐
 - After Post: ☐
 - Before Delete: ☒
 - After Delete: ☐
- Formula facts (ErrorMessage):**

Action	Type	Recipient:	Formula
Error display	On screen		'Deletion is aborted'
Send message	On the screen of t	Administrator	'Deletion was aborted. User='UserCode

Figure C3

C.4 Browser – Search Condition

The operator in the workstation named “user1” can only view the item browser records that the field “Code” starts with “000-000”. The title of browser will also display the phrase ‘Only for computer user1’.

The screenshot shows the EDA(Alerts) window with the following configuration:

- Type:** Browser - Search condition
- Description:** BrowserCheck
- Formula:** ComputerName='user1'
- Indication:** Only for computer user1
- Formula facts (BrowserCheck):**

Action	Type	Recipient:	Formula
Run	SQL filter		'A.CODE LIKE "000-00%" '

Figure C4

5. USER-DEFINED FIELDS

A. [General Features](#)

B. [Operations Commands](#)

C. [Built-In Functions](#)

A. General Features

User-defined fields (Local and General Fields) are used for displaying data using simple or advanced calculations between database fields. They can be used inside browsers, EDA (alerts), imports, reports, printout forms and in any other tool that they are available as an option. Figure A1 shows an example of user-defined fields inside “Sales Documents” browser.

“**General Fields**” are designed in the same way as “**Local Fields**” and the only difference is that they are available from every tool (e.g. Browsers) of the entity that they are saved. For example, if you create a “Local field” inside a browser of Customers object, then this field will be available for use only in this specific browser. But, if instead you create this field inside “General Fields”, then it will be available from every browser of Customers object.

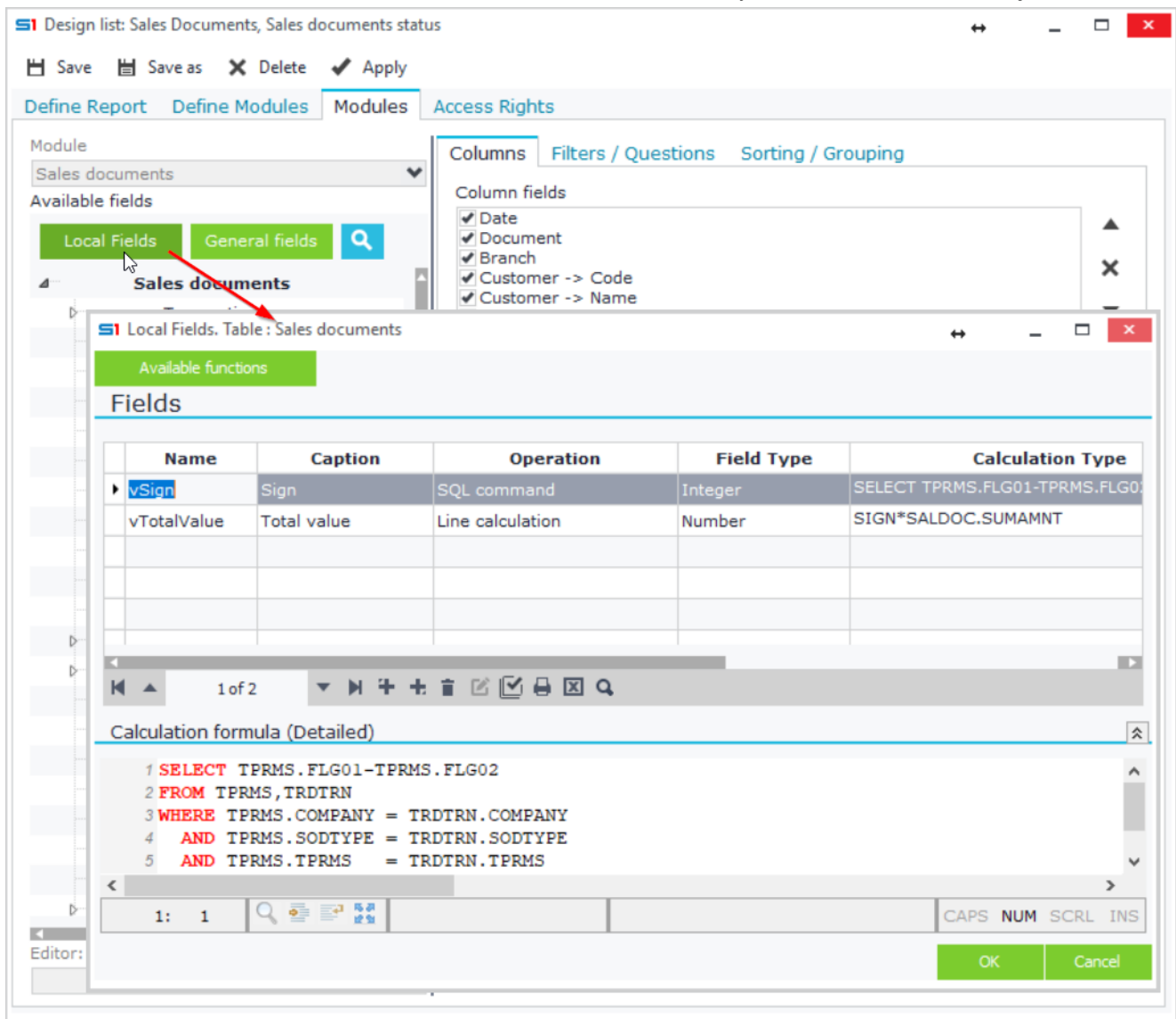


Figure A1

"**Local Fields**" or "**General Fields**" buttons display a window that lets you create the user-defined fields inside the grid "Fields" (Figure A2), which has the following columns.

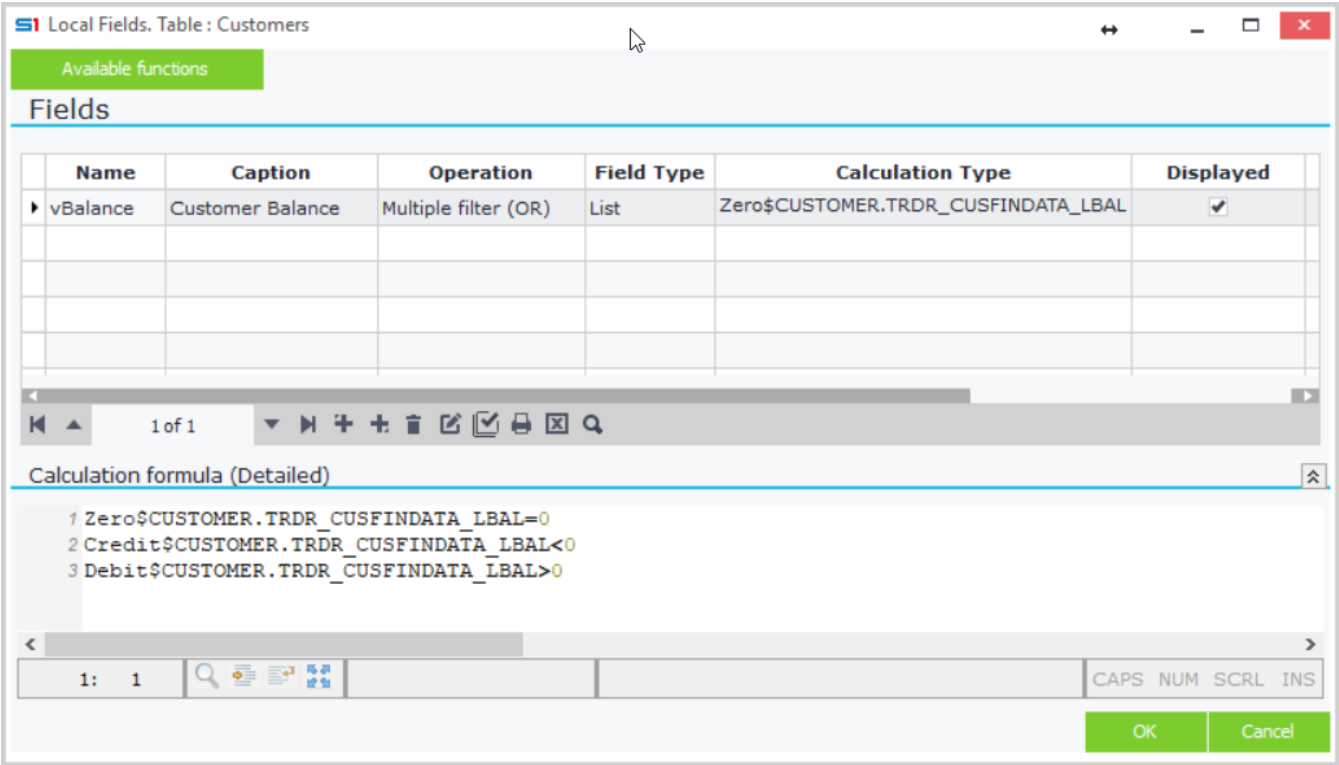


Figure A2

Name

Field name that can be used as reference between the user-defined fields of the current browser, alert, etc. The default name for every new field begins with the letter V and is followed by an auto increment number. It is recommended to change this name using a name that meets your needs.

Caption

Defines the title of the field, as it will be displayed in dialog filters, columns, etc.

Operation

Defines the way that data will be accessed and are discussed in the following sections. All the available operation options are summarized in the table below.

Field type

Field type options differ depending on the selected operation. For example, if the operation is "SQL filter" then the only available option is "Boolean".

Calculation formula

This field is used for entering the formula commands that will be used to calculate the value of the defined field.

Displayed

Select whether the field is available in the design columns of the tool through the **Displayed** option.

Note: Defined fields that use "Function" operation cannot be displayed, since they can only be used from other defined fields.

B. Operation Commands

The following table contains the available operation options and the SoftOne controls they can be used (Figure A3).

Operation Commands / SoftOne Tools							
	Browsers	Design Reports	Open Reports	Imports	Printout forms	EDA (Alerts)	Remote Commands
Line Calculation	✓	✓	✓	✓	✓	✓	✓
Calculation of line + Totals	✓	✓					
SQL command	✓	✓		✓	✓	✓	
Question	✓	✓	✓				
SQL filter	✓	✓					
Single filter	✓	✓					
Multiple filter (OR)	✓	✓					
Multiple filter (AND)	✓	✓					
Function	✓	✓	✓	✓	✓	✓	✓
Multi Result SQL	✓	✓		✓	✓		

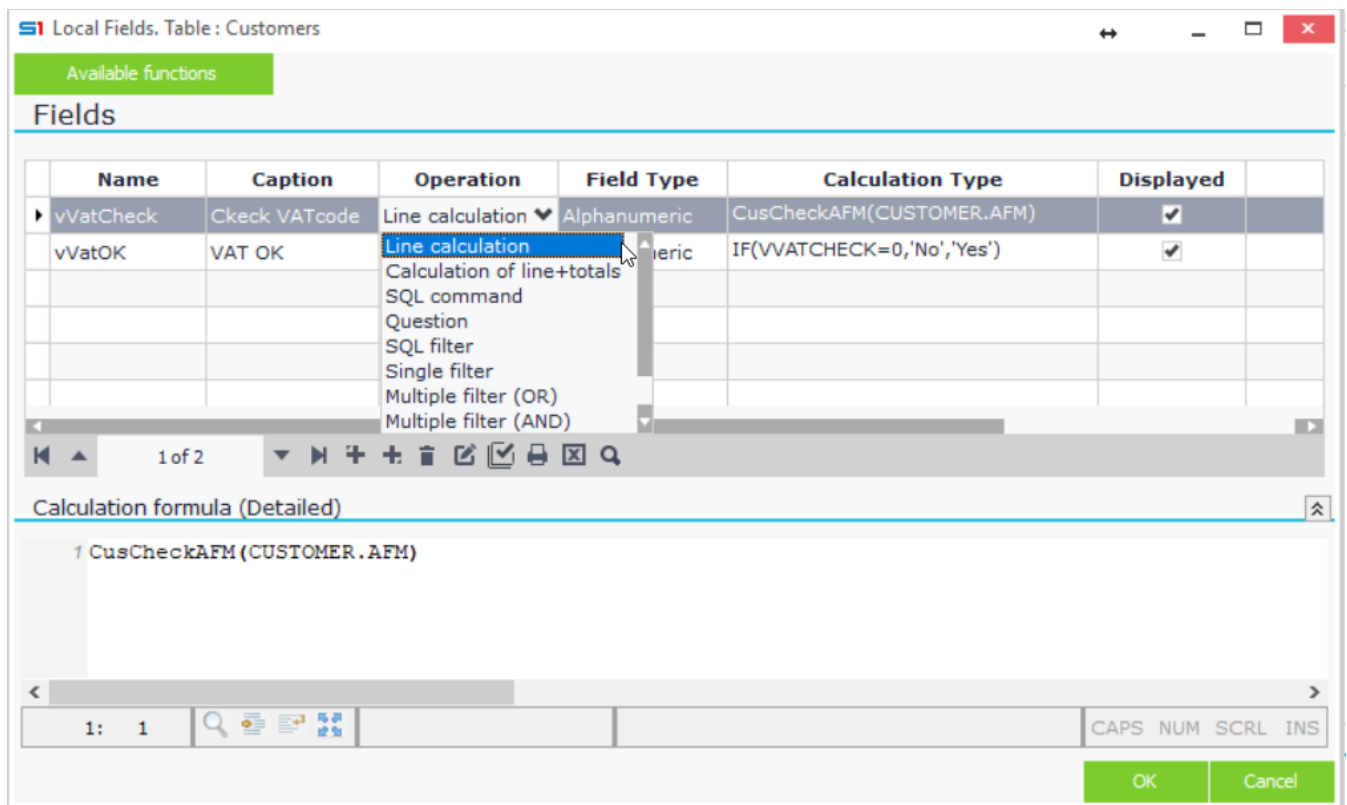


Figure A3

B.1 Line Calculation

This operation is used for displaying data in columns, using math calculations or built-in functions. Calculations are executed for every record of the dataset. This means that if for example, you create a “Line Calculation” field inside a browser that at runtime it displays 1000 records, then the calculation will be executed 1000 times (each for every record) (Figure B1.1).

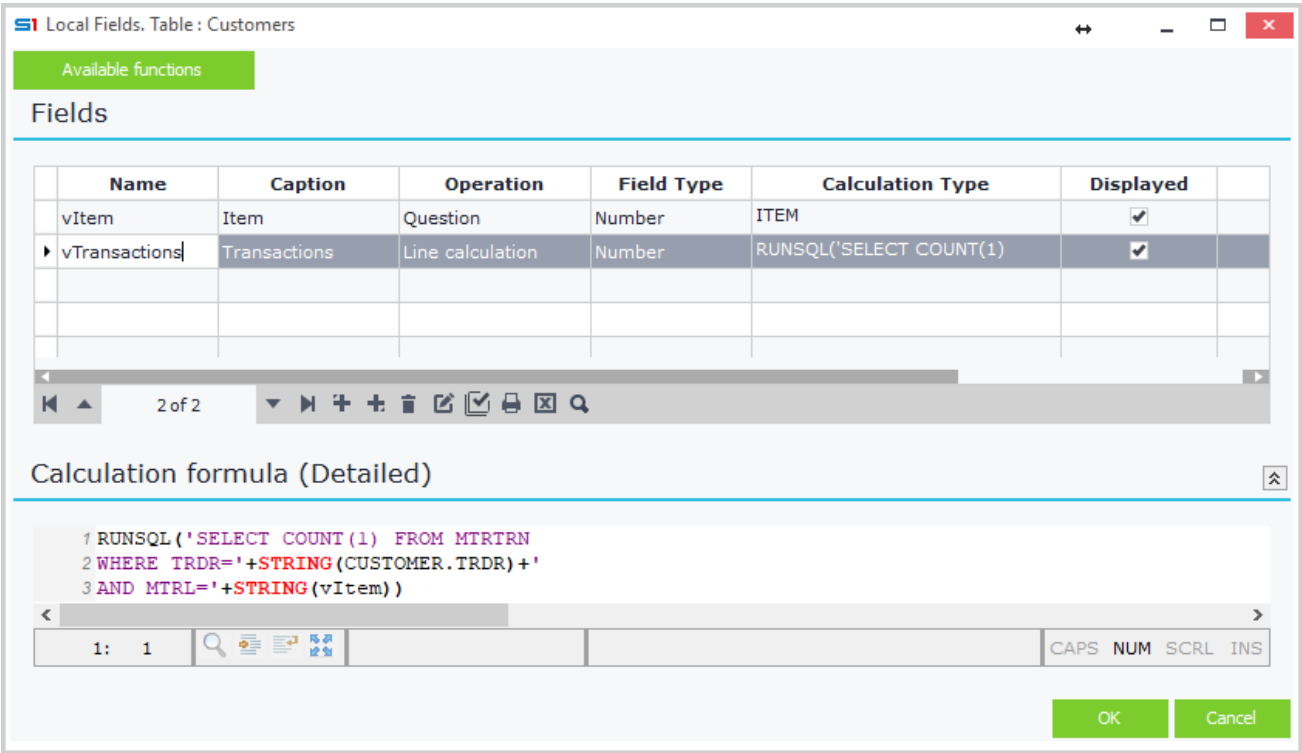


Figure B1.1 – Line calculation field using RUNSQL function

Suppose that you use the built-in function **RUNSQL** for retrieving results in a field, then one SQL query will be executed per record displayed. This will cause **slow performance**, because every record will produce one SQL query that will be executed from the database server to return the results (Figure B1.2). This means that user-defined fields using RUNSQL function may cause reports or browsers delay, so it is much better to use the operation option “**SQL command**” (if available from the specific control) when you need to use SQL statements.

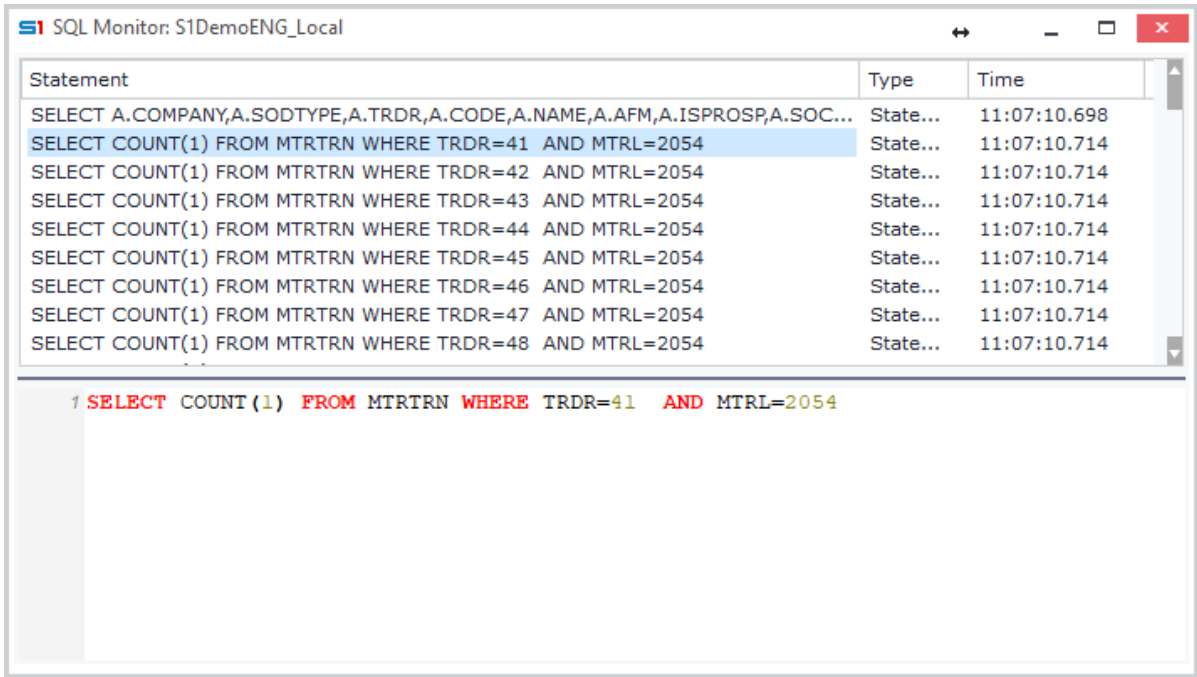


Figure B1.2 – SQL Monitor

Calculations between fields is achieved using a defined field with "Line Balance / Calculation" operation. Select the field type (e.g. Number) and then enter or drag and drop the fields, applying the preferred math formulas.

Note: All built-in functions and system parameters, e.g. UserCode (Login User), CompanyCode (Login Company), etc. are available through this operation command.

Example 1 (Browser)

User-Defined field inside the Sales Statistics browser that calculates the gross profit, using the expression: Sales value - Sales cost.

Create a defined field and select "Line calculation" as operation and "Number" as field type (Figure B1.3).

Enter the calculation formula (or drag and drop the fields): VSALSTAT.SalVal-VSALSTAT.SCpurmtk

The screenshot displays the 'Local Fields. Table: Sales statistics' dialog box. The 'Fields' table is as follows:

Name	Caption	Operation	Field Type	Calculation
vGrossProfit	Gross Profit	Line calculation	Number	

The 'Calculation formula (Detailed)' section shows the formula: `VSALSTAT.SalVal-VSALSTAT.SCpurmtk`.

Below the dialog, the 'Sales statistics-GrossProfit' browser is shown, displaying a table with the following data:

Date	Code	Name	Code	Description	Sales qty 1	Sales value	Gross Profit
01/01/2018	117	Kalampoka Melina	10001	TV LED 32 LG RTDXA32	10,00	2.000,00	2.000,00
01/01/2018	117	Kalampoka Melina	10002	TV LCD 21	1,00	200,00	200,00
01/01/2018	117	Kalampoka Melina	10003	Digital Camera 8 MP	2,00	200,00	200,00
01/01/2018	908	General Retail customer	10009	Home cinema 1000 Watt	1,00	74,63	74,63
01/01/2018	101	Salesconsul GmbH	10003	Digital Camera 8 MP	1,00	51,48	51,48
01/01/2018	101	Salesconsul GmbH	10002	TV LCD 21	25,00	7.500,00	7.500,00
01/01/2018	101	Salesconsul GmbH	10004	Photo michachi DSLR 10 MP	1,00	314,60	314,60
01/01/2018	101	Salesconsul GmbH	10005	Flash Camcorder	16,00	8.236,80	8.236,80
01/01/2018	101	Salesconsul GmbH	10006	Flash Full HD Camcorder	4,00	1.258,40	1.258,40
01/01/2018	101	Salesconsul GmbH	10007	HDD 60GB Camcorder	20,00	6.292,00	6.292,00
01/01/2018	101	Salesconsul GmbH	10008	Digital Photo Frame 8 "	1,00	57,20	57,20
01/01/2018	101	Salesconsul GmbH	10009	Home cinema 1000 Watt	1,00	88,40	88,40
01/01/2018	101	Salesconsul GmbH	10010	Crutches	1,00	180,00	180,00
01/01/2018	101	Salesconsul GmbH	10011	Pharmacy kit	2,00	83,20	83,20
01/01/2018	101	Salesconsul GmbH	10012	Liquid decontamination (Lots)	2,00	34,00	34,00
					890,00	279.504,47	279.504,47

Figure B1.3 – Calculations between Fields

Example 2 (Report)

User-Defined field inside the report “Sales per customer” that displays the concatenation of the name and surname of Salesmen.

Create a new user-defined field, use the operation option “Line Balance” and select “Alphanumeric” in Field Type. Drag and drop the fields **NAME** and **NAME2** from **Salesperson** node or enter the following formula: CUSTOMER.SALESMAN_PRSNIN_NAME+' '+CUSTOMER.SALESMAN_PRSNIN_NAME2 (Figure B1.4).

Notice that when you drag and drop the field “PRSN.NAME”, SoftOne uses the shorter (alias) name X_SNAME.

The screenshot is divided into two main parts. The top part shows the 'Design report' window for 'Sales statistics per customer'. On the left, under 'Available fields', the 'Salesperson' node is expanded, showing 'Employee Data, (PRSNIN), (PRSN)'. The 'Name, (Alphanumeric), (NAME)' field is selected and highlighted with a red arrow. On the right, the 'Fields' table shows a new field 'vFullName' with the operation 'Line calculation' and field type 'Alphanumeric'. Below this, the 'Calculation formula (Detailed)' section shows the formula: `CUSTOMER.X_SNAME+' '+CUSTOMER.SALESMAN_PRSNIN_NAME2`. The bottom part shows the 'Sales per Customer' report. It includes a header with company information and a table with columns for No., Code, Name, Salesman Full Name, and monthly sales data (January, February, March) for Quantity and Value.

No.	Code	Name	Salesman Full Name	January		February		March		Quantity
				Quantity	Value	Quantity	Value	Quantity	Value	
1	310	3rd District General Hospital	Angelo Clarcson	0,00	0,00	0,00	0,00	0,00	0,00	0,0
2	121	Alexis Chalkidis	Angelika Houston	102,00	1.127,00	0,00	0,00	0,00	0,00	18,0
3	304	Anagnostopoulos Ioannis (potentia...	Brand Rose	115,00	1.919,48	0,00	0,00	0,00	0,00	0,0
4	115	Anastasios Kontomari	Karol Herrera	100,00	25.168,00	1,00	0,00	0,00	3,59	0,0
5	281	Anastasios Martonis	Karol Herrera	0,00	0,00	1,00	12.000,00	5,00	22,50	0,0
6	238	Andreas Petrides	Jimmy Hendrix	0,00	0,00	7,00	1.500,00	-3,00	-157,20	0,0
7	248	Apostolos Panagiotou	Brand Rose	0,00	0,00	9,00	4.226,30	0,00	0,00	0,0

Figure B1.4

B.2 Calculation of line+totals

This operation is the same as “Line Balance / Calculation”. It differs to the fact that it can only be used in numeric field types for displaying the totals if there is grouping in the browser or report.

This means that in case you want to display the totals of a defined field in browsers or reports that have grouped data then you have to use this operation.

It uses exactly the same commands and functions as “Line Calculation”.

Local Fields. Table : Sales statistics

Available functions

Fields

Name	Caption	Operation	Field Type	Calculation
vCostPrice	CostPrice	Function	Number	CostPrice
vSalesCost	SalesCost	Line calculation	Number	MCase(VCOSTPRICE, 1,V
vSalesValue	SalesValue	Line calculation	Number	VSALSTAT.SalVal
vGrossProfit	GrossProfit	Line calculation	Number	VSALESVALUE-VSALESCC
vSumGrossProfit	%GrossProfit	Calculation of line+totals	Number	VGROSSPROFIT*100/VSA

5 of 5

Calculation formula (Detailed)

1 VGROSSPROFIT*100/VSALESVALUE

1: 1

OK Cancel

Sales Statistics

List

Turnover PIVOT

Sales statistics

	Sales qty 1	Sales value	SalesCost	SalesValue	GrossProfit	%GrossProfit
Subscription [1]	27.195,00	967.109,14	345.952,86	967.109,14	621.156,28	64,23
Adventure [669]	4.028,00	57.141,26	8.460,41	57.141,26	48.680,85	85,19
Medical [178]	949,00	36.389,15	24.391,84	36.389,15	11.997,31	32,97
Cars [177]	2.431,00	68.816,64	30.491,40	68.816,64	38.325,24	55,69
Optics / Lenses [45]	1.318,00	67.540,00	34.250,64	67.540,00	33.289,36	49,29
Dough Products [169]	3.548,00	384.974,00	73.100,39	384.974,00	311.873,61	81,01
Beverage Products [27]	17,00	11.952,00	749,36	11.952,00	11.202,64	93,73
Meat [34]	210,00	5.250,00	3.481,29	5.250,00	1.768,71	33,69
Gas Stations [108]	150,00	32.246,00	5.372,11	32.246,00	26.873,89	83,34
Publications [112]	298,00	14.508,00	5.912,71	14.508,00	8.595,29	59,25
Software [359]	262,00	5.978,00	3.279,08	5.978,00	2.698,92	45,15
	36,00	360,00	216,00	360,00	144,00	40,00
	10,00	7.500,00	1.734,26	7.500,00	5.765,74	76,88
s	2.410,00	317.443,86	75.822,78	317.443,86	241.621,08	76,11
	5.415,00	11.007.451,42	262.185,97	11.007.451,42	10.745.265,45	97,62
	529,20	11.751,30	7.109,12	11.751,30	4.642,18	39,50
	12.114,00	2.601.525,26	400.356,63	2.601.525,26	2.201.168,63	84,61
	626,20	56.150,00	15.760,29	56.150,00	40.389,71	71,93
	109.054,40	23.099.840,53	2.508.851,46	23.099.840,53	20.590.989,07	89,14

Sorting / Grouping

Sorting: Date

Grouping: Comm.Category

Figure B2.1

B.3 SQL command

This operation is used in SQL statements for retrieving data from the database. It substitutes the RUNSQL function in browsers, reports and printout forms. It is executed within the select statement of the report, so it runs faster.

Example 1 (Browser)

User-defined field in Items browser that displays the Name field of the Item commercial category.

Create a defined field in Item browser and select SQL Command as operation, Alphanumeric as field type and enter the formula:

SELECT NAME FROM MTRCATEGORY WHERE SODTYPE=51 AND COMPANY=:X.SYS.COMPANY AND MTRCATEGORY=A.MTRCATEGORY (Figure B3.1)

Note that the alias A is used for linking the primary key field MTRCATEGORY of the table MTRCATEGORY with the MTRCATEGORY field of item. Aliases can be easily spotted in the SQL statement that runs if you execute the browser. Sql statements are displayed inside the SQL Monitor tool (Figure B3.2).

Name	Caption	Operation	Field Type	Calculation Type	Displayed
vComCat	Comm. Category	SQL command	Alphanumeric	SELECT NAME FROM	<input checked="" type="checkbox"/>

Calculation formula (Detailed)

```

1 SELECT NAME
2 FROM MTRCATEGORY
3 WHERE SODTYPE=51 AND COMPANY=:X.SYS.COMPANY AND MTRCATEGORY=A.MTRCATEGORY
  
```

OK Cancel

Figure B3.1

Statement	Type	Time
SELECT A.COMPANY,A.SODTYPE,A.MTRL,A.CODE,A.NAME,A.CODE1,A.CODE2,A.MT...	State...	13:55:12.480

```

1 SELECT A.COMPANY,A.SODTYPE,A.MTRL,A.CODE,A.NAME,A.CODE1,A.CODE2,A.MTRTYPE,A.
MTRTYPE1,A.MTRCATEGORY,ISNULL(A.PRICEW,0) AS PRICEW,ISNULL(A.PRICER,0) AS PRICER
,(SELECT NAME FROM MTRCATEGORY WHERE SODTYPE=51 AND COMPANY=1000 AND MTRCATEGORY=
A.MTRCATEGORY) AS V1 FROM MTRL A WHERE A.COMPANY=1000 AND A.SODTYPE=51 ORDER BY A
.CODE,A.MTRL
  
```

Figure B3.2 – SQL Monitor

B.4 Question

This operation is used for defined fields that will be displayed as dialog filters in browsers or reports. It is usually used combined with the operation SQL filter for applying the filters inside the where clause of SQL statements.

Example (Browser)

In Customers browser create a user-defined field that will display the records that don't have transactions within a date range defined in dialog filters. Set also the current year as the default value of the date.

Create two user-defined fields **vDateFilter** and **vRunFilter** as in Figure B4.1.

Local Fields, Table: Customers

Available functions

Fields

Name	Caption	Operation	Field Type	Calculation Type	Displayed
vDateFilter	Date	Question	Date	\$DATERANGE	<input checked="" type="checkbox"/>
vRunFilter	Filter	SQL filter	Boolean(N/O)	NOT EXISTS	<input checked="" type="checkbox"/>

2 of 2

Calculation formula (Detailed)

```

1 NOT EXISTS
2 (SELECT 1 FROM TRDTRN
3 WHERE SODTYPE=13
4 AND TRDR=A.TRDR
5 AND TRNDATE BETWEEN [SQLDATE(USR.TRDRvDateFilterL)] AND [SQLDATE(USR.TRDRvDateFilterH)])

```

1: 1

CAPS NUM SCRL INS

OK Cancel

Figure B4.1

Field **vDateFilter** uses the command **\$DATERANGE** which allows the user to select dates from a range control (Date from – Date to) (Figure B4.2).

Customers

List Filter New Save Delete

Customers who are not initiated by date

Code, from: to:

Name, from: to:

T.R.No., from: to:

Date: Current Week (30/04/2018 - 06/05/2018)

Filter: ☒ Yes

Figure B4.2 – Dialog filters using Question Operation

The second defined field **vRunFilter** uses a SQL statement for filtering the records, according to the selected date range. The formula is: NOT EXISTS (SELECT 1 FROM TRDTRN WHERE SODTYPE=13 AND TRDR=A.TRDR AND TRNDATE BETWEEN [SQLDATE(USR.TRDRV1L)] AND [SQLDATE(USR.TRDRV1H)])

Square brackets [] are escape characters and they can be used inside **SQL commands** in order to add SoftOne functions or any other non-SQL commands. Notice that for **SQL Filters** you also need to add a colon **[: xxx]**.

In our example we have used the internal function **SQLDATE**, which returns the date in SQL format ('yyyymmdd') and the parameters **USR.TRDRV1L** and **USR.TRDRV1H**.

Figure B4.3 displays the SQL statement that is executed at runtime through the SQL Monitor tool.

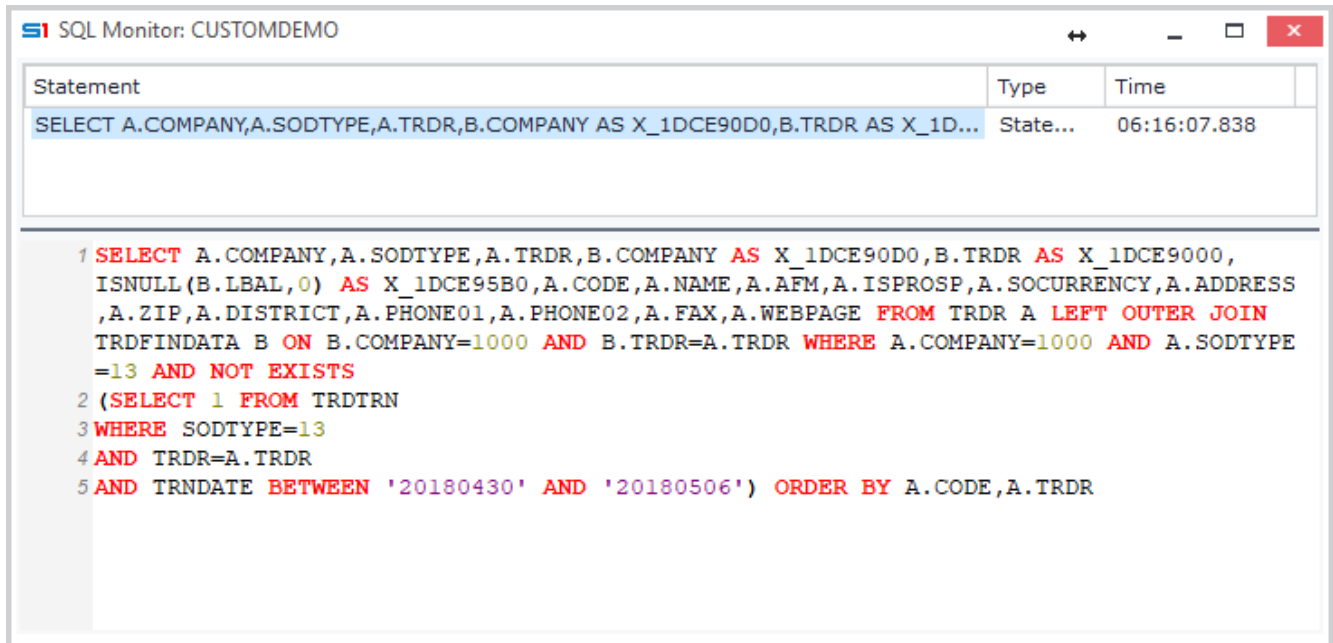


Figure B4.3 – SQL Monitor

The **USR.TRDRV1L** and **USR.TRDRV1H** return the values the user will enter inside the dialog filter field **vDateFilter** at runtime. Names **TRDRV1L** and **TRDRV1H** are found with the use of the shortcut keys **CTRL+SHIFT+F12** that is performed in the dialog filter field **vDateFilter** at runtime (Figure B4.4).

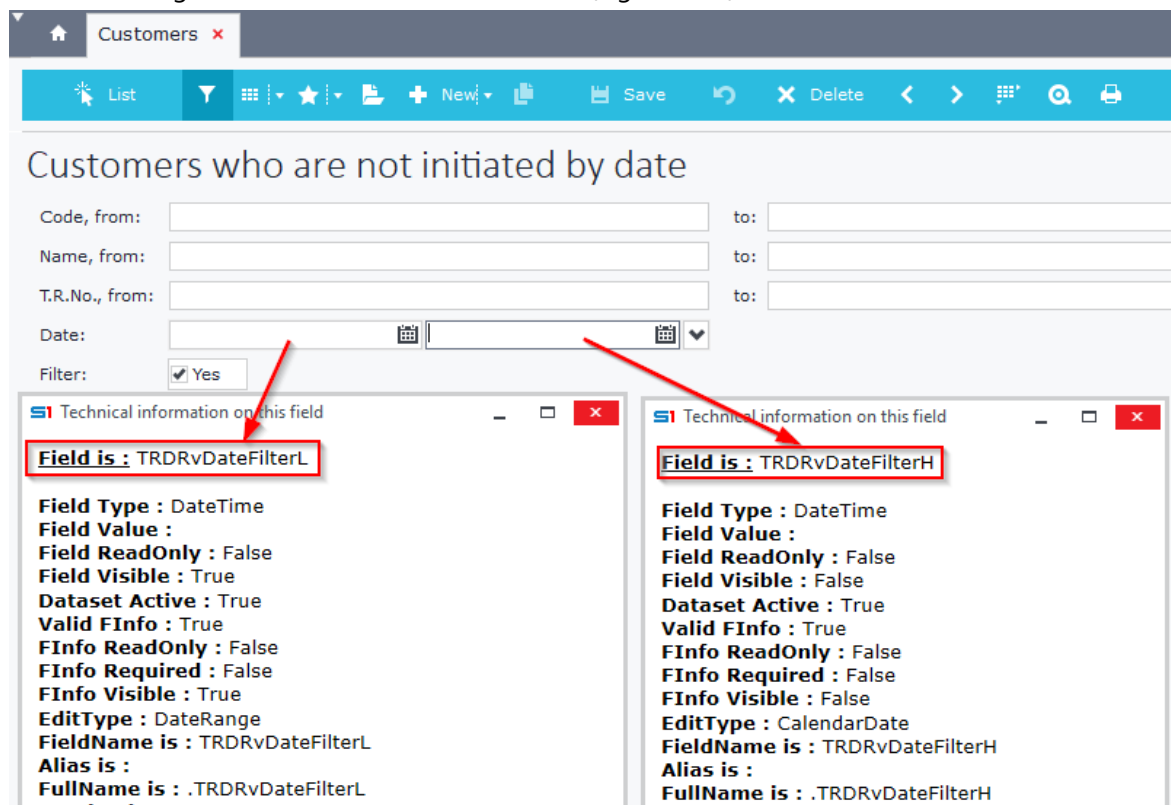


Figure B4.4 – CTRL+SHIFT+F12

The default value is set through the filters tab of the browser in design mode. Select the field **vDateFilter** and enter the value **131**, which is the value for **Current Year**, as it is defined through the Date Ranges object (Figure B4.5). The filtering is performed at runtime when users select a date range and enable (Yes) the dialog filter **vRunFilter**.

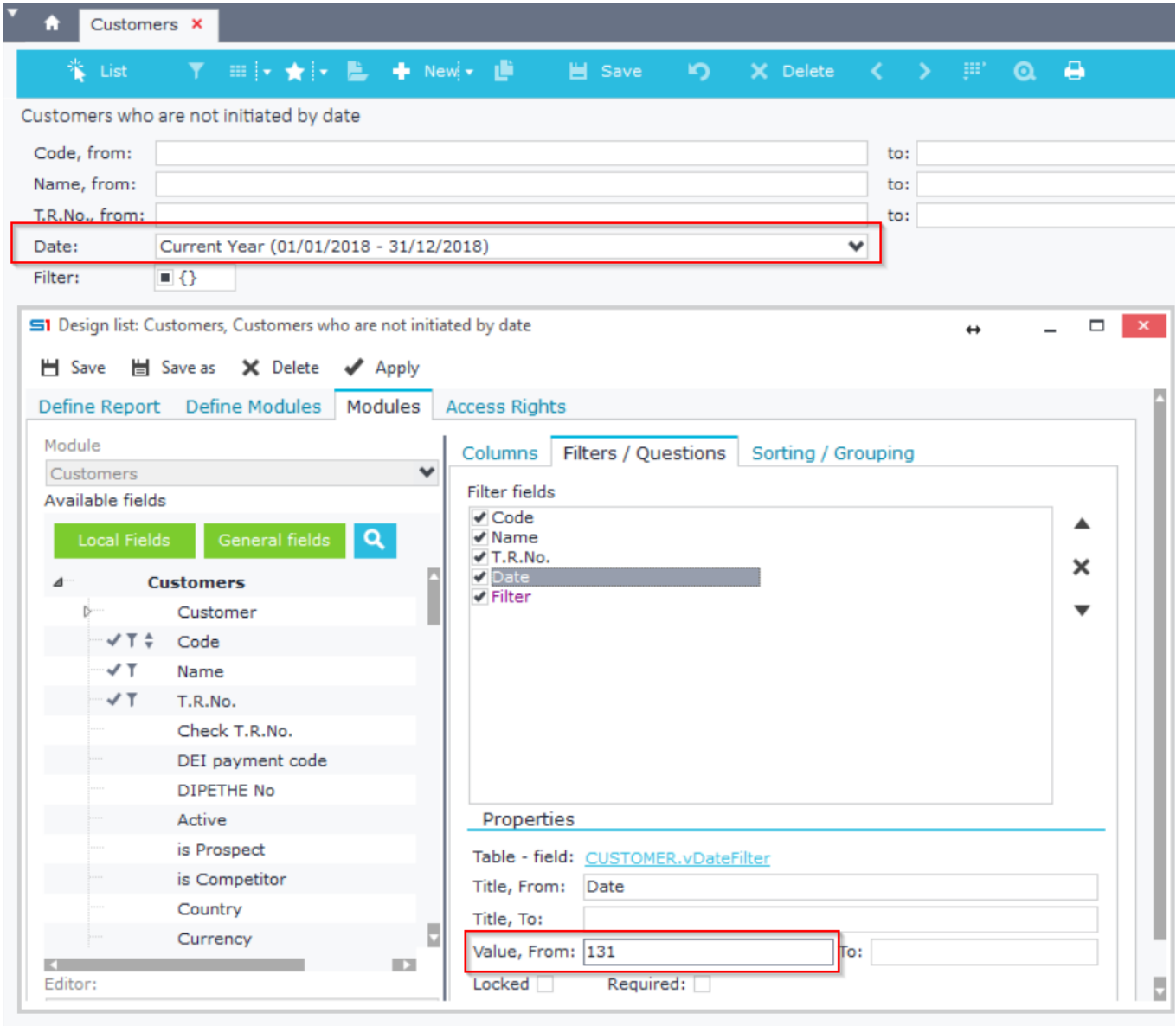


Figure B4.5

B.5 SQL Filter

This operation is used for filtering the records of reports or browsers. It creates a checkbox (Yes/No) in the filters of the browser/report and if enabled it adds a where clause in the main SQL statement. Example of this operation can be found in the previous section.

B.6 Single filter / Multiple filter

The operation types **“Single filter”** and **“Multiple filter”** are used for creating dialog filters with specific values that are selectable through combo box control.

At runtime, when users select a value, then the data are filtered according to the formula of the second column. Use the button (Figure B6.1) next to the textbox of the calculation formula to enter the values that will populate the combo box (Figure B6.2). The field will be displayed in dialog filters as in Figure B6.3.

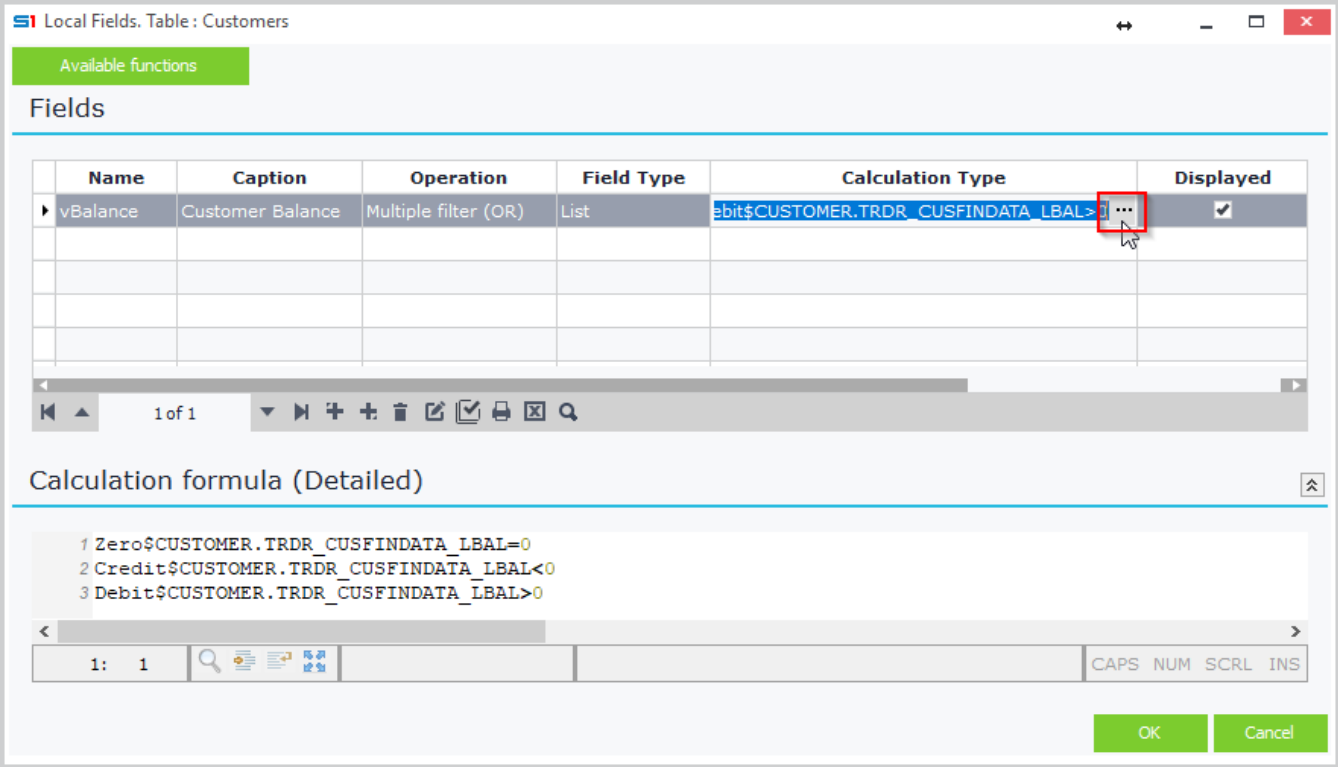


Figure B6.1

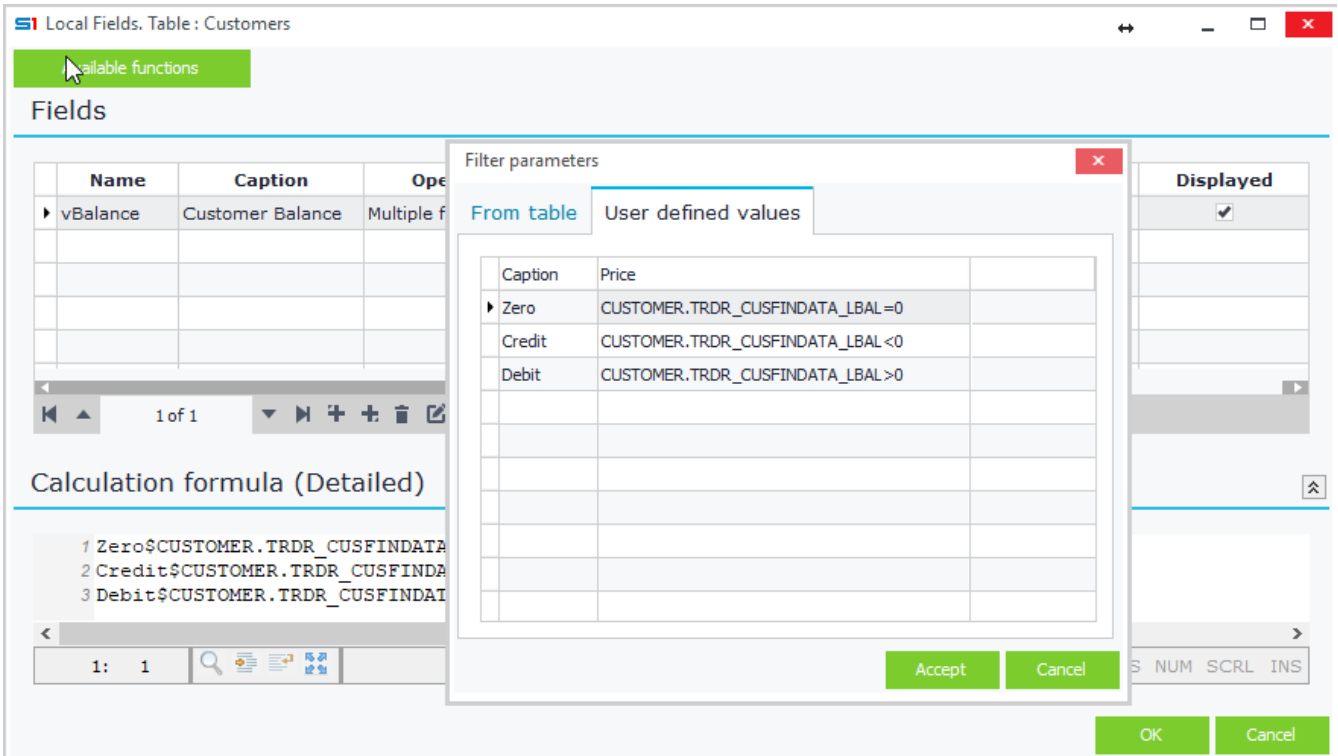


Figure B6.2

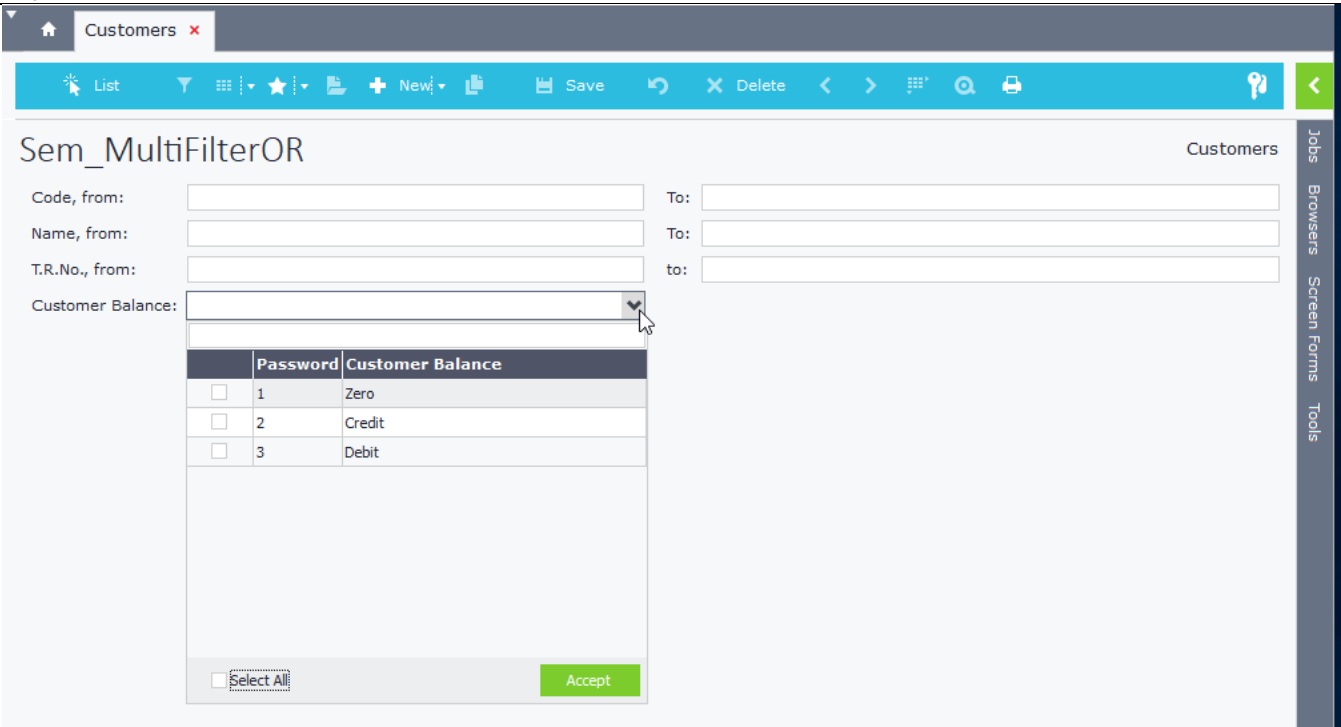


Figure B6.3

B.7 Function

This operation is used for creating custom functions that can be later used from other user defined fields. Note that fields created using this operation cannot be displayed as columns.

B.8 Multi Results SQL

This operation creates an OUTER APPLY SQL statement to the main query of the browser. This is very useful when you need to retrieve multiple column data from the same tables, because you don't have to write (and execute) the same SQL statement many times.

Suppose you need to display three columns (debit, credit and turnover) in a browser of Customers. One way to do it would be to add the fields from the **Customer details** node as in Figure B8.1.

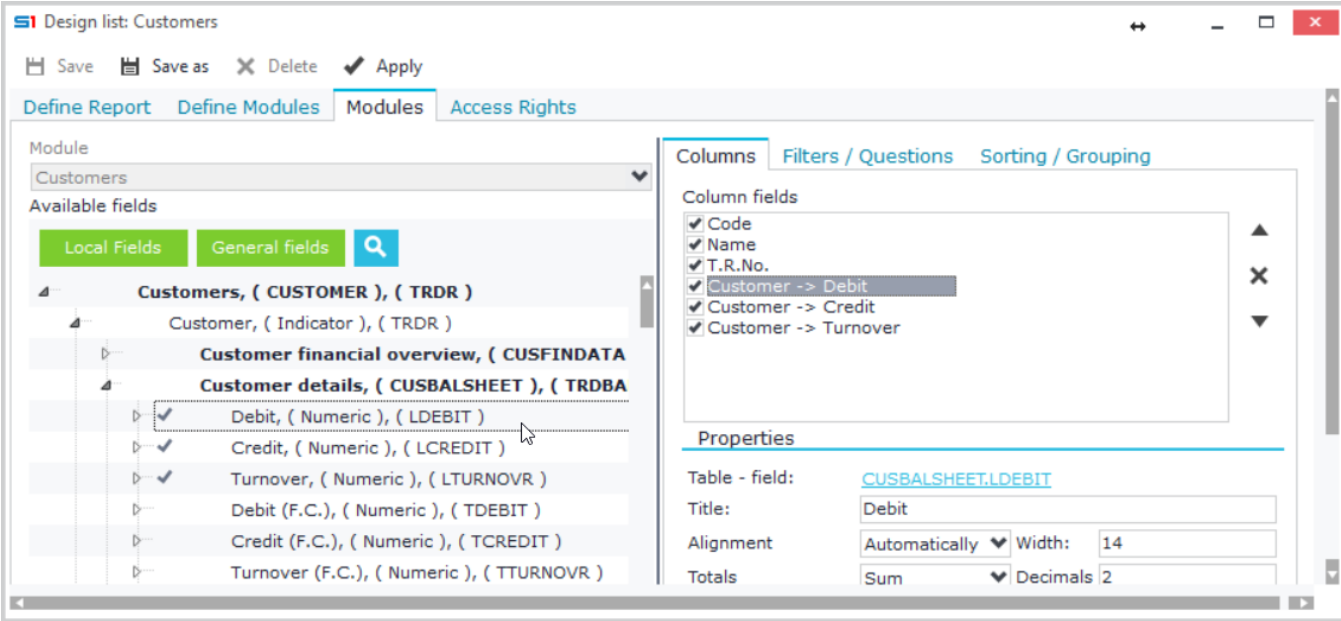


Figure B8.1

This creates three nested queries that select different columns (LDEBIT, LCREDIT, LTURNVR) from the same table (TRDBALSHEET) using also the same where clause, as you can see in the SQL Monitor of Figure B8.2. The results would be as in Figure B8.3.

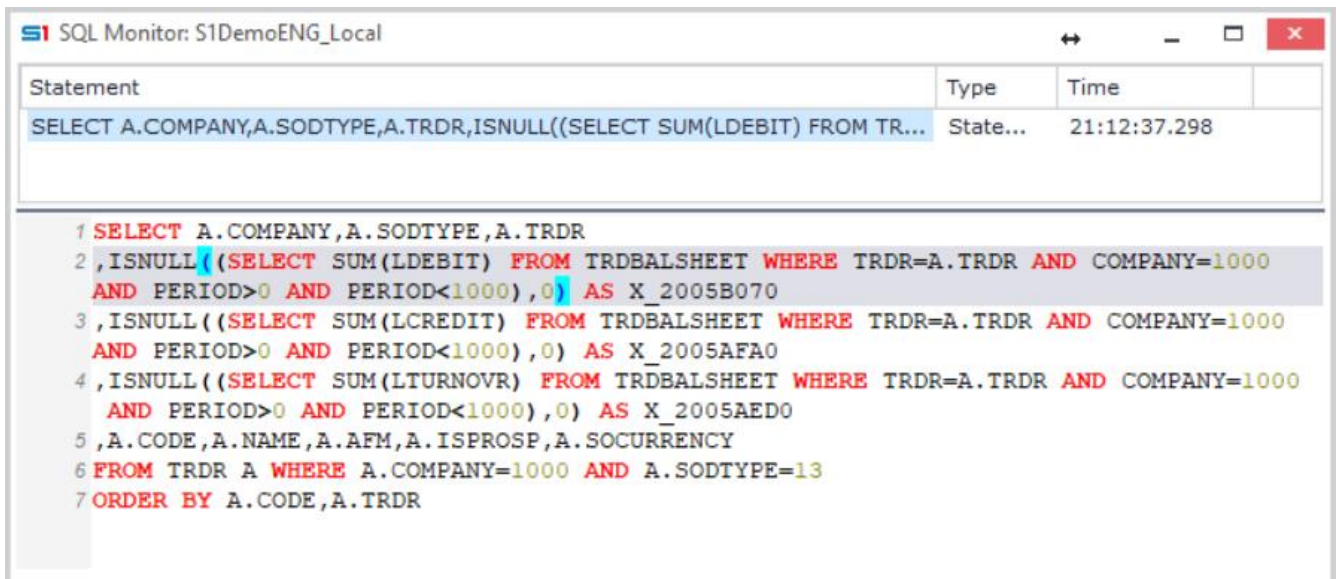


Figure B8.2

Customer Debit-Credit-TurnOver						
Code	Name	T.R.No.	Debit	Credit	Turnover	
101	Salesconsul GmbH	073586841	1.261.086,04	2.657,00	1.024.216,29	
102	Nick Panterben	23	211.895,06	152.396,80	172.028,50	
103	Holding AE	345345	13.436,29	18.494,15	10.598,60	
104	Oikoset AU		1.131,61		513,50	
105	Thunder Tiger ABEE		1.092,00	612,00	400,00	
106	Shipping SA	140398745	18.951,72	8.104,87	14.838,80	
107	SoftOne AE	999863881	1.410,08	12.343,00	496,00	
108	Kouskoukis George	111	900,00			
109	Patrick Theodore		1.000,00			
110	Dustin Standt		1.100,00			
111	Kelesidis Claus		1.200,00	122,00		
112	Apple AE		110.934,82	9.250,00	89.134,00	

Figure B8.3

So, it would be better if we could create an OUTER APPLY join with TRDBALSHEET and return all three columns with a single query. The outer apply operator always returns the data of the left join (TRDR) irrespectively of its match with our table, so there would be no problem with the data returned.

- Create a user-defined field that uses **Multi Results SQL** as operation, and write the following SQL statement:

```

SELECT
    1 AS VMULTIQUERY ,
    SUM(TB.LDEBIT) AS DEBIT ,
    SUM(TB.LCREDIT) AS CREDIT ,
    SUM(TB.LTURNVR) AS TURNOVER
FROM
    TRDBALSHEET TB
WHERE
    TB.TRDR = A.TRDR
AND
    COMPANY = :X.SYS.COMPANY
AND
    TB.PERIOD > 0 AND TB.PERIOD < 1000
  
```

- Create three user-defined fields that will be added as columns inside the browser. The names of the columns must be the same as the ones inside the Multi Results SQL statement (Debit, Credit, TurnOver). Use SQL command as Operation and Number as Field Type (Figure B8.4). Leave Calculation Type blank.

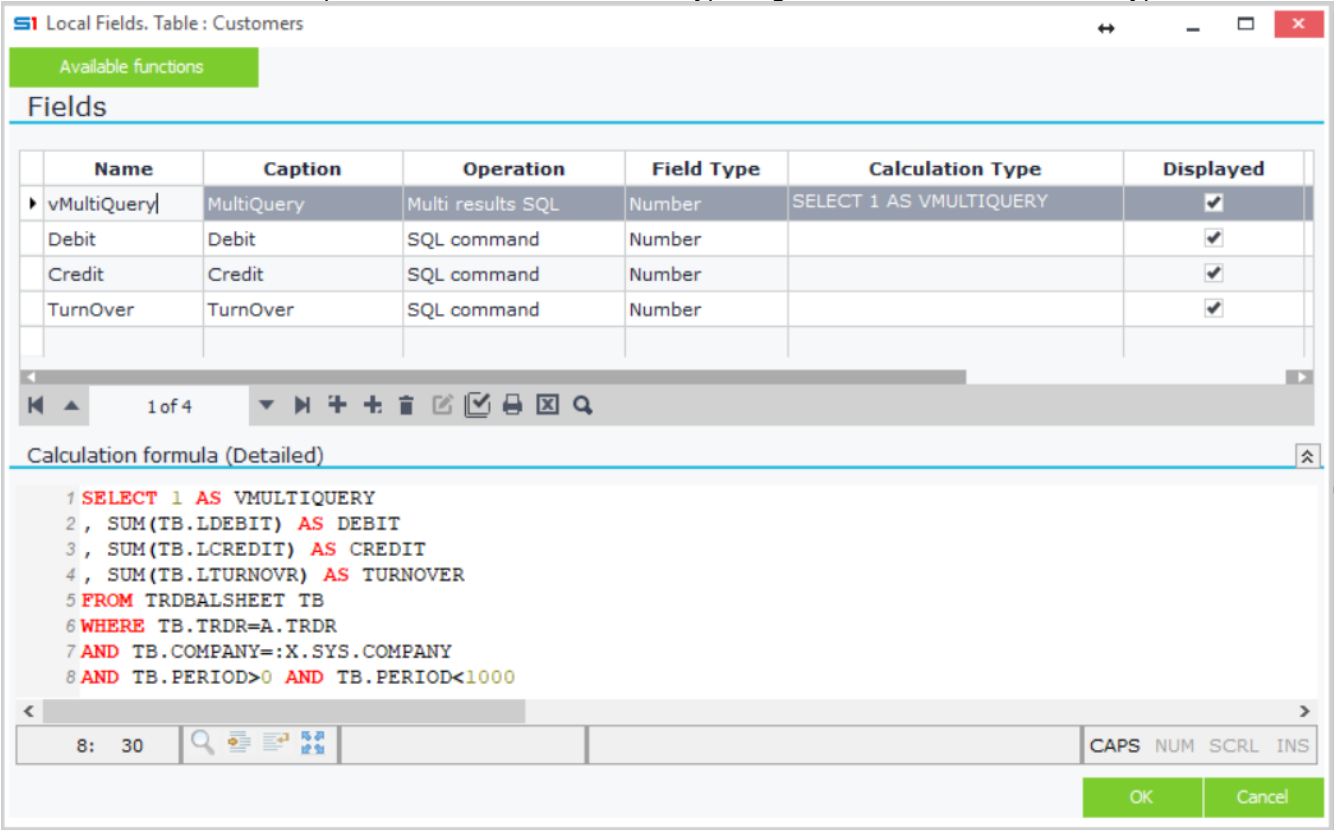


Figure B8.4

- Add all four fields inside the browser (Columns Tab).
- Uncheck the field that uses "Multi results SQL operation" (Figure B8.5). This column is needed, because it applies the OUTER APPLY join and is also added in SQL SELECT statement, so we need it just for reference (1 AS VMULTIQUERY), but it does not need to be visible.

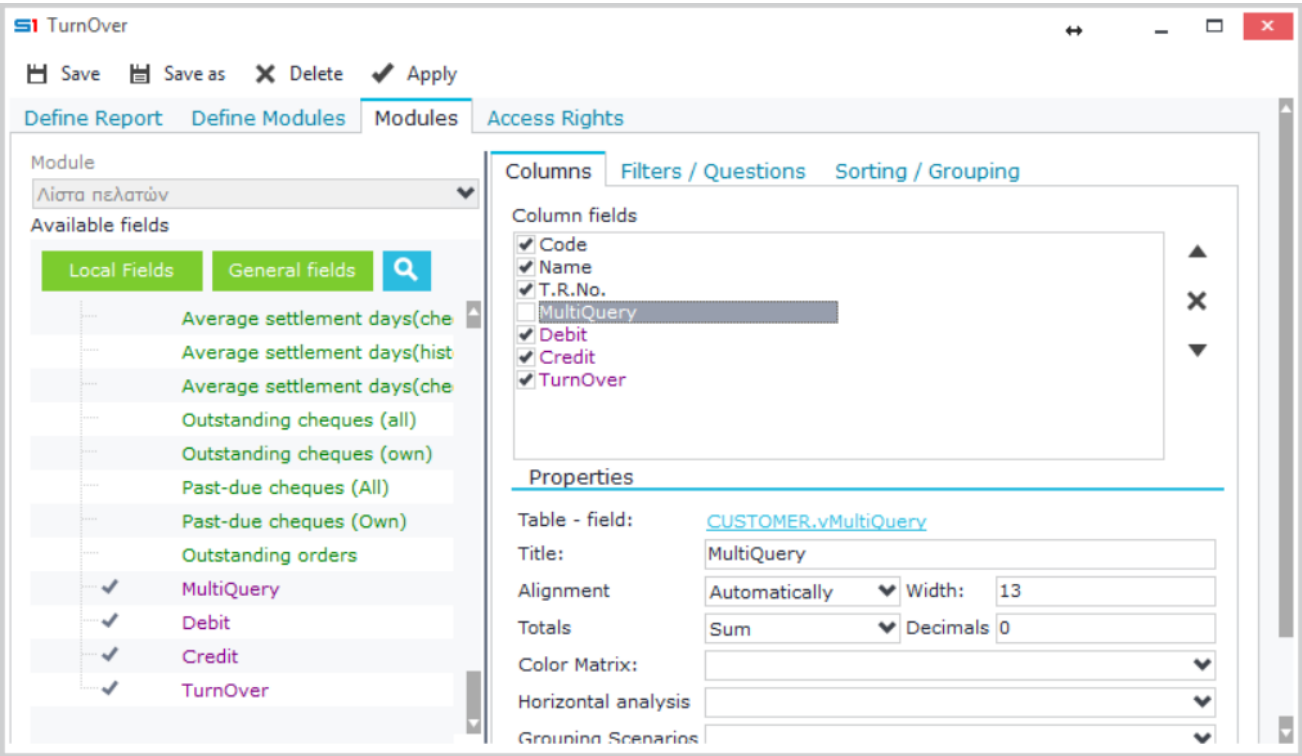


Figure B8.5

In Figure B8.6 (SQL Monitor) you can see the select statement with the OUTER APPLY join when browser is executed (Figure B8.7). The results are the same as in Figure B8.3.

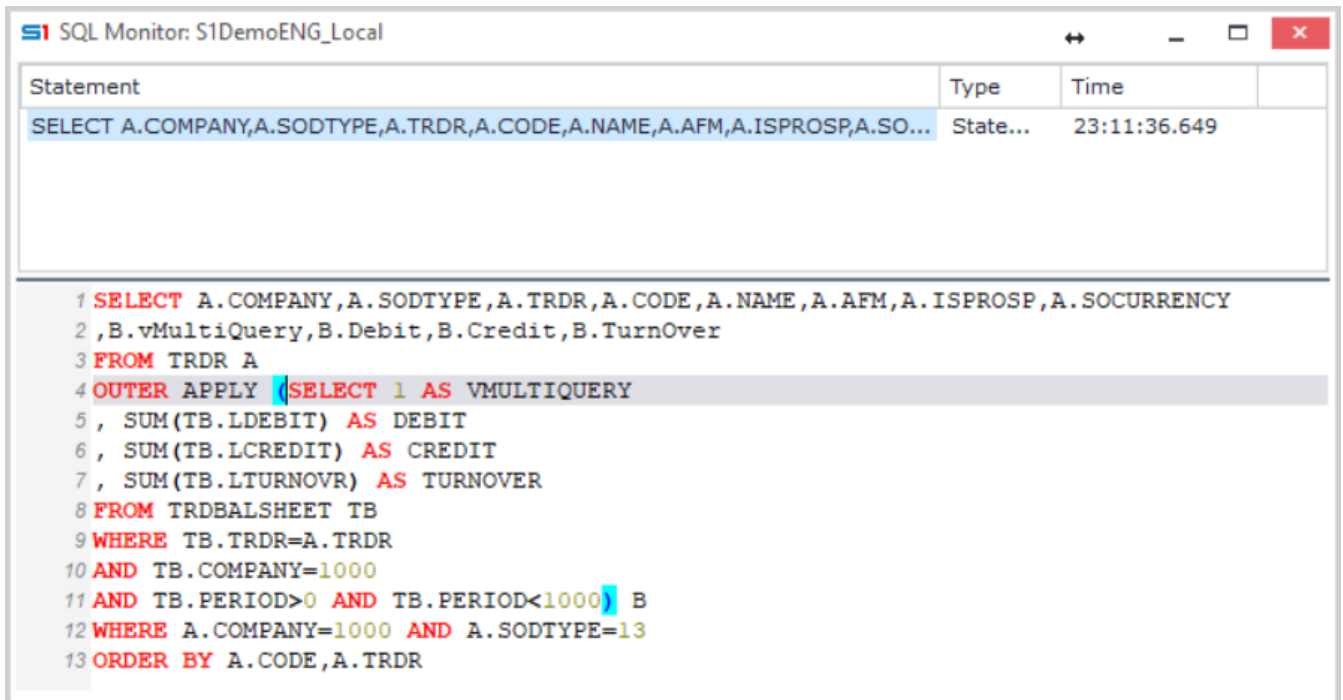


Figure B8.6

The image shows a screenshot of the 'Customers' window in a database application. It displays a table with columns 'Code', 'Name', 'T.R.No.', 'Debit', 'Credit', and 'TurnOver'. The table contains 12 rows of data.

Code	Name	T.R.No.	Debit	Credit	TurnOver
101	Salesconsul GmbH	073586841	1.261.086,04	2.657,00	1.024.216,29
102	Nick Panterben	23	211.895,06	152.396,80	172.028,50
103	Holding AE	345345	13.436,29	18.494,15	10.598,60
104	Oikoset AU		1.131,61	0,00	513,50
105	Thunder Tiger ABEE		1.092,00	612,00	400,00
106	Shipping SA	140398745	18.951,72	8.104,87	14.838,80
107	SoftOne AE	999863881	1.410,08	12.343,00	496,00
108	Kouskoukis George	111	900,00	0,00	0,00
109	Patrick Theodore		1.000,00	0,00	0,00
110	Dustin Standt		1.100,00	0,00	0,00
111	Kelesidis Claus		1.200,00	122,00	0,00
112	Apple AE		110.934,82	9.250,00	89.134,00

Figure B8.7

C. Built-in Functions

Predefined functions are available in user defined fields and are very useful for advanced calculations. Select them through the window that is displayed by clicking on the button **“Available functions”** (Figure C1). Syntax, parameters, info and examples of the functions are displayed on the right-hand side of the window. Basic functions and examples of them are discussed below.

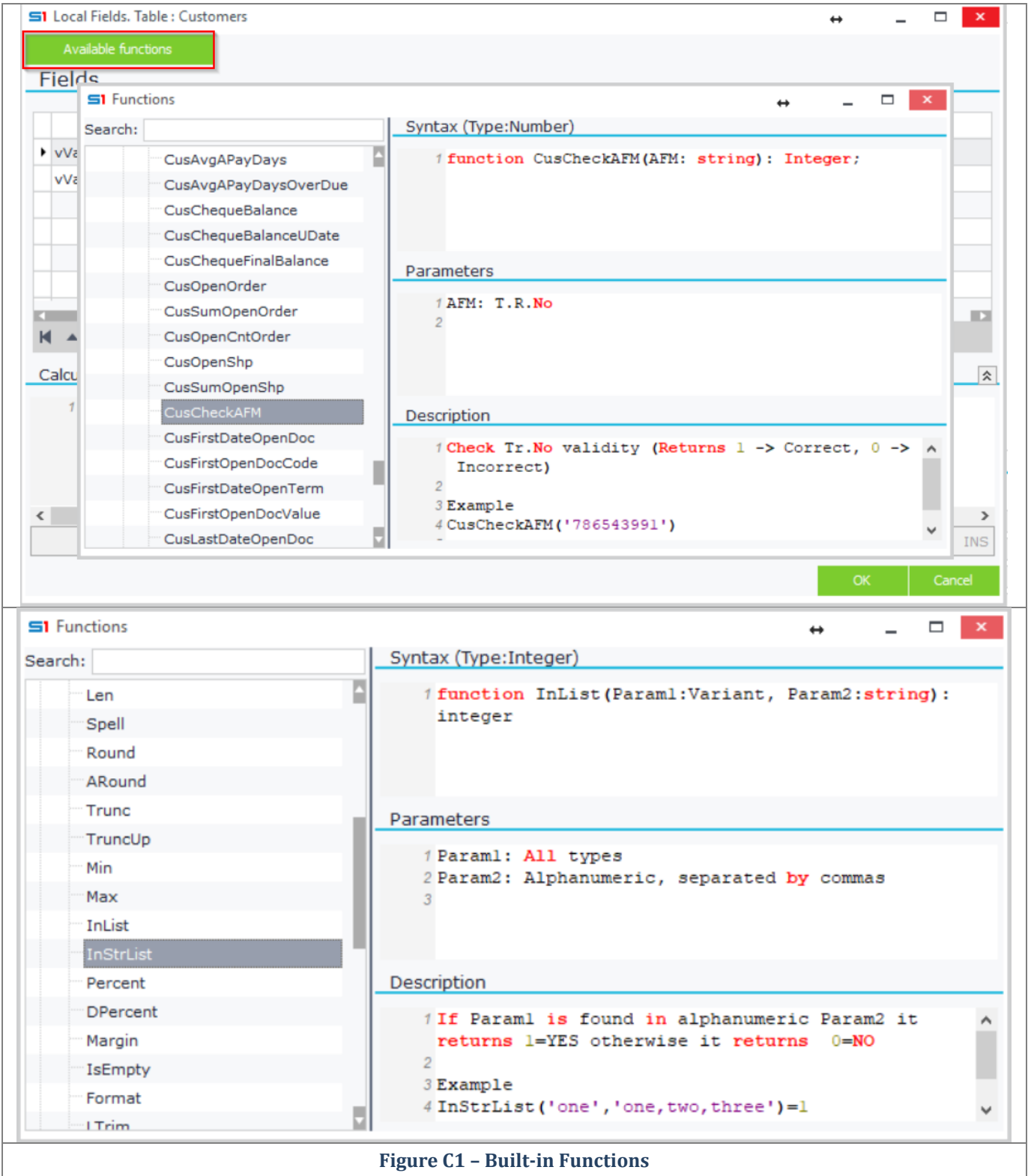


Figure C1 – Built-in Functions

Abs (Param1: double): double

Returns the absolute number defined in Param1.

Example (Browser)

Defined field in a Sales statistics browser that displays the absolute number of the expression: (Sales value + Vat Value) / Abs (Sales value). The expression will be executed only if the sales quantity is larger than zero.

Create a new defined field using the operation "Line calculation" and "Number" as Field type.

Enter the following formula expression (Figure C2):

```
IF (VSALSTAT.SalQty1<>0
, (ABS (VSALSTAT.SalVal+VSALSTAT.SalVatAmnt) ) /VSALSTAT.SalQty1
,0)
```

This expression uses also the built-in function if for the validation check.

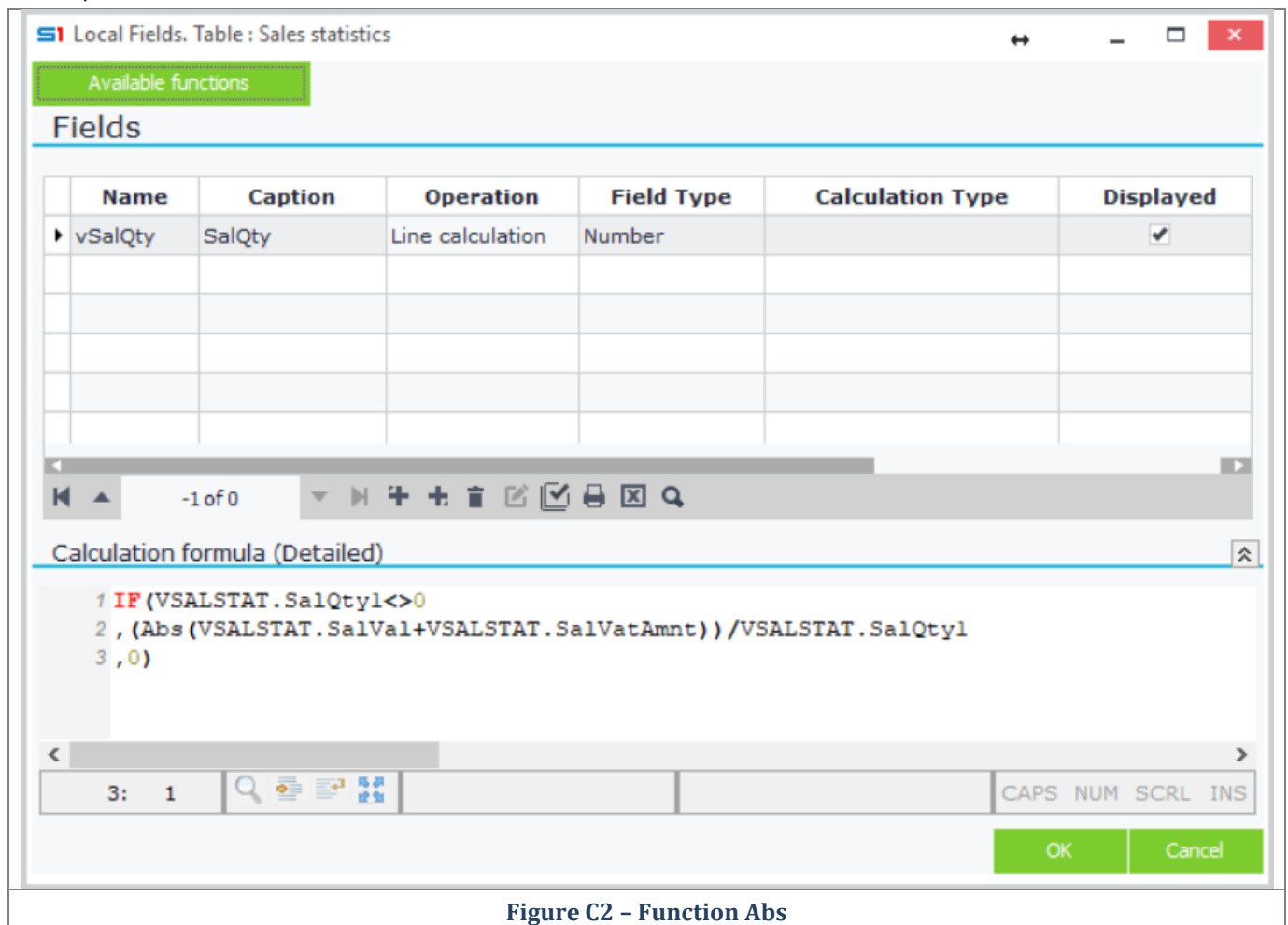


Figure C2 – Function Abs

GetConvertSeries (SOSOURCE: number, SERIES: number): string

Returns the conversion series split by semicolon (of the given sosource and series).

Example

```
GetConvertSeries(1351, 7001)//returns 7021;7721;7041
```


FormatDateTime (Param1: string, Param2: Date): String

Returns the date of the Param2 in Param1 format.

Example (Browser)

Defined field in Sales statistics browser that displays the insert time of sales documents, in a specific format (hours, minutes, seconds).

Create a new defined field, set the Operation to "Line calculation", Field type to "Number" and enter the following formula (Figure C3):

FormatDateTime ('hh:mm:ss', VSALSTAT.SoTime)

This formula returns the date of the field in "hh:mm:ss" format.

The figure consists of two screenshots from a software application. The top screenshot shows a dialog box titled "Local Fields. Table : Sales statistics". It has a tab "Available functions" and a section "Fields" containing a table. The table has columns: Name, Caption, Operation, Field Type, Calculation Type, and Displayed. A row is added with Name "vTime", Caption "Time", Operation "Line calculation", Field Type "Alphanumeric", Calculation Type "FormatDateTime(''hh:mm:ss'',VSALSTAT.SoTime)", and Displayed checked. Below the table is a "Calculation formula (Detailed)" section showing the formula "FormatDateTime (''hh:mm:ss'',VSALSTAT.SoTime)". The bottom screenshot shows the "Sales Statistics" browser window. It has a toolbar with icons for List, Filter, Star, Delete, and others. The main area displays a table with columns: Date, Time, Code, Name, Code, and Descr. The 'Time' column displays the formatted time values for each row, such as "12:44:44" and "11:43:51".

Figure C3 – Function FormatDateTime

SetValue (Param1: string, Param2: variant): integer

This function is usually used in EDA (Alerts) for setting a value (Param2) to a field (Param1). Note that the first parameter (Param1) is string, which means that fields have to be inserted in single quotes.

Example (EDA)

EDA in Customers that sets to the value of field Code1 the value of field Code, adding zeroes as a prefix in such a way that the whole length of the field is no longer than 15 characters.

Click on the EDA tool of the customers object to create a new EDA.

- Add a **Module rule** and enable the event «Before DB Insert».
- Click on the toolbar button **"Fields"** and then on **Local fields** (available through the node <Event fields>).
- Create a new user-defined field, set the operation to "Calculate", the field type to "Alphanumeric" and enter the formula (Figure C4) : `LPad(CUSTOMER.CODE1,15,'0')`

Return to the main screen of the EDA and create a new Formula fact using "Run" as Action, "Function" as Type and enter the formula (Figure C5): `SETVALUE('CUSTOMER.CODE',EVTCUSTOMER.vAddZeros)`

You may drag and drop the field **EVTCUSTOMER.vAddZeros** instead of typing it.

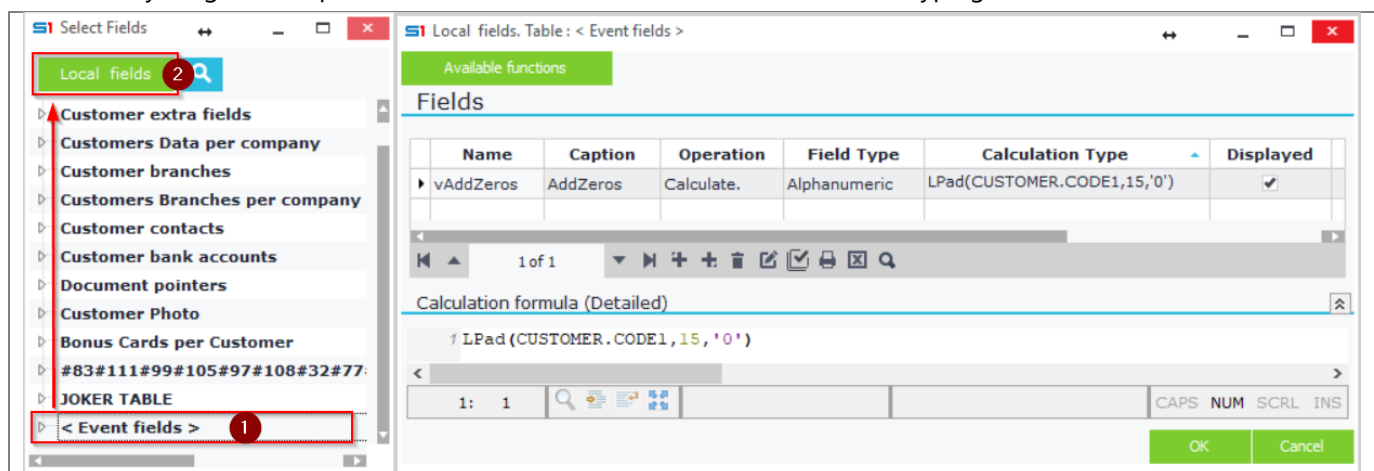


Figure C4 – Function Lpad

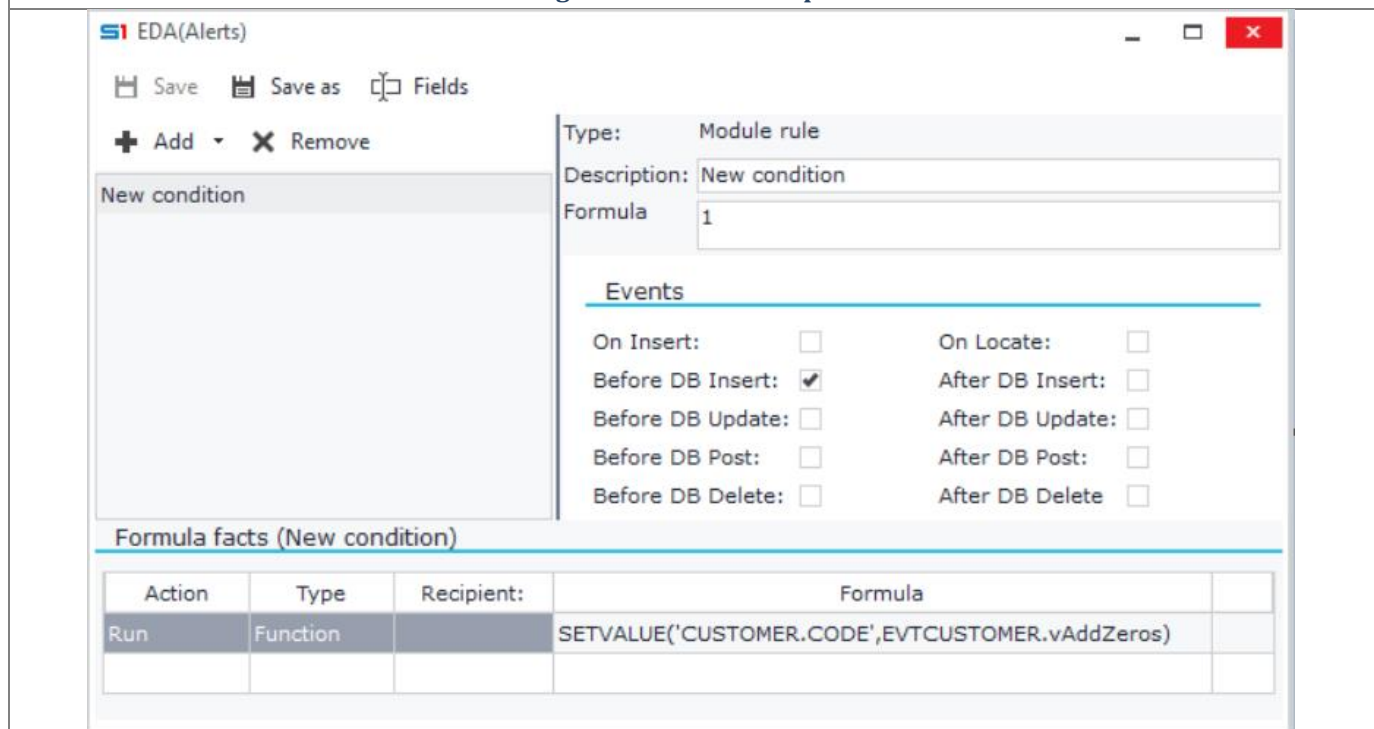


Figure C5 – Function SetValue

RunSQL (Param1: string): string

Executes the SQL statement defined in Param1 and returns the results. This function is used for retrieving data from database objects. Use this function wisely, because for every record returned it executes the custom query (defined in Param1) and this can cause slow performance.

Example 1 (Browser)

Defined field in a Sales statistics browser that displays the field FINCODE of the “converted from” document that is linked to the Sales document displayed in each record of the browser.

Create a defined field, select “Line calculation” as operation, “Alphanumeric” as field type and enter the formula: RunSql ('select f.fincod from findoc f,mtrlines m where m.findocs=f.findoc and m.fincod = '+STRING (VSALSTAT.Findoc)) (Figure C6)

Note that the formula uses the function **String** for converting the FINDOC id of each record (VSALSTAT.FINDOC) to string format.

Note: If you needed to add a reference to a varchar field then you would have to use the **QUOTESTR** function. Suppose that you need to filter data using field FINCODE (type varchar), then the sql statement should be:

```
RunSql ('select f.fincod
from findoc f,mtrlines m
where m.findocs=f.findoc
and m.fincod='+QUOTESTR (VSALSTAT.FINCODE) + '
and m.findoc='+STRING (VSALSTAT.Findoc) )
```

Figure C7 displays the sql statement in SQL Monitor.

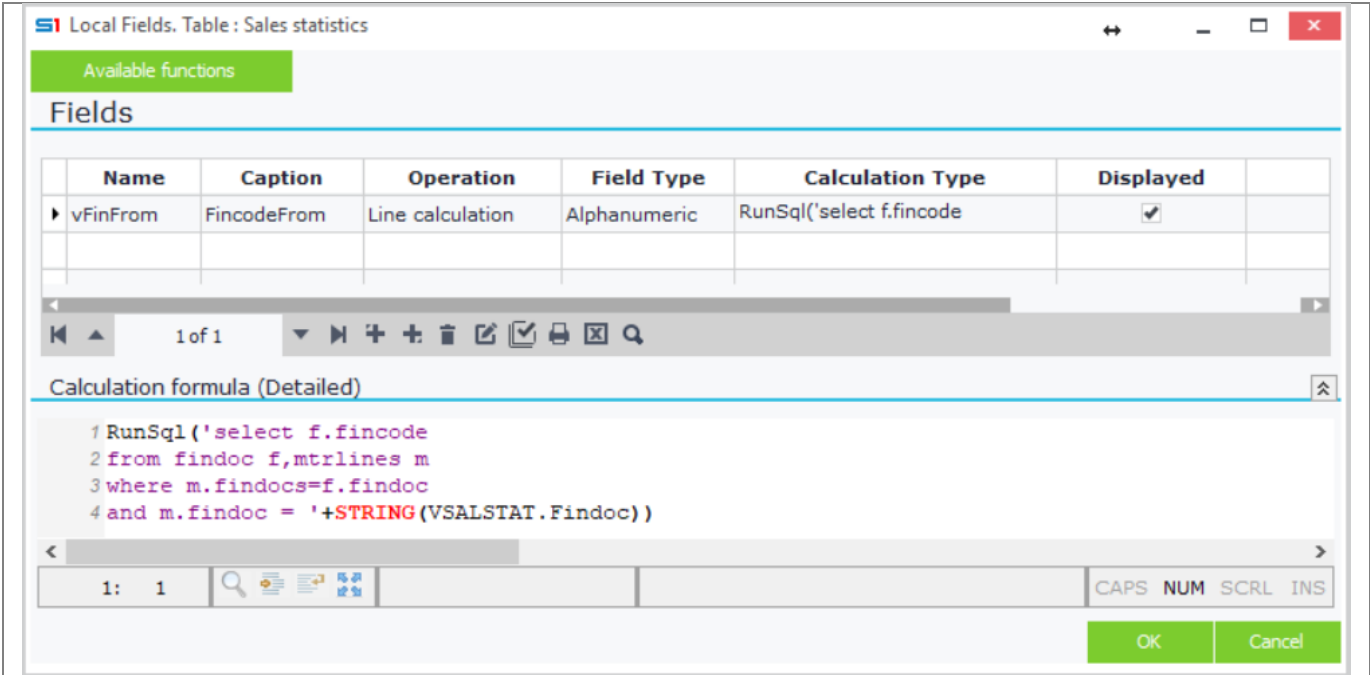


Figure C6 – Defined field using RUNSQL function

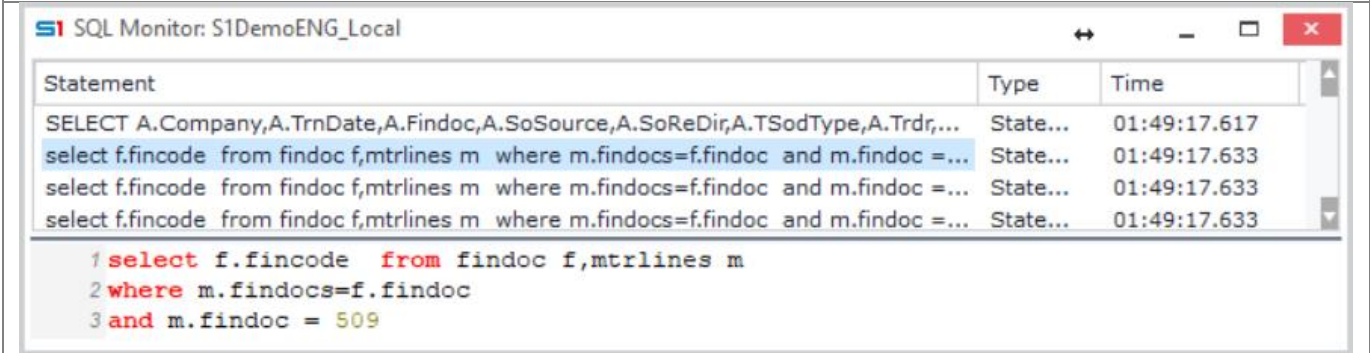


Figure C7 – SQL Monitor

Example 2 (EDA-Alert)

User-defined field in Sales documents EDA that returns Company T.R. No.

This can be implemented either using the RUNSQL function or using the TABLE function (Figure C8). The SQL statement produced is the same, the only thing that is different is the way you can use the system parameter Login Company.

The first defined field uses the value of the field SALDOC.COMPANY. The formula is:

RUNSQL('SELECT AFM FROM COMPANY WHERE COMPANY='+STRING(SALDOC.COMPANY))

The second defined field uses the X.SYS.COMPANY system parameter, prefixed with a colon, that retrieves the login company. The formula is:

RUNSQL('SELECT AFM FROM COMPANY WHERE COMPANY=:X.SYS.COMPANY')

The third defined field retrieves the login company using the internal parameter CompanyCode, that is available through "Available functions" button. The formula is:

RUNSQL('SELECT AFM FROM COMPANY WHERE COMPANY='+STRING(CompanyCode))

Figure C9 displays the SQL queries that will be executed at runtime.

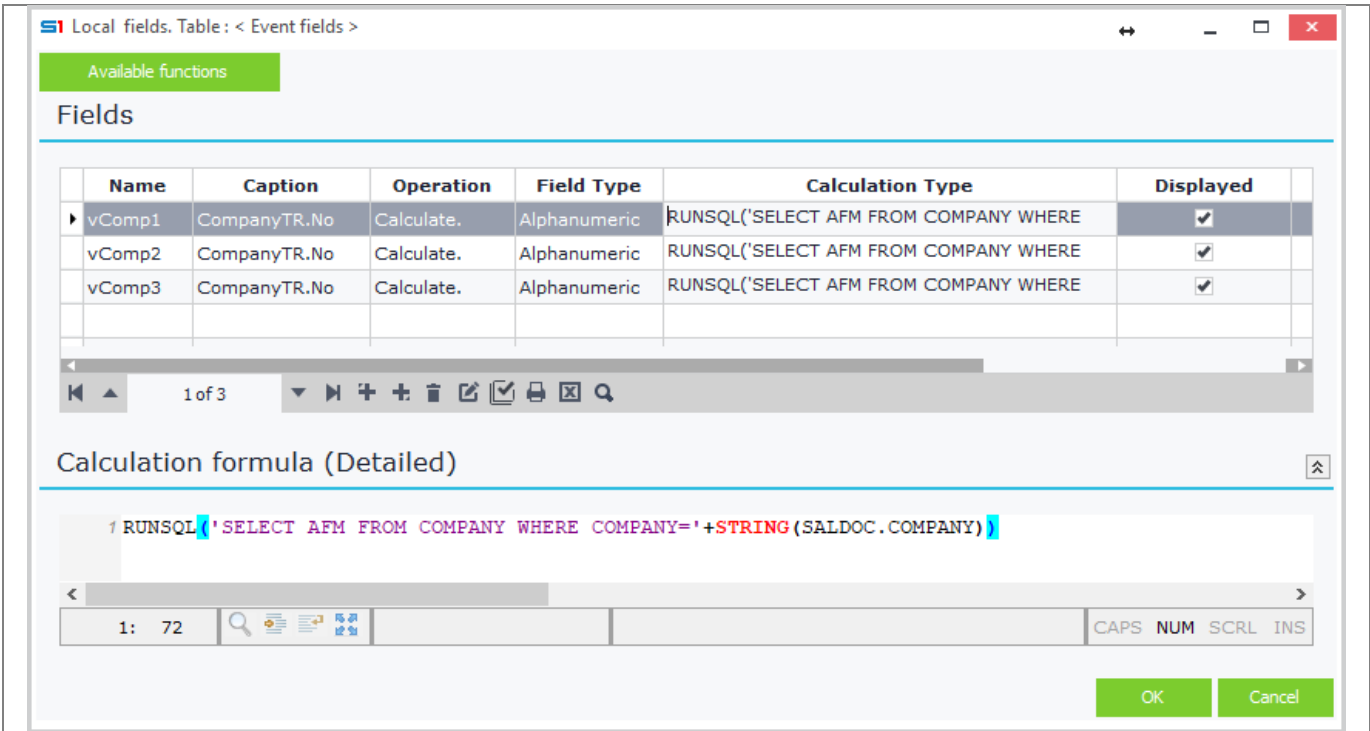


Figure C8

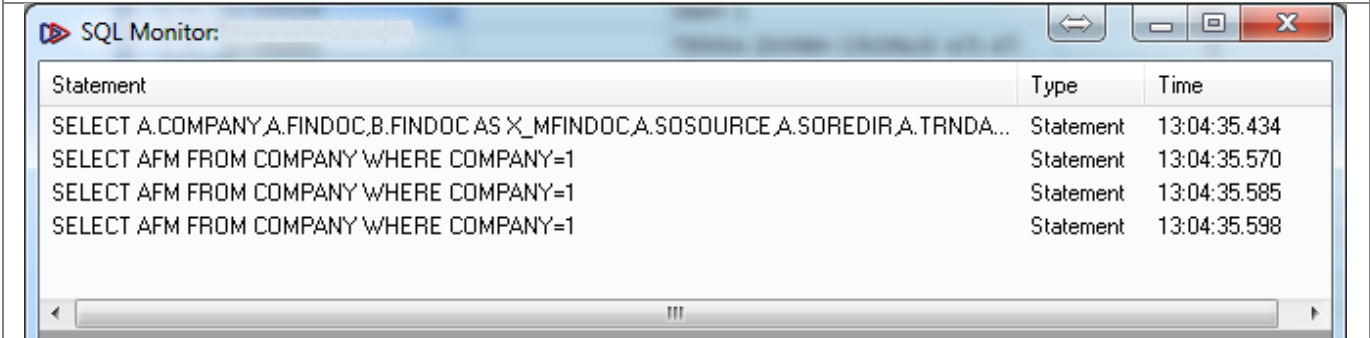


Figure C9 – SQL Monitor

Copy (Param1: string, Param2, Param3: integer): string

Returns the number of characters defined in Param3 for the string Param1, starting from Param2 position.

Example (Printout Form)

Defined fields in sales documents printout form that display the field Comments of Items in the recurrent section of the form. The text will wrap every 20 characters.

Create the two defined fields inside the "Material Lines" (table MTRLINES) as in Figure C10. The defined field **vLine1** will display the first 20 characters of the field MTRL.COMMENTS using the formula:

`COPY(MTRLINES.MTRL_MTRL_REMARKS,1,20)`

The defined field V2 will be used to display all the rest text of the field MTRL.COMMENTS using the formula:

`COPY(MTRLINES.MTRL_MTRL_REMARKS,21,1000)`

Field wrap is achieved through the \$ sign. Insert the field **vLine1** in the recurrent section of the form and then enter the **vLine2** field, in the properties, using the \$ sign (Figure C11). Set the width to 20 characters.

At runtime the form will be printed as in Figure C12, the comments of the first line are wrapped in four lines.

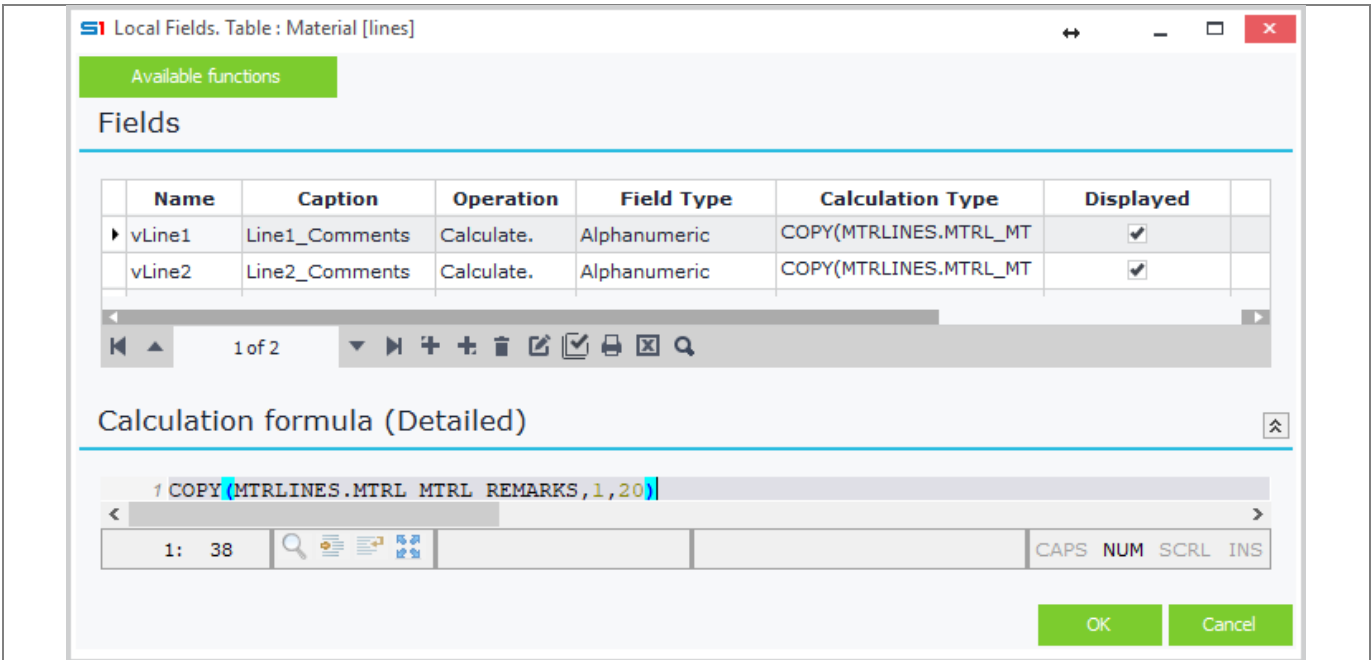


Figure C10 –Function Copy

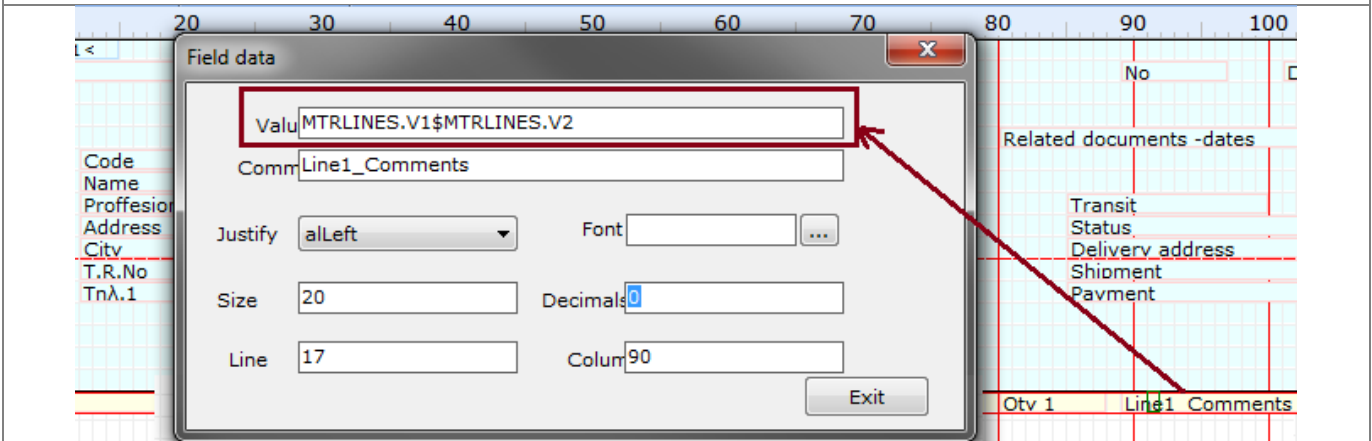


Figure C11

10001	TV LCD 32	1,00	FIRST 20 CHARACTERS
			21 ...40 CHARACTER
			41 ... 60 CHARACTER
			61 ... 80 CHARACTER
10002	TV LCD 21	1,00	
10003	Digital Camera 8 MP	1,00	

Figure C12

6. DATABASE DESIGNER

A. [General Features](#)

B. [Fields](#)

C. [Tables](#)

D. [String Lists](#)

E. [Database Views](#)

F. [Objects](#)

G. [Virtual Tables](#)

H. [Report Objects](#)

A. General Features

SoftOne provides a Database Management tool that allows you to customize the structure of the database and also create new Business Objects. This means that you are allowed to create new fields in SoftOne tables, create your own custom tables, database views and new custom Business Objects, virtual tables, string lists and scripts.

Customizations are maintained during future upgrades of SoftOne.

Database is managed through the **S1 Designer** tool, which is inside the “**S.D.K. – Customization tools**” folder under “**Tools**” section menu. Each record of S1 Designer has its own name, description, versioning and of course custom fields, tables, objects etc. Figure A1 shows an example of different designer records.

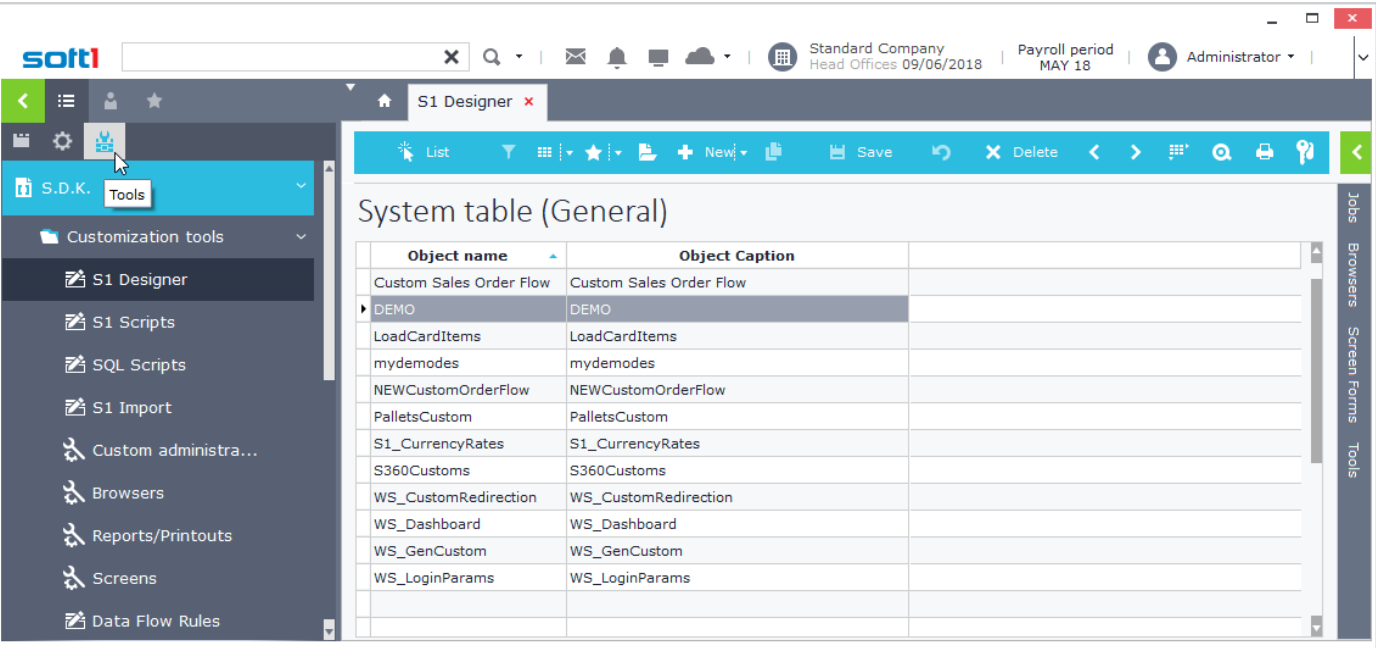


Figure A1 –SoftOne Designer

Designer entries can be merged in one designer record (project) by inheriting others (Used Projects). The option of creating many design projects can be very helpful when deploying customer projects that involve different implementors. At the same time, they can be also used to generate a single custom solution for exploitation (Figure A2).

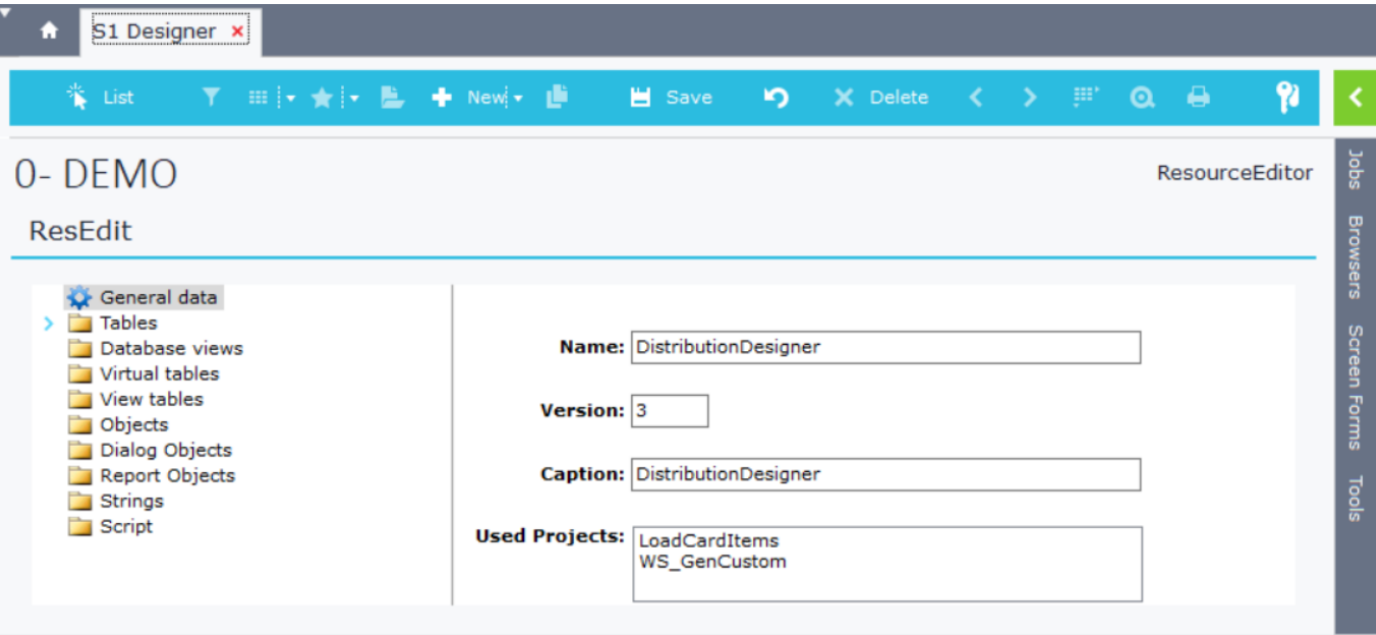


Figure A2 - Designer with Used Projects

A1. Synchronization

SoftOne designer uses an internal sync tool that synchronizes the database whenever you make changes that affect the database structure (new fields, tables, views, constraints). Synchronization takes place upon **manual change of the version** of any designer record (Figure A3).

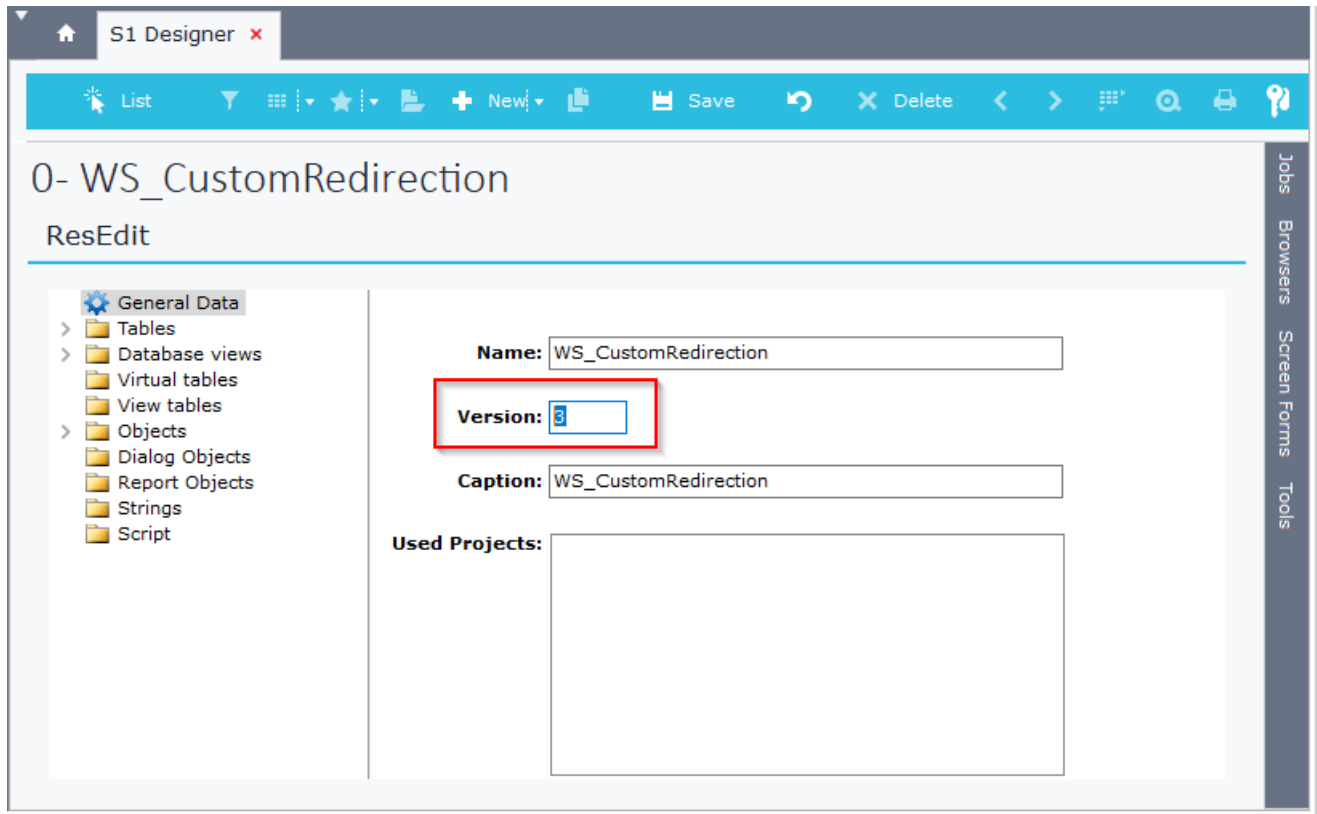


Figure A3 – Designer version change

Any time you change the version of a SoftOne designer record you will be requested to synchronize upon your next login to the application. The different versions of the design projects will be displayed in a dialog message asking you to accept the synchronization. In the example of Figure A4 the sync occurred because the version of the custom designer "WS_CustomRedirection" has changed from 2 to 3.

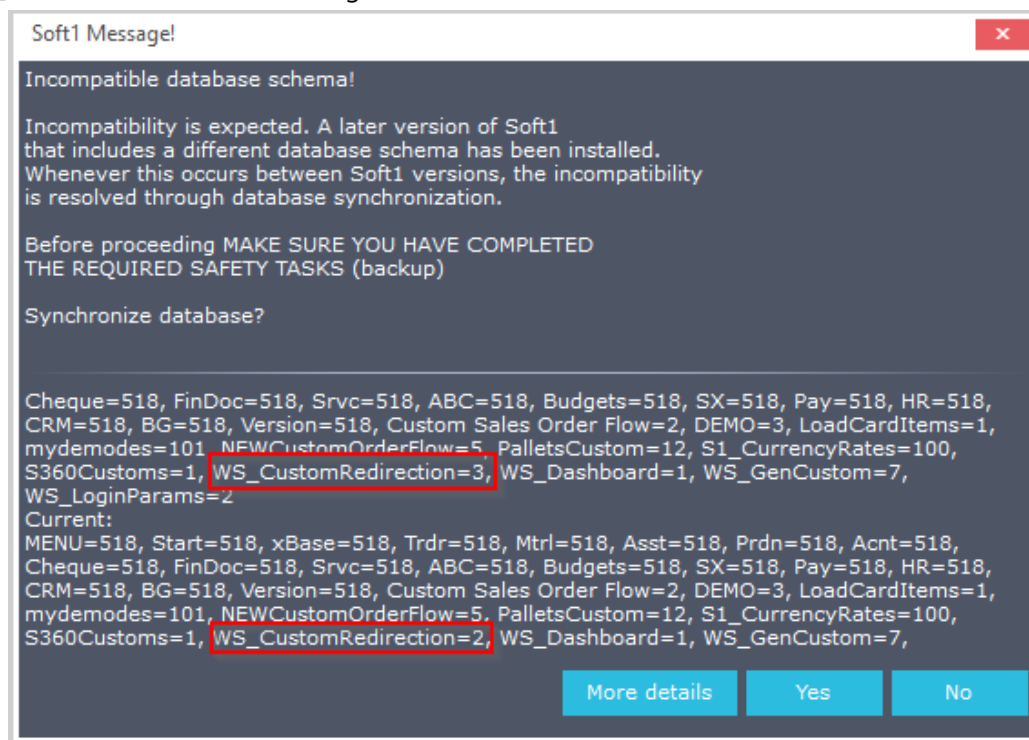


Figure A4 - Sync database message box

Chapter 6 – Database Designer

The versions of all the database designer records are saved as delimited string inside the field **CSTGEN** of the table **CSTINFO** for **CSTTYPE=98** (Figure A5).

```
SELECT CSTGEN FROM CSTINFO WHERE CSTTYPE=98
```

```
MENU=518;Start=518;xBase=518;Trdr=518;Mtrl=518;Asst=518;Prdn=518;Acnt=518;Cheque=518;
FinDoc=518;Srv=518;ABC=518;Budgets=518;SX=518;Pay=518;HR=518;CRM=518;BG=518;
Version=518;
Custom Sales Order Flow=2;DEMO=3;LoadCardItems=1;mydemodes=101;
NEWCustomOrderFlow=5;PalletsCustom=12;S1_CurrencyRates=100;S360Customs=1;
WS_CustomRedirection=2;WS_Dashboard=1;WS_GenCustom=4;WS_LoginParams=2
```

Database explorer

1: 44

SELECT CSTGEN FROM CSTINFO WHERE CSTTYPE=98

CSTGEN
MENU=518;Start=518;xBase=518;Trdr=518;Mtrl=518;Asst=518;Prdn=518;Acnt=518;Cheque=518;FinDoc=518;Srv=518;ABC=518;Budgets=518;SX=518;Pay=518;HR=518;CRM=518;BG=518;Version=518;Custom Sales Order Flow=2;DEMO=3;LoadCardItems=1;mydemodes=101;NEWCustomOrderFlow=5;PalletsCustom=12;S1_CurrencyRates=100;S360Customs=1;WS_CustomRedirection=2;WS_Dashboard=1;WS_GenCustom=4;WS_LoginParams=2

1 of 1

Success. Execution Time 0 ms

Figure A5 – Custom designer records

Each custom designer structure is saved in the field **SODATA** of the table **CSTINFO** for **CSTTYPE=97** (Figure A6).

```
SELECT * FROM CSTINFO WHERE CSTTYPE=97
```

Database explorer

1: 39

SELECT * FROM CSTINFO WHERE CSTTYPE=97

CSTINFO	CSTTYPE	CSTNAME	NAME	SODEFAULT	SODATA	UPDUSERNAME	SOUPDDATE	CSTGEN
0	97	Custom Sales Order Flow	Custom Sales Order Flow			Administrator	21/12/2017 14:03:56	
0	97	DEMO	DEMO			developer	27/04/2016 11:32:43	
0	97	LoadCardItems	LoadCardItems	0		developer	21/02/2013 03:17:47	
0	97	mydemodes	mydemodes			Administrator	05/05/2018 11:30:25	
0	97	NEWCustomOrderFlow	NEWCustomOrderFlow			demo user	16/12/2016 11:02:45	
0	97	PalletsCustom	PalletsCustom	0		developer	06/05/2016 16:04:23	
0	97	S1_CurrencyRates	S1_CurrencyRates	0		developer	17/03/2014 09:57:50	
0	97	S360Customs	S360Customs			developer	13/06/2016 11:52:35	
0	97	WS_CustomRedirection	WS_CustomRedirection	0		Administrator	09/06/2018 23:24:11	
0	97	WS_Dashboard	WS_Dashboard	0		Administrator	11/05/2018 21:54:58	
0	97	WS_GenCustom	WS_GenCustom	0		Administrator	19/05/2018 15:35:15	
0	97	WS_LoginParams	WS_LoginParams	0		Administrator	04/06/2018 11:39:30	

1 of 12

Success. Execution Time 0 ms

Figure A5 – Custom designer records

Note: Version numbers of new database designers must be greater than the older ones or else the synchronization will fail and you will not be able to login SoftOne. In this case you need to manually update (through SSMS) the value of the field CSTGEN (CSTTYPE=98) with a version number less than the number of the new designer.

The synchronization file is saved inside the «**Log**» folder of the "**Program Profile Directory**". Its path can be found inside the window "**About**" (User – About) (Figure A7). The filename of the log file uses the same name as the connection file XCO, stating also the execution time of the synchronization (e.g. " S1DemoENG_20180505-160924.log"). This file contains all the sync jobs that took place and the possible errors that prevented the creation of tables, database views, or constraints in the database (Figure A8).

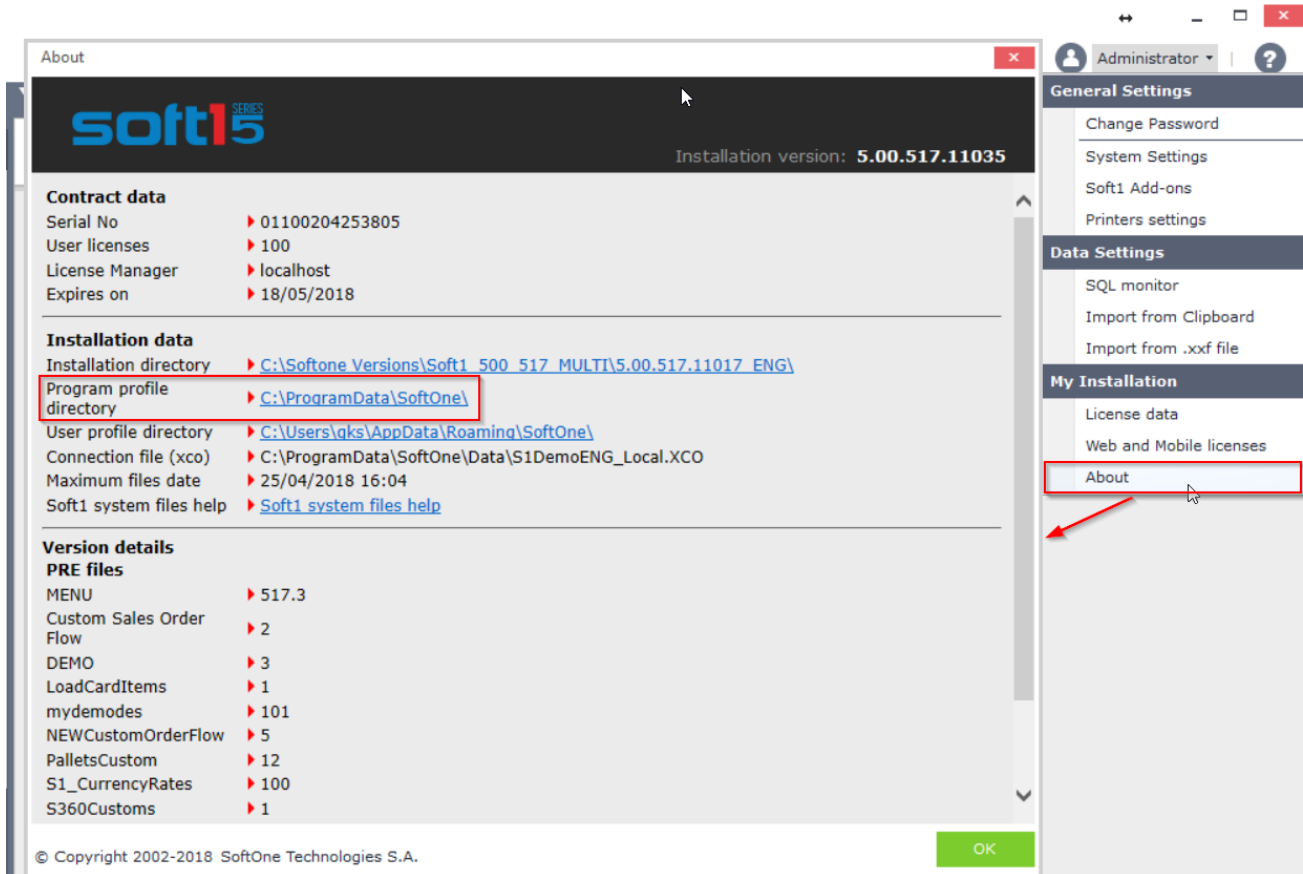


Figure A7 – About Window

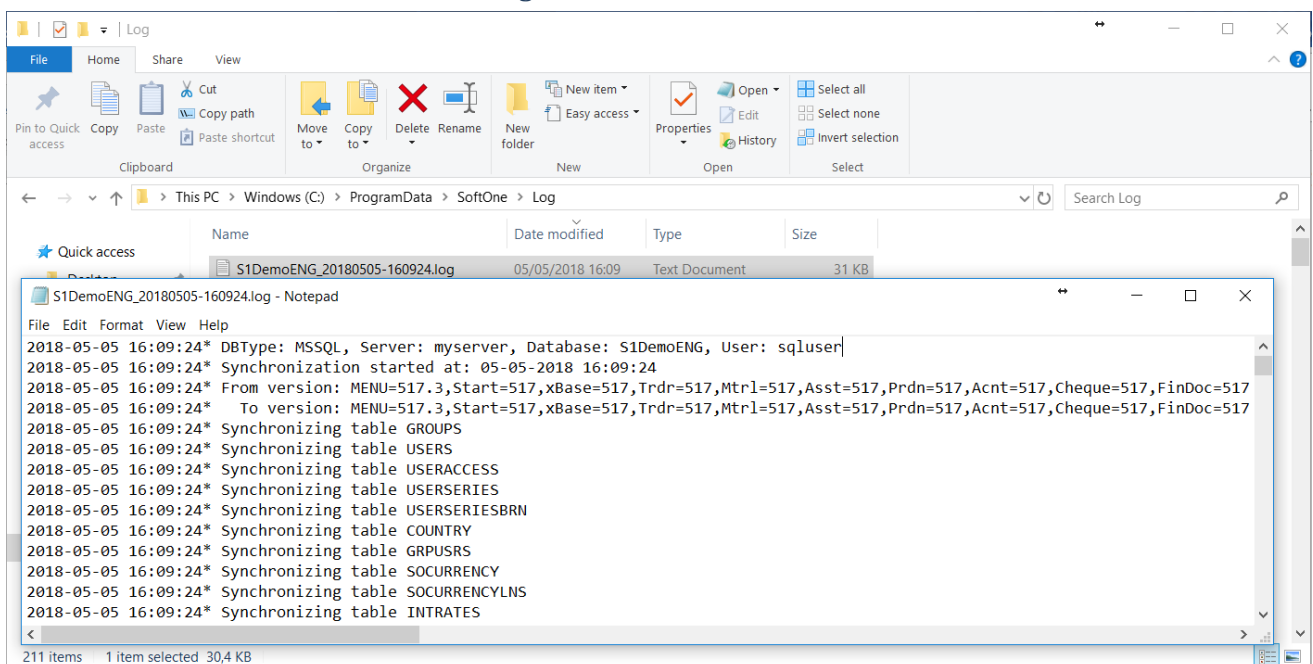


Figure A8 – Log File

A2. Publication – Custom Administration

SoftOne designer records can be exported throughout the **Custom Administration** tool as .cst files. Multiple export of designer records and other cst files (browsers, forms, etc.) is also supported. This is useful for creating comprehensive **Solutions**, which can be also enriched with additional publication information, such as the manufacturer’s name, date restriction – shareware (Figure A10) or restriction to certain SoftOne Serial Number. All the above can be password protected and suggest that any custom solution can be copyrighted (Figure A10).

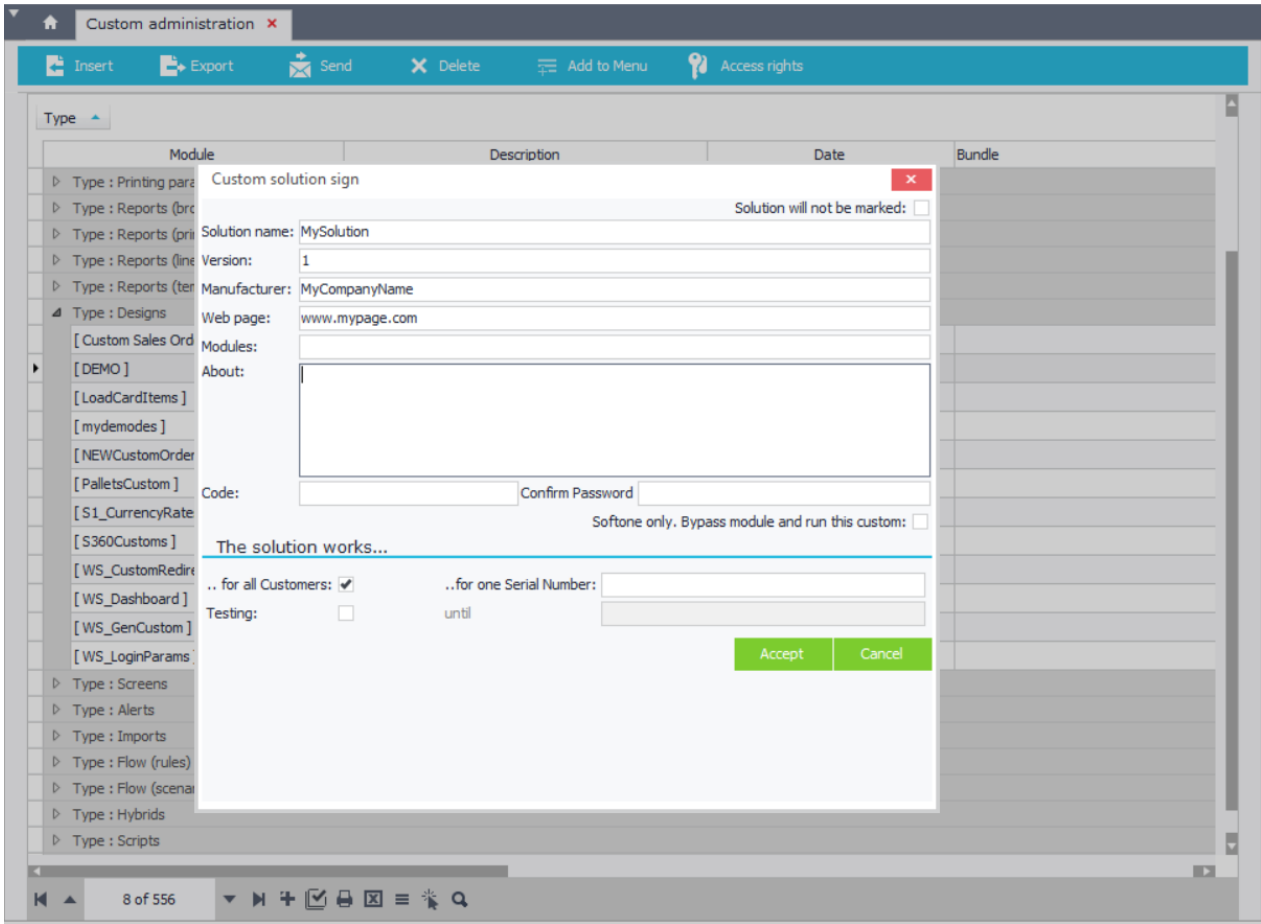


Figure A9 - Custom solution publication for All Customers

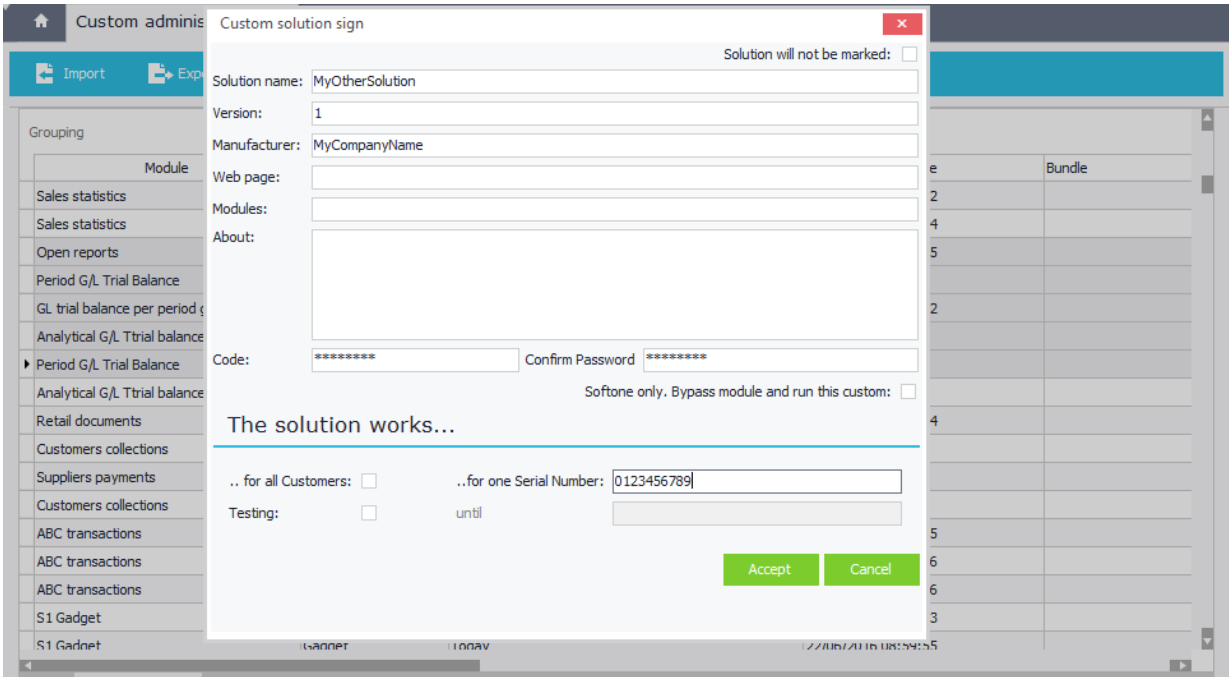


Figure A10 - Custom solution publication for a single Serial Number (password protected)

Custom Administration records (Designers, Forms, Browsers, etc.) can also be grouped using **Bundles** (Figure A11).

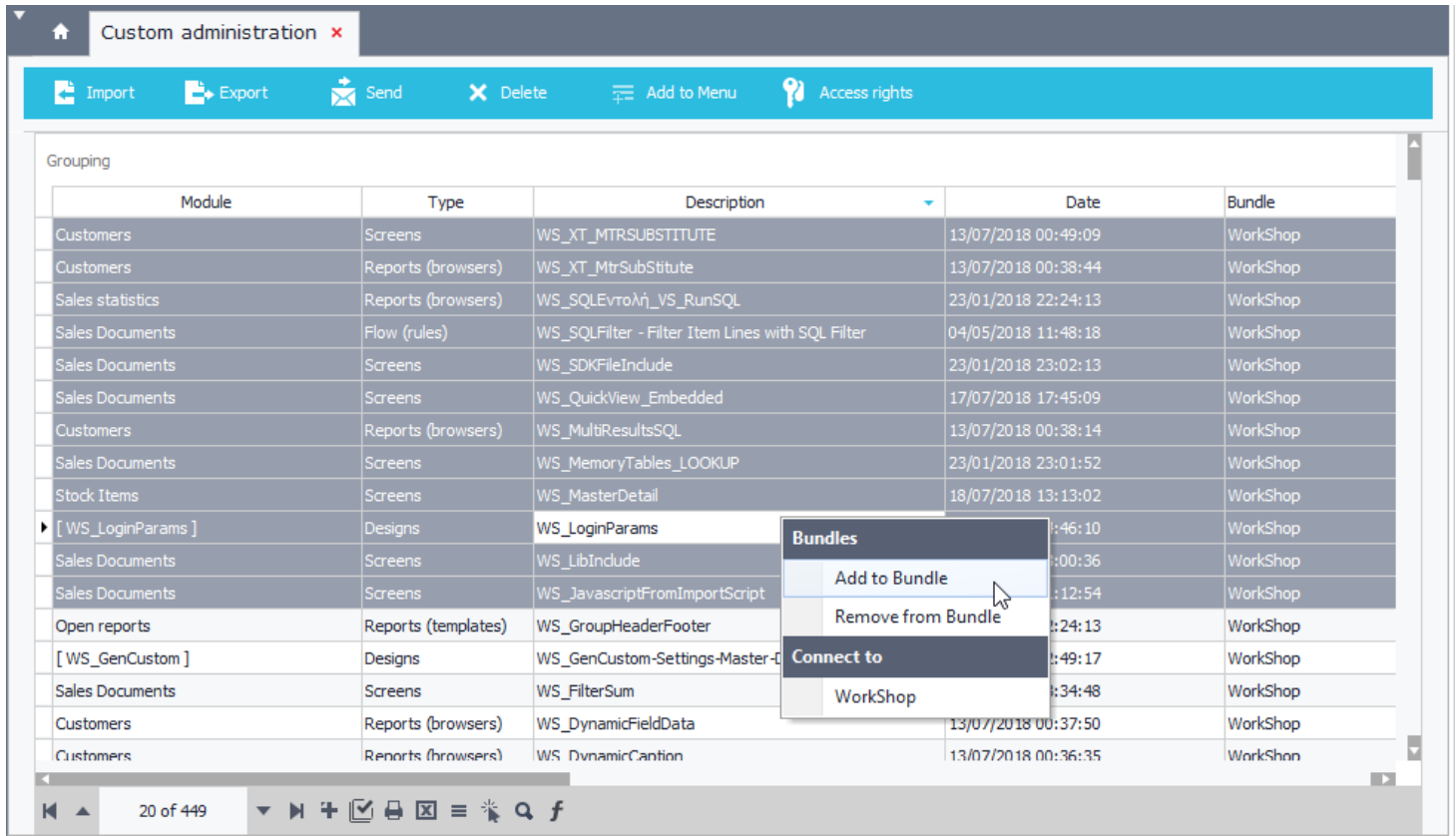


Figure A11 – Bundles

Note that **search** is also available using **CTRL+F**, or the corresponding button below the grid (Figure A12).

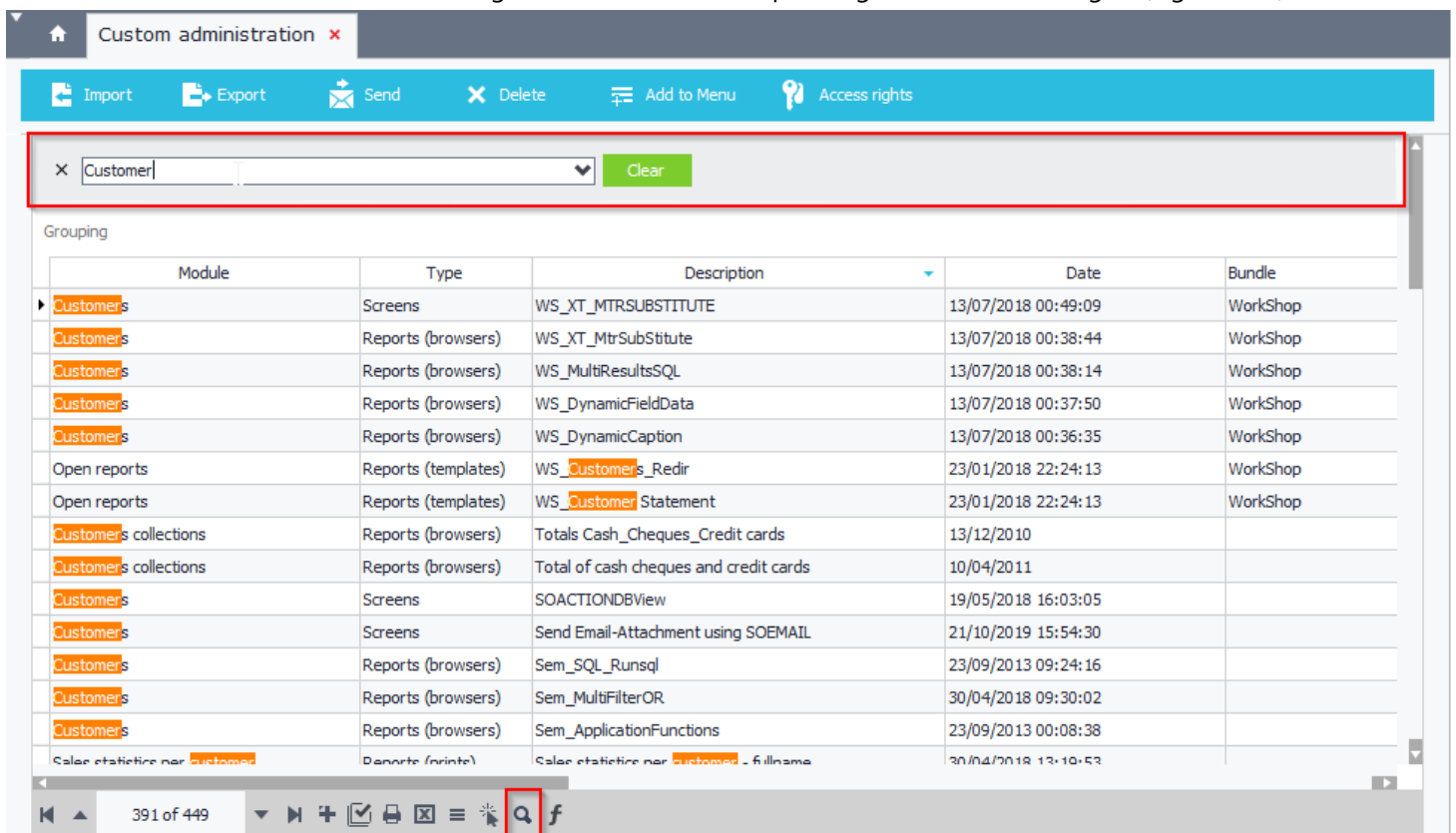


Figure A12 – Search

B. Fields

Inside SoftOne designer entries you can create new fields in any table of the database, even SoftOne internal tables. In order to create a new field, you have to select the desired table (e.g. PRSN), move to the last field and then use the down arrow of the keyboard, which will bring you up a new empty field record.

Field names should always begin with the prefix **CCC**. This ensures that these fields will not be altered in any way when upgrading to newer SoftOne versions. Fields are stored in database, so sync is required in order for them to be available for further design in forms, browsers, etc.

Note: It is recommended that fields' length should be no longer than 14 characters.

Figure B1 shows an example of adding fields inside table MTRL using different data types.

Float Fields can be set to display the same number of decimal places as the ones you have selected inside each company's properties (Figure B2 - [Numeric TextBox Control](#)).

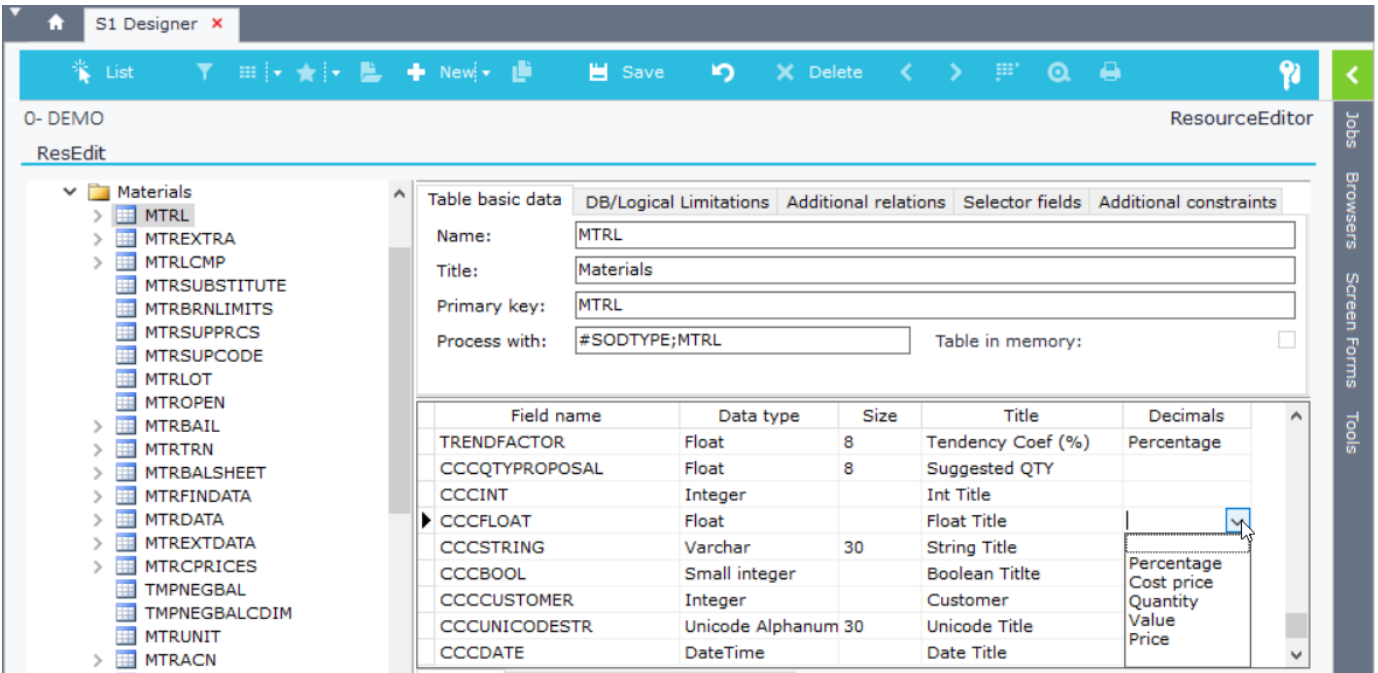


Figure B1

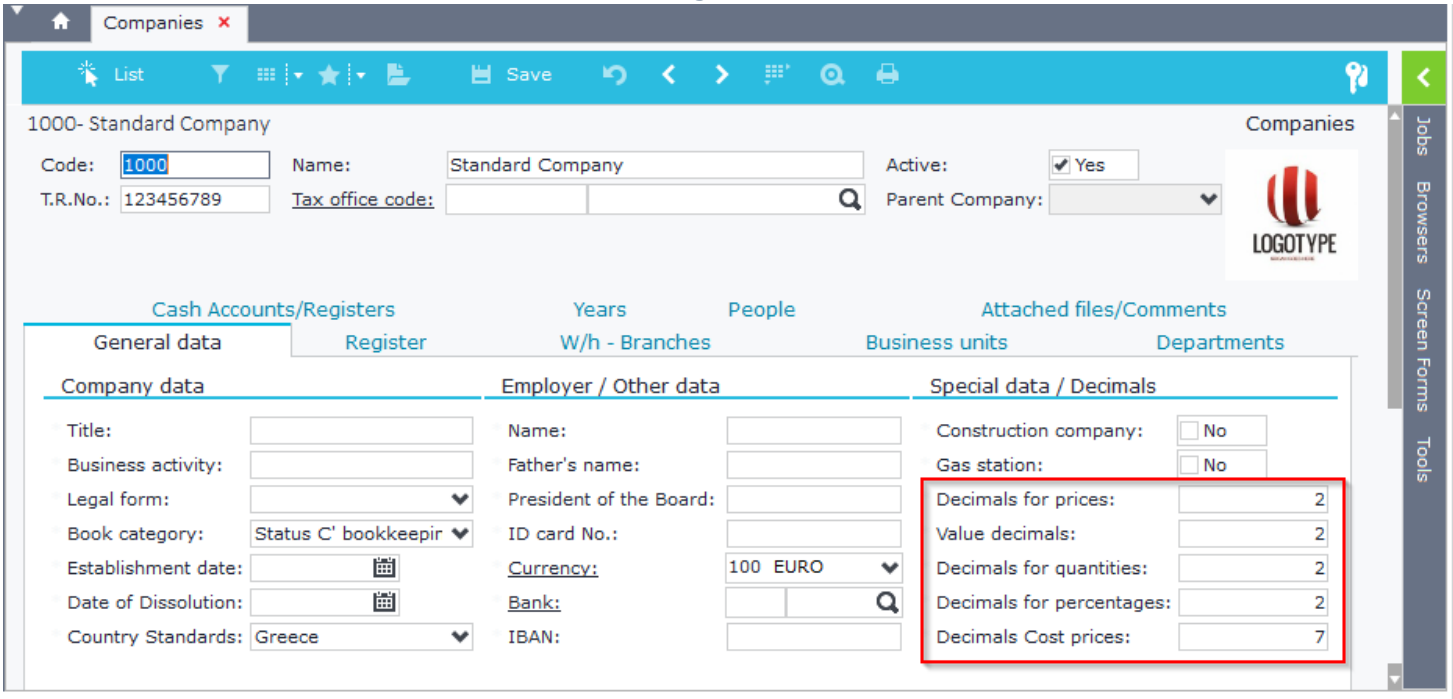


Figure B2

B.1 Field Data Types

Database table fields require a name and a data type and can be displayed with different controls depending on the given editors. Analysis of all the available Editors is provided in the section [Editors](#) in Chapter "Screen Forms". The following table lists the available data types and their specifications. Always keep in mind that a table can contain a maximum of 8,060 bytes per row (SQL Server).

Data Type	Min	Max	Storage Size
Small Integer	-32,768	32,767	2 bytes
Integer	-2,147,483,648	2,147,483,647	4 bytes
Float	-1.79E + 308	1.79E + 308	4 bytes (24 digits)
CInteger (Identity)	1	2,147,483,647	4 bytes
DateTime	1753/01/01	9999/12/31	8 bytes
Varchar	0 chars	8000 chars	number of chars + 2 bytes
Text	0 chars	2,147,483,647 chars	number of chars + 4 bytes
Image	0 bytes	2,147,483,647 bytes	
Unicode (nVarchar)	0 chars	4000 chars	2 * number of chars + 2 bytes

B.2 Field Properties

Field properties (Figure B4) are summarized in the table below:

Field name	Data type	Size	Title	Decimals
USERS	Small integer	2	Code	
CODE	Varchar	25	User	
NAME	Varchar	64	Full Name	
ISACTIVE	Small integer	2	Active	

General	Display	Search by	Information
Size:	2		
Title:	Code		
Required:	<input checked="" type="checkbox"/>		
Default value:			
Forced value:			
Always included in SELECT statement:	<input type="checkbox"/>		

Figure B4

PROPERTY	DESCRIPTION
Size	Sets the size of the field (Varchar data type).
Title	Sets the title of the field
Required	Specifies whether a value for the field is required for the record to be saved (e.g. null values are not permitted).
Default Value	Sets a default value to the field that applies to new entries. Users can change it at runtime.
Forced Value	Indicates the forced value applied to a field. Users cannot change it at runtime.
Always included in select statement	Indicates that the field will always appear in SQL SELECT statement.
Size (in characters)	Sets the default width size display of the field inside SoftOne (when displayed in datagrids or browsers).
Hidden field	Indicates whether the field is displayed or not.
Editor	Sets the field editor. See editors .

B.3 Selector Fields

Fields can be set to retrieve and store data from other tables, database views, view tables or string lists. These links (selectors) to other objects are done using **editors** and the data stored depend on the editor attributes used.

The available data types for creating selectors are "Integer", "Small Integer" and "Varchar" and they are usually used according to the primary key of the linked object (table, view, string list).

A selector is built upon entering the name of the object (table, view, string list) in the **Editor** property of the field (tab **Search by**). It is recommended to use the same field names as the corresponding linked objects.

In Figure B3 you can see an example of linking the field TRDBRANCH of the table FINDOC with the table TRDBRANCH. Apart from the linked table name, there is also an extra command F[TRDR=:FINDOC.TRDR] that limits the corresponding data (of TRDBRANCH table) to the ones that the value of field TRDR is the same as FINDOC.TRDR value. These commands are explained in the section [Editors Attributes](#).

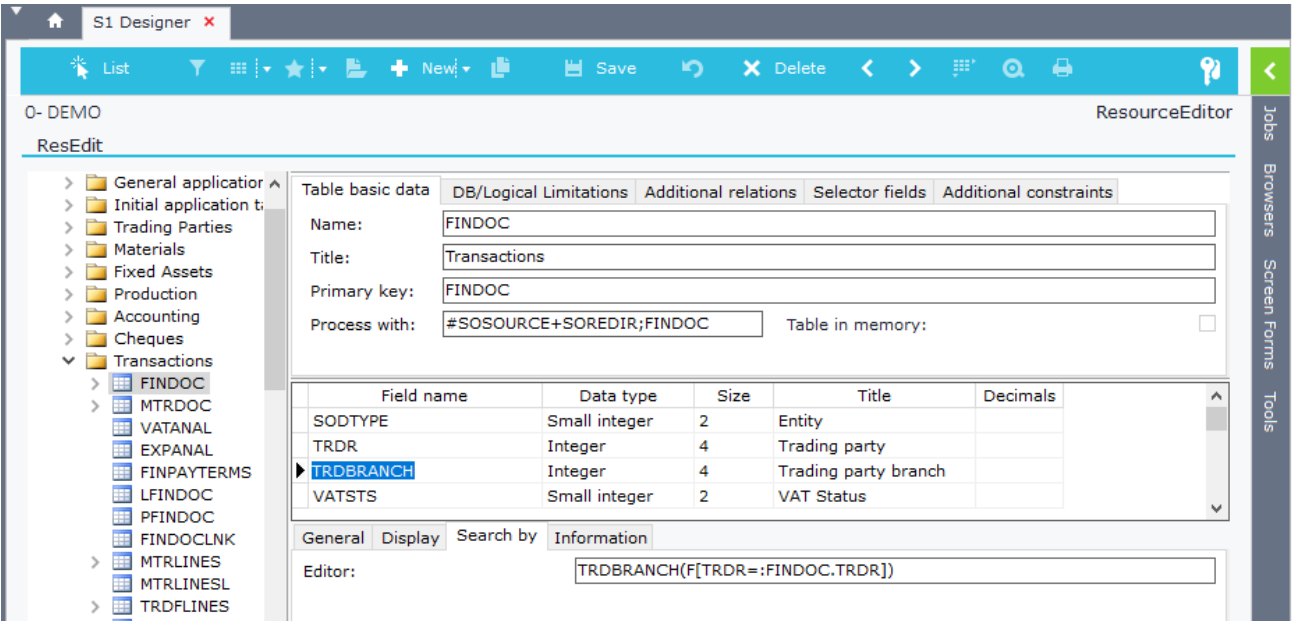


Figure B3

Another use of editors is for altering the control that displays the fields' data inside forms. The available editor commands are explained in the section [Editor Commands](#). In Figure B4 there is an example of the field ISACTIVE (Table TRDR) that is displayed as a checkbox (Yes / No), because it uses the command \$Y.

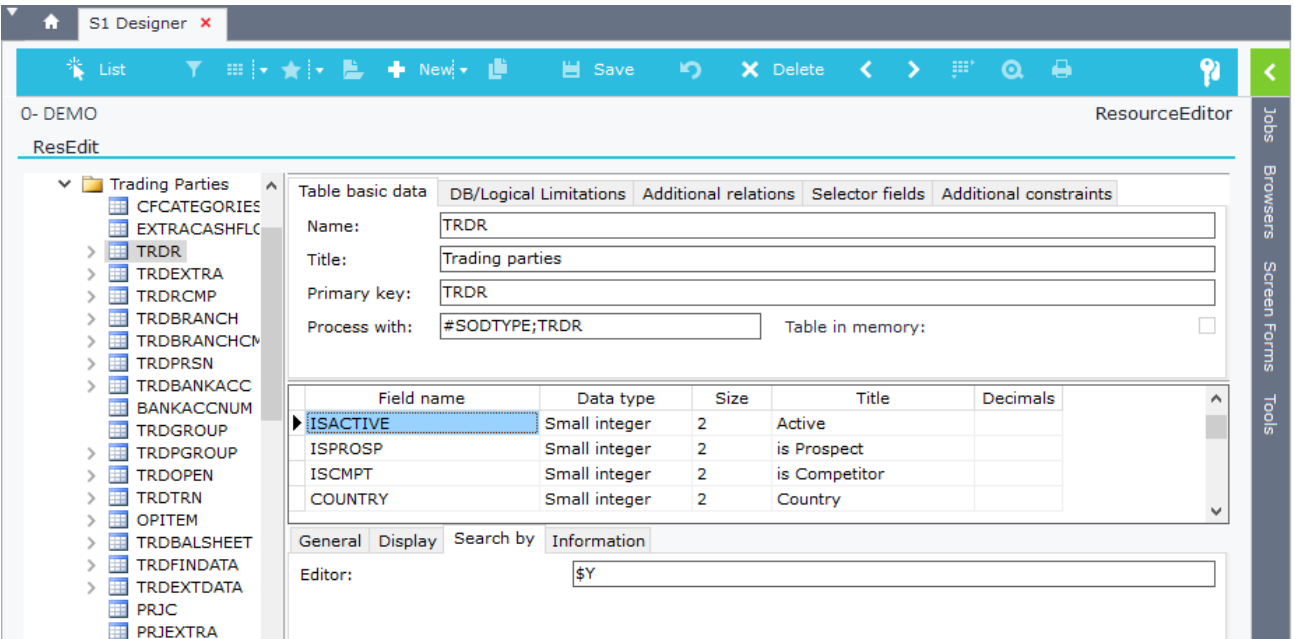


Figure B4

C. Tables

New tables are created in database using the right click option “New – Table”. Table names must always begin with the prefix **CCC**, while the fields inside them can be created using any name.

The name of the primary key field is recommended to be the same as the table name. This maintains the same logic as in SoftOne tables, which shortens the process of using internal commands in controls (editors, locate statement expressions etc.).

Note: Field names **COMPANY** and **ISACTIVE** are predefined and they always appear in the “where clause” of the “selector list” SQL statements. This means that if you create a custom table that contains the above fields and you use it as selector table (editor) in a field, then the SQL statement for retrieving data will be like the following:

SELECT FLD1, FLD2,... **FROM** CCCMYTABLE **WHERE** COMPANY=[LoginCompany] **AND** ISACTIVE=1 **AND**...

Tables are stored in database, so sync is required for them to be available in SoftOne object forms, browsers etc.

C.1 Table properties

Table properties are defined through the different options of the tabs “Table basic data”, “DB/Logical Limitations”, “Additional relations” and “Selector fields” that are discussed in the following sections.

Figure C1

C.1.1 Table Basic Data

The available properties of this tab (Figure C1) are summarized below:

Name

Name of the table (Database Name). Always use prefix CCC.

Title

Title / caption of the table. This is the name that will be displayed when adding the table in SoftOne objects.

Primary key

Table primary key fields divided by semicolons that will be used to create the database table primary keys (e.g. CCCFIELD1;CCCFIELD2;CCCFIELD3).

Table in memory (Checkbox)

This option defines that the data of the custom table will be loaded in memory on login. Use it in tables that don't have a lot of data. Analysis of this type of tables can be found in the section [C2. Tables in memory](#).

Process with

This property is the same as the **H[ObjectName]** [editor attribute](#) which is defined in field editors and has different uses depending on the type of the table you want to create (Memory, Master, Child Table). It also provides a different use when creating database views.

A. Object Redirection

Custom tables with corresponding custom objects like SoftOne tables (Customers, Items, etc.), can be used as selectors to other fields. Example of this is the field TRDR (Customer) of table FINDOC (Sales Documents) that is linked to the inherited table Customer (under the database table TRDR).

If these tables are used as selectors from other fields, then the caption of the selector fields is turned into hyperlink and redirects to the corresponding object that is defined inside the property **"Process with"**. Keep in mind that fields are linked to tables through the editor property. These **selector fields**, that retrieve data from tables, are displayed via **"Selector List"** control as in figure C2.

Figure C2 shows an example of a custom field named "Model" (CCCAIRMODEL) that retrieves data from the table "CCCAIRMODEL" and links ("Process with" property) to the object "CCCAIRMODEL". This means that clicking on the caption (hyperlink) of the field "Model" redirects you to the custom object "Aircraft Models" (Figure C3).

Note: It is advised to use the name for tables and objects linked to them, as in the above example.

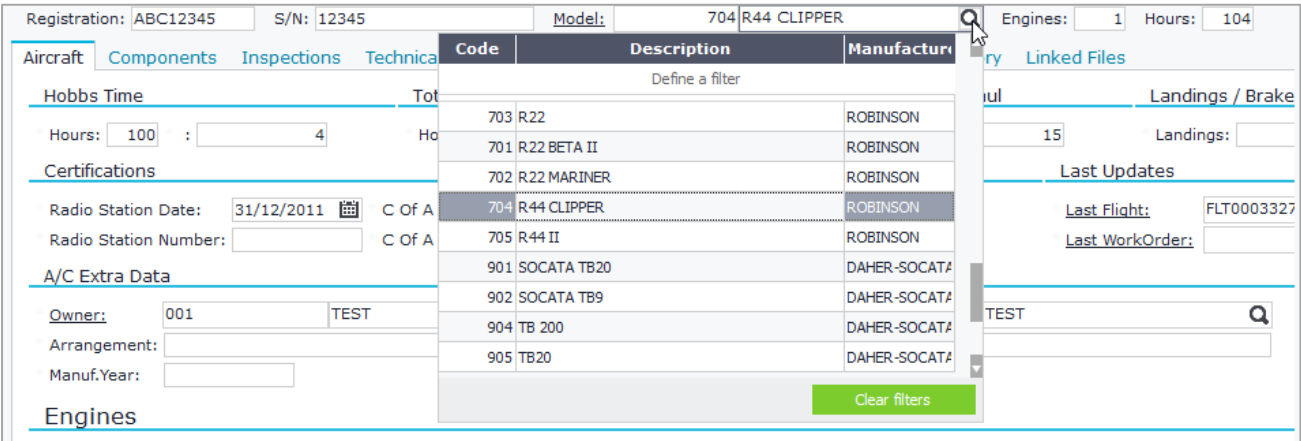


Figure C2

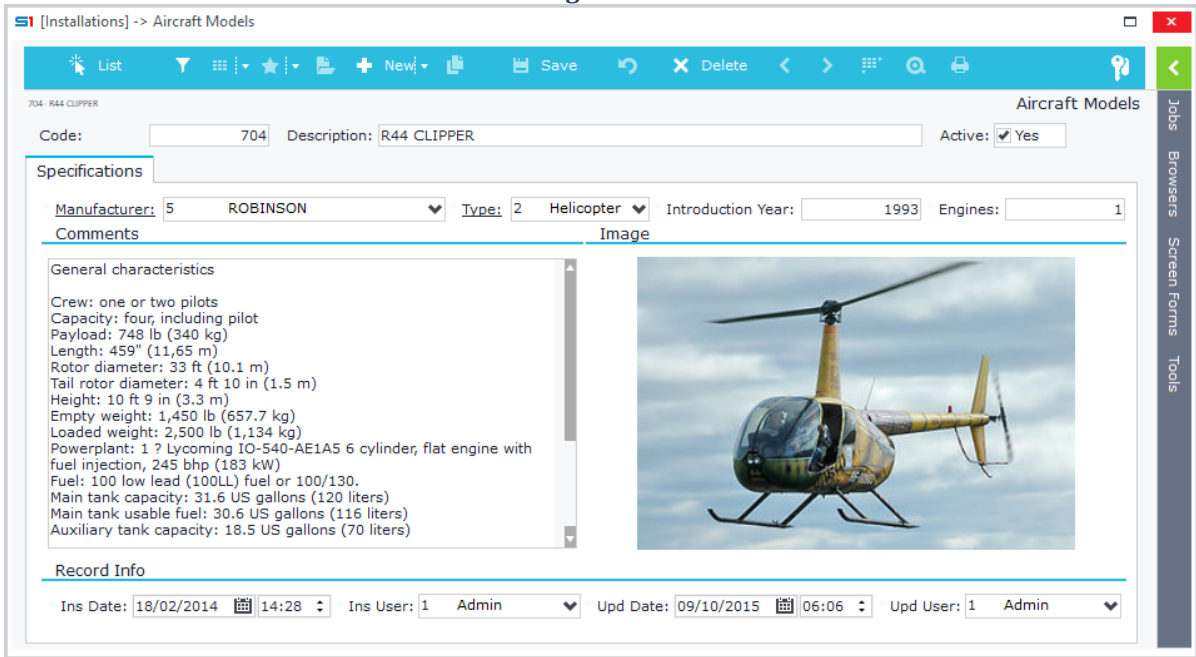


Figure C3

B. Memory Tables Processing (Command \$)

Process with \$ (Figure C4) is used for tables in memory and defines that the data of these tables will be processed through a simple edit list control (Figure C5) and not using an object. The menu job that displays the table for editing is created using the job type "Table processing" as in Figure C6.

Table basic data

DB/Logical Limitations

Additional relations

Selector fields

Name:

CCCAIRMANFCTR

Title:

Aircraft Manufacturers

Primary key:

CCCAIRMANFCTR

Process with:

\$

Table in memory:

☒

Field name	Data type	Size	Title	Decimals
CCCAIRMANFCTR	Small integer	2	Code	
NAME	Varchar	128	Name	
ISACTIVE	Small integer	2	Active	
COMMENTS	Varchar	512	Comments	

General

Display

Search by

Information

Size:

2

Title:

Code

Required:

☒

Default value:

Forced value:

Always included in SELECT statem

☐

Figure C4

Aircraft

S1 Designer

Aircraft Manufacturers

Save

Delete

Code	Name	Active	Comments
1	DIAMOND	Yes	
2	AIRBUS H/C	Yes	
3	AIRBUS H/C DEUTSCHLAND	Yes	
4	PIPER	Yes	
5	ROBINSON	Yes	
6	SIKORSKY-SCHWEIZER	Yes	
7	DAHER-SOCATA	Yes	
8	AIRBUS H/C SUPER PUMA	Yes	
9	AGUSTA	Yes	

1 of 11

Figure C5

S1 Job parameters

General Data

Properties

Job type:

Table processing

Command/File:

CCCAIRMANFCTR

Available jobs

Job title:

Aircraft Manufacturers

OK

Cancel

Figure C6

C. Browser Redirection

The property “**Process with**” can be used for object redirection from browser records. This is usually used in database views that are displayed in objects and have only a browser view and not a form view.

An example of the above is the object-database view “Sales Statistics” (Figure C6). When users double click on one record/line of the browser they are redirected to the corresponding object and record (SALDOC or RETAILDOC) that is defined in the property “Process with” of the database view.

Date	Code	Name	Code	Description	Sales qty 1	Sales value
01/01/2018	117	Kalampoka Melina	10001	TV LED 32 LG RTDXA32	10,00	2.000,00
01/01/2018	117	Kalampoka Melina	10002	TV LCD 21	1,00	200,00
01/01/2018	117	Kalampoka Melina	10003	Digital Camera 8 MP	2,00	200,00
01/01/2018	908	General Retail custom	10009	Home cinema 1000 Watt	1,00	74,63
01/01/2018	101	Salesconsul GmbH	10003	Digital Camera 8 MP	1,00	51,48
01/01/2018	101	Salesconsul GmbH	10002	TV LCD 21	25,00	7.500,00
01/01/2018	101	Salesconsul GmbH	10004	Photo michachi DSLR 10 MP	1,00	314,60
01/01/2018	101	Salesconsul GmbH	10005	Flash Camcorder	16,00	8.236,80
01/01/2018	101	Salesconsul GmbH	10006	Flash Full HD Camcorder	4,00	1.258,40
01/01/2018	101	Salesconsul GmbH	10007	HDD 60GB Camcorder	20,00	6.292,00
01/01/2018	101	Salesconsul GmbH	10008	Digital Photo Frame 8 "	1,00	57,20
01/01/2018	101	Salesconsul GmbH	10009	Home cinema 1000 Watt	1,00	88,40
01/01/2018	101	Salesconsul GmbH	10010	Crutches	1,00	180,00
01/01/2018	101	Salesconsul GmbH	10011	Pharmacy kit	2,00	83,20
01/01/2018	101	Salesconsul GmbH	10012	Liquid decontamination (Lots)	2,00	34,00
01/01/2018	101	Salesconsul GmbH	10013	Medical oxygen bottle (lots)	2,00	503,36
01/01/2018	101	Salesconsul GmbH	10014	Rescue Life Jackets	2,00	150,00
					890,00	279.504,47

Figure C6

The expression formula for browser custom redirection is the following:

#ObjectID;RecordID

This means that if for example you have created a custom object that uses the id 100001 and you want to make a redirection from a database view to this object, using as locate record the value of the field “Myfield”, then you should enter the expression: **#100001;Myfield**

ObjectID is the unique number that defines objects. For example, the objectID of Customers is 13, of Items 51, of Sales Documents 1351, of Purchase Documents 1251, of Retail Documents 11351. This means that by entering the number of the object, the records of the browser will redirect the user to the specified object. There is also the option of variant object redirection, that depends on the value of a field of the table.

For example, the Sales Statistics object redirects either to Sales Documents or to Retail Documents depending on the value of the fields SOSOURCE and SOREDIR. Sales and Retail Documents share the same SOSOURCE value, which is 1351, but the value of the field SOREDIR is 0 for Sales Documents and 10000 for Retail Documents.

So, the “**Process with**” property of the database view VSALSTAT is: **#SOSOURCE+SOREDIR;FINDOC;MTRTRN**. SOSOURCE and SOREDIR values define the redirection object id (e.g. $1351+10000 = 11351$ which is the id of RETAILDOC [Retail documents], $1351+0=1351$ which is the id of SALDOC [Sales documents]).

RecordID defines the record of the redirection. The above example uses two fields, FINDOC value is used for locating the header record and MTRTRN for locating the child table record (MTRLINES).

Note: All the fields that are used inside the property “Process with” need to be in the select statement of the object browser. This means that you can either add them as columns inside the browser or enable the field property “Always included in select statement” of the database view.

In Figures C7 and C8 you can see an example of a custom database view CCCVMTRSTAT that selects data from the table MTRTRN, just as VMTRSTAT. The “**Process with**” property is: **#SOSOURCE+SOREDİR;FINDOC;MTRTRN**. Also notice that in order to use the object CCCVMTRSTAT in browser-only mode, you need to use the parameter **BROWSERONLY=1** (Figure C9).

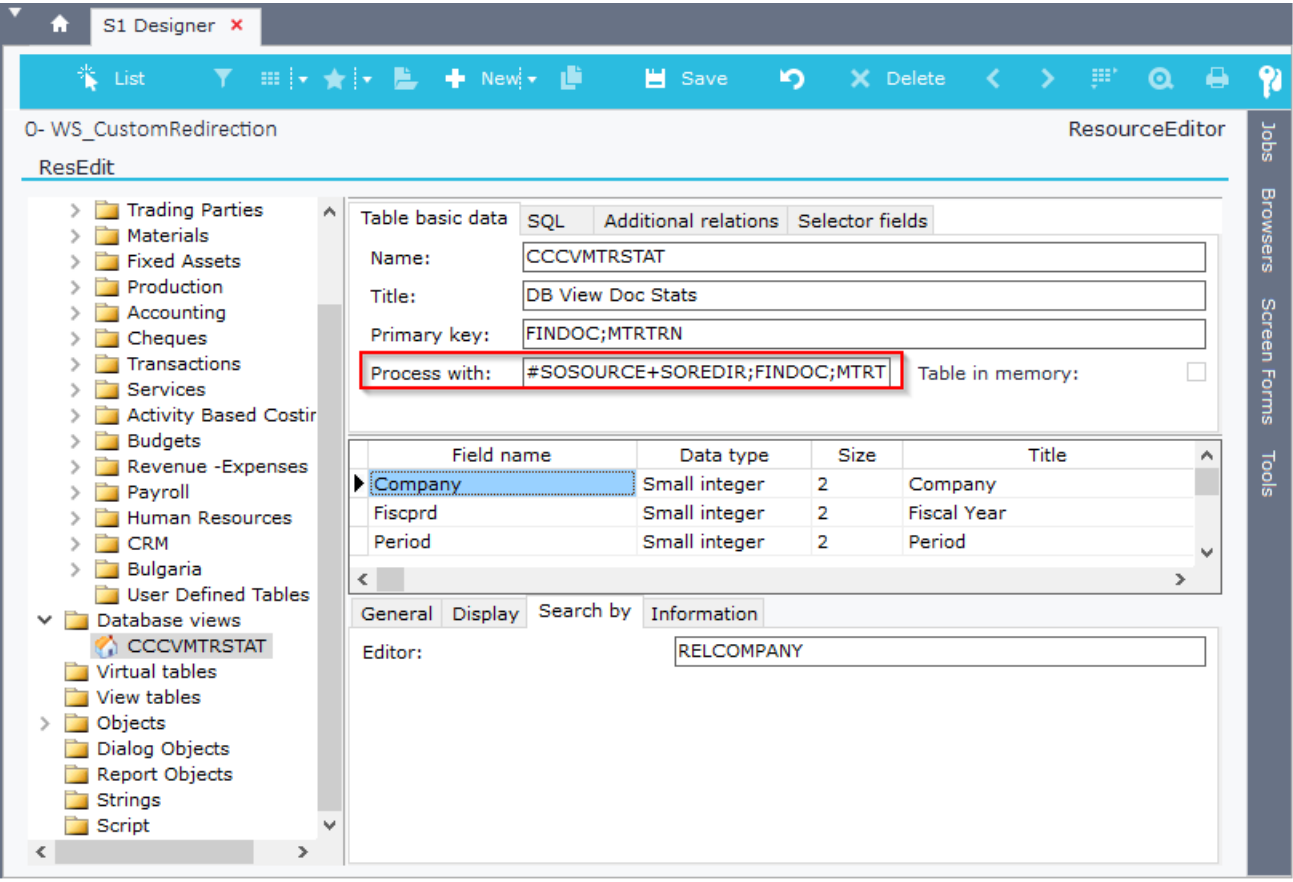


Figure C7

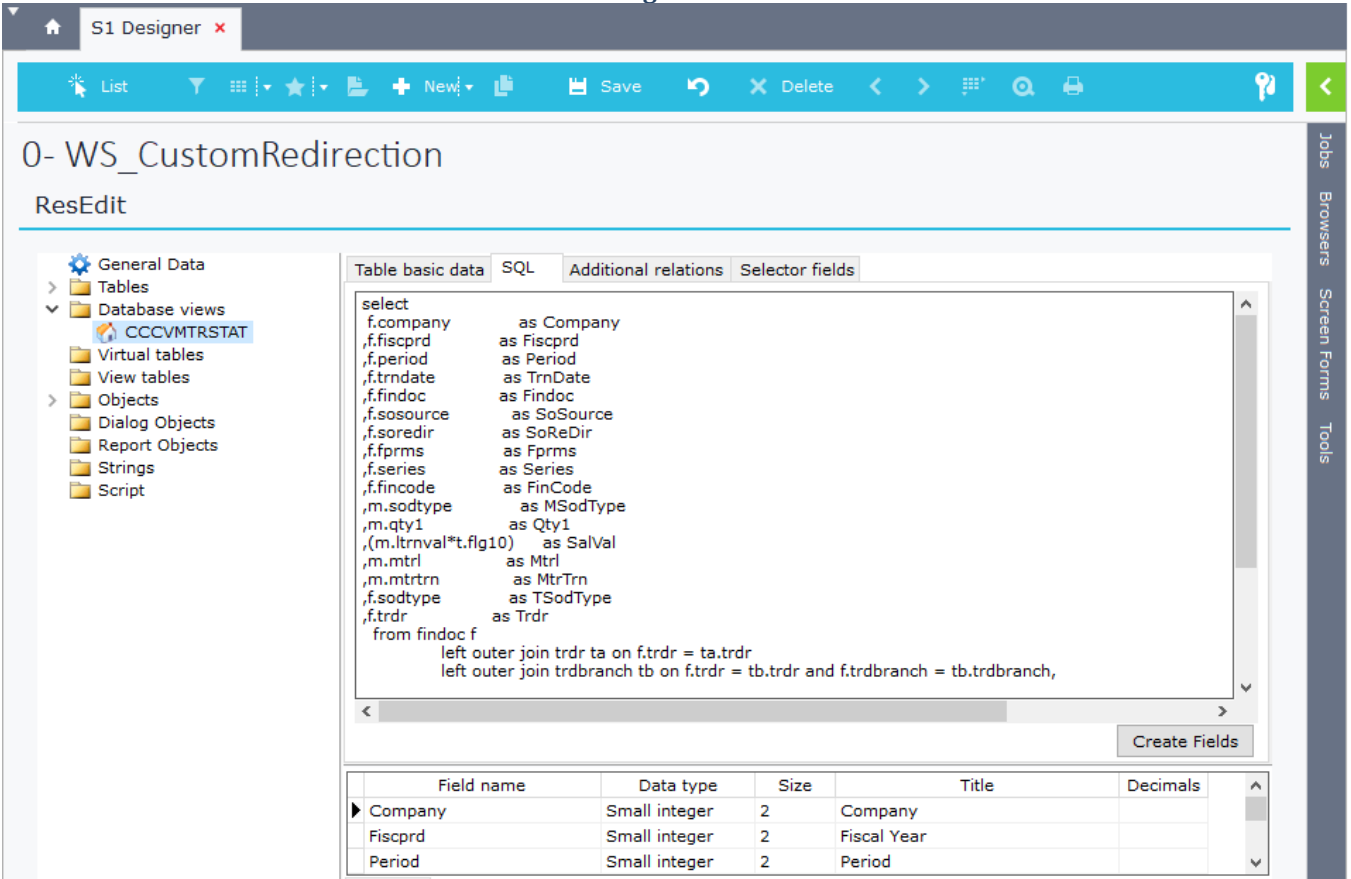


Figure C8

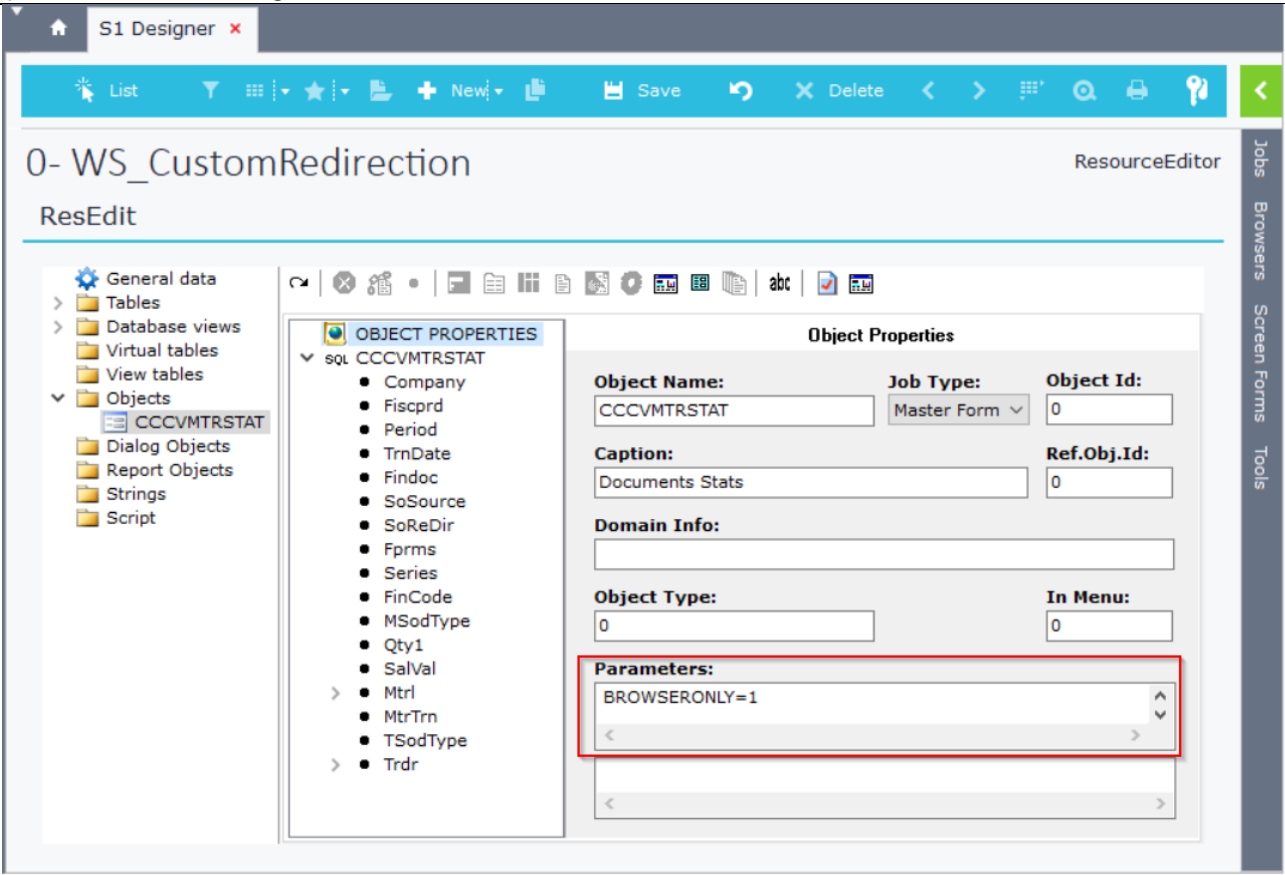


Figure C9

C.1.2 DB/Logical Limitations

This tab contains the constraint definitions of tables. The available commands are described in the following table.

Database Limitations/ Logical Limitations		
Command	Process Analysis	Syntax / Example
XP	Defines the name of primary key of the table. The fields of the primary key are the ones entered in the textbox "Primary key" of the tab "Basic Data".	XP_KEYNAME (KEYNAME is the name of the primary key)
XU	Creates a unique, non-clustered index. Used when you want to prevent duplicate values.	XU_KEYNAME = FIELD1, FIELD2,... Example: Unique customer branch codes. XU_TRDBRANCH_CODE=TRDR, CODE
XI	Creates non-unique, non-clustered index. Used for creating indexes.	XI_KEYNAME = FIELDNAME1, FIELDNAME2,...
XF	Creates a foreign key to a table.	XF_TABLE_FOREIGNKEY = REFERENCETABLE or when fields of both tables and foreign table have the same name XF_TABLE_FOREIGNKEY Example 1: Table TRDR Command: XF_TRDR_SALESMAN = PRSN ConstraintName: XF_TRDR_SALESMAN Foreign Key: SALESMAN Reference Table: PRSN Example 2: Define that the field COUNTRY of the table Bank has a foreign key to the primary key field COUNTRY of the table XF_BANK_COUNTRY
XC	Check constraint Sets restrictions on data added to the table.	XC_KEYNAME = (FIELDNAME>=value) Example: Restrict values entered to FIELD1 (Values permitted are between 100 to 500). XC_KEYNAME =(FIELD1> 100) AND (FIELD1<500)
XD	Foreign key with Cascade Delete. Used in child tables. If a record of the parent table is deleted then the corresponding records of the child table are also deleted.	XD_CHILDTABLE_HEADERTABLE Example: Cascade delete customer branches. XD_TRDBRANCH_TRDR
XT	Additional relations / Additional constraints (SoftOne or CCC tables) 1. Defines table links to fields (apart from editor links) e.g. In Customers browser, Field TRDR is linked to Financial Data). 2. Defines the "Expand by" property of browsers (e.g Item Lines in Sales Documents). It can be used either in SoftOne tables (tab Additional constraints) or in CCC tables.	1. XT_TABLEFIELD_OBJECT=J[Extra Joins(Values First)] Example: XT_TRDR_CUSFINDATA=[J(B.COMPANY=:X.SYS.COMPANY AND B.TRDR=A.TRDR)] 2. XT_MASTERTABLE_DETAILTABLE[L(ID1;ID2;+LINENUM)] Example: XT_FINDOC_ITELINES=[L(FINDOC.FINDOC;+LINENUM)]
XR	Logical/Internal Limitation: Creates foreign key in objects	XR_TABLE_FIELDNAME = OBJECT Example: Relative item XR_MTRL_RELITEM = ITEM
XR	Logical/Internal Limitation: Cascade Delete in objects	XR_CHILDTABLE_PARENTTABLE Example: Item lines in documents XR_MTRLINES_FINDOC

C.1.3 Selector Fields

The datagrid of this tab lists the columns that will be displayed in the selector list of fields that are linked to this table (through editor property). Figure C9 is an example of creating a custom table (CCCAIRMODEL) that is used as selector from the field CCCAIRMODEL (Figure C10) and is displayed in form as in Figure C11.

Note: The selector list works properly only when you have **valid display sizes** in all the fields that you define as selector fields. This means that except for the sizes you have to define inside this datagrid, you also need to enter proper values in the **display size property** of the fields.

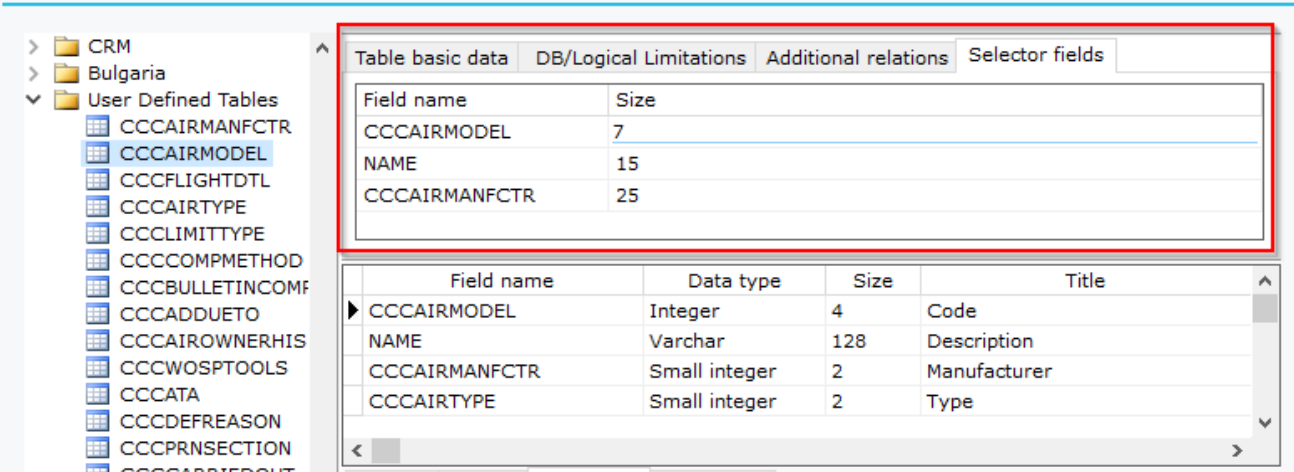


Figure C9

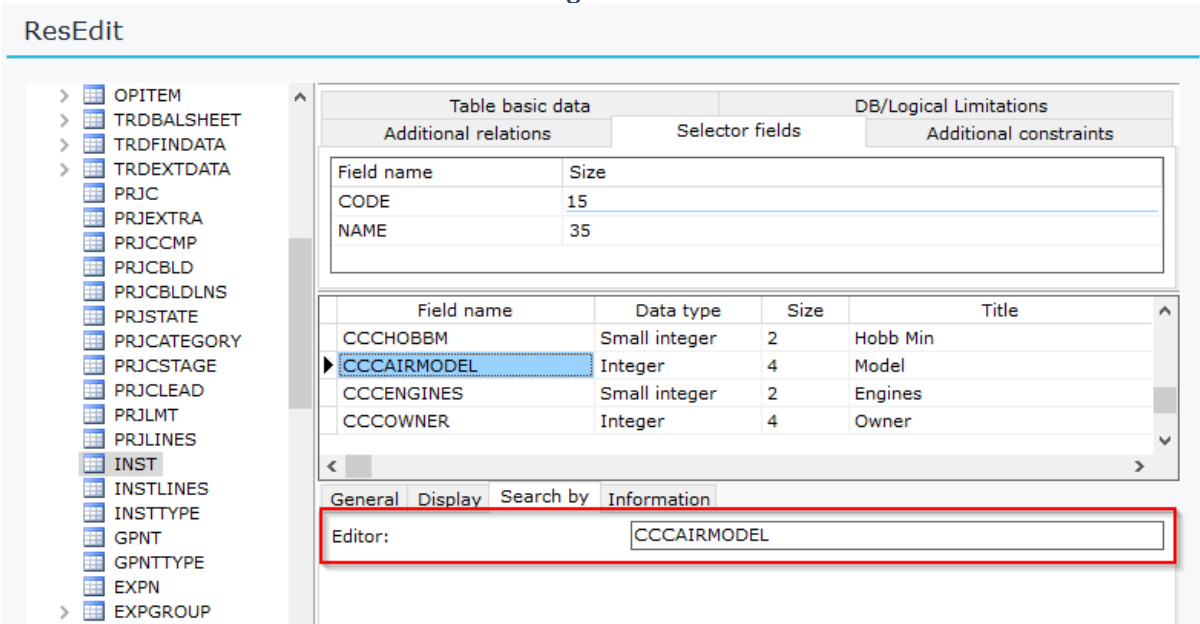


Figure C10

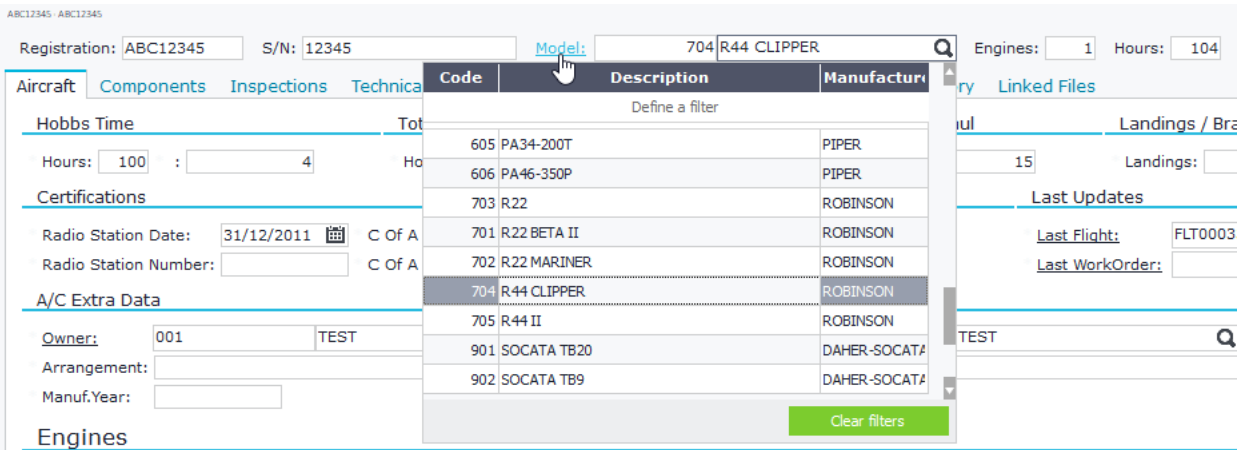


Figure C11

C.2 Memory Tables

Memory tables are database tables that are loaded into memory during application log in. This means that their data are available from every SoftOne control or script without the need of SQL statements. It is recommended that you add tables in memory only when you are certain that they will have small size.

When these tables are used as from selector fields then they are displayed as combo boxes. Their data are processed through a simple edit list control and not using an object (Figure C2.1). The menu job that displays the table data for editing is created using the job type "Table processing" as in Figure C2.2.

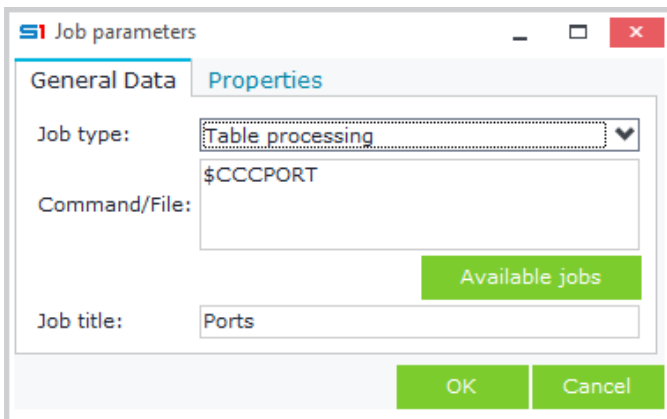


Figure C2.1

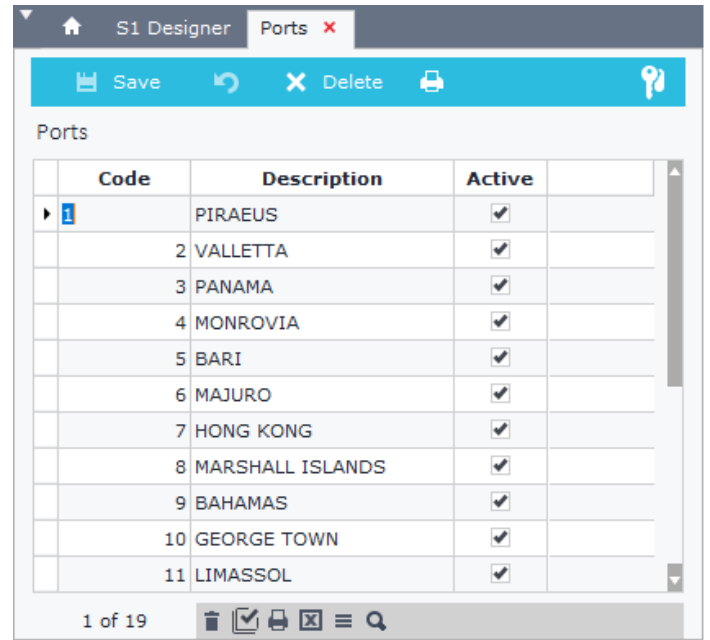


Figure C2.2

The steps of creating a table in memory are the following:

- Right click the left pane of SoftOne designer and select "New table" from the pop-up menu.
- Enter the name of the table as it will be created in the database.
- Enter the title of the table as it will be displayed in SoftOne objects.
- Enter the primary keys separated by semicolons (;).
- Enter the command \$ in the property "Process with".
- Add the fields and select the property "Required" for the **primary key fields** (Figure C2.3).

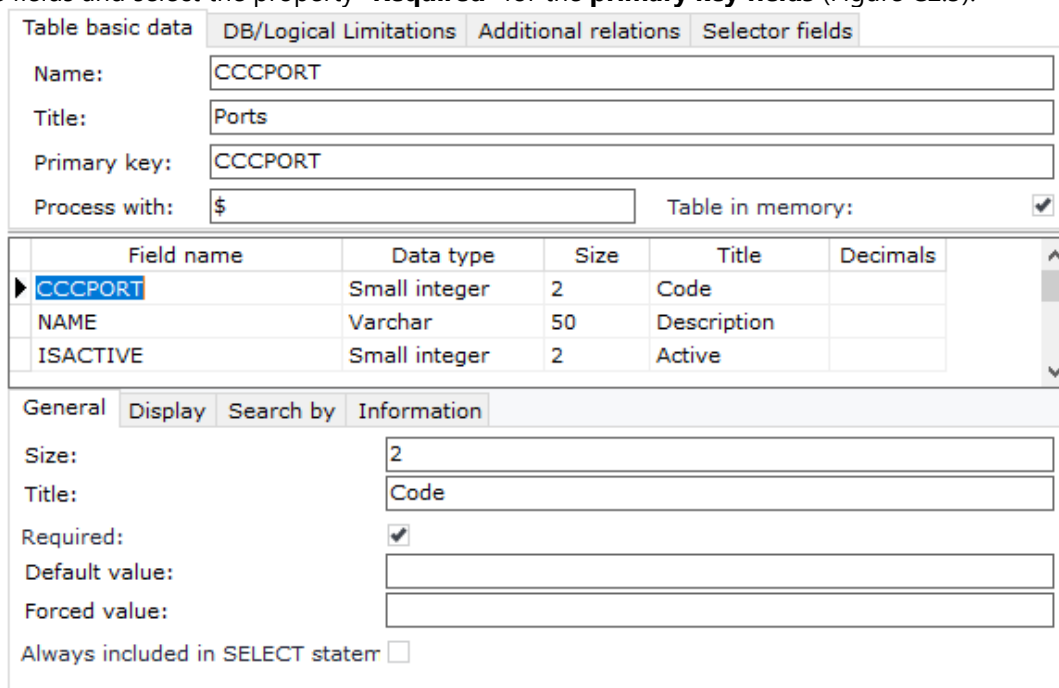


Figure C2.3

- Optionally, you may enter default or mandatory values inside the properties “Default value” and “Forced value”. Default values are auto inserted when inserting a new record, while Forced values are auto inserted when posting one (Figure C2.4).

Field name	Data type	Size	Title	Decimals
CCCPORT	Small integer	2	Code	
NAME	Varchar	50	Description	
▶ ISACTIVE	Small integer	2	Active	

General

Display

Search by

Information

Size:

2

Title:

Active

Required:

☐

Default value:

1

Forced value:

Always included in SELECT staterr

☐

Figure C2.4

- Enter the display size of the fields in order to be displayed properly in the edit list control. This affects only the display size of the column of the field as it will be displayed inside SoftOne and not the database table.
- Enter the name of the primary key in the textbox of the tab “DB/Logical Limitation” (Figure C2.5). This is the name of the PRIMARY KEY constraint of the table.

Table basic data

DB/Logical Limitations

Additional relations

Selector fields

XP_CCCPORT

Figure C2.5

- In the datagrid of the tab “Selector fields” enter the columns and their display size that will be used when this table is used as selector list from a selector field (Figure C2.6). Make sure that the display size of the fields is greater than zero or else the fields will no be visible in the selector list.

Table basic data

DB/Logical Limitations

Additional relations

Selector fields

Field name	Size
CCCPORT	5
NAME	15

Field name	Data type	Size	Title	Decimals
CCCPORT	Small integer	2	Code	
▶ NAME	Varchar	50	Description	
ISACTIVE	Small integer	2	Active	

General

Display

Search by

Information

Size (in characters)

20

Hidden field:

☐

Display field:

0

Figure C2.6

The table will be available from SoftOne forms and menu, only when you sync the database, which is done by increasing the version number of SoftOne database designer.

Link fields to memory table

Inside the configuration window of an object form, select a field, that uses the same type as the primary key of the memory table (usually small integer), and enter the name of the table in the editor property (Figure C2.7).

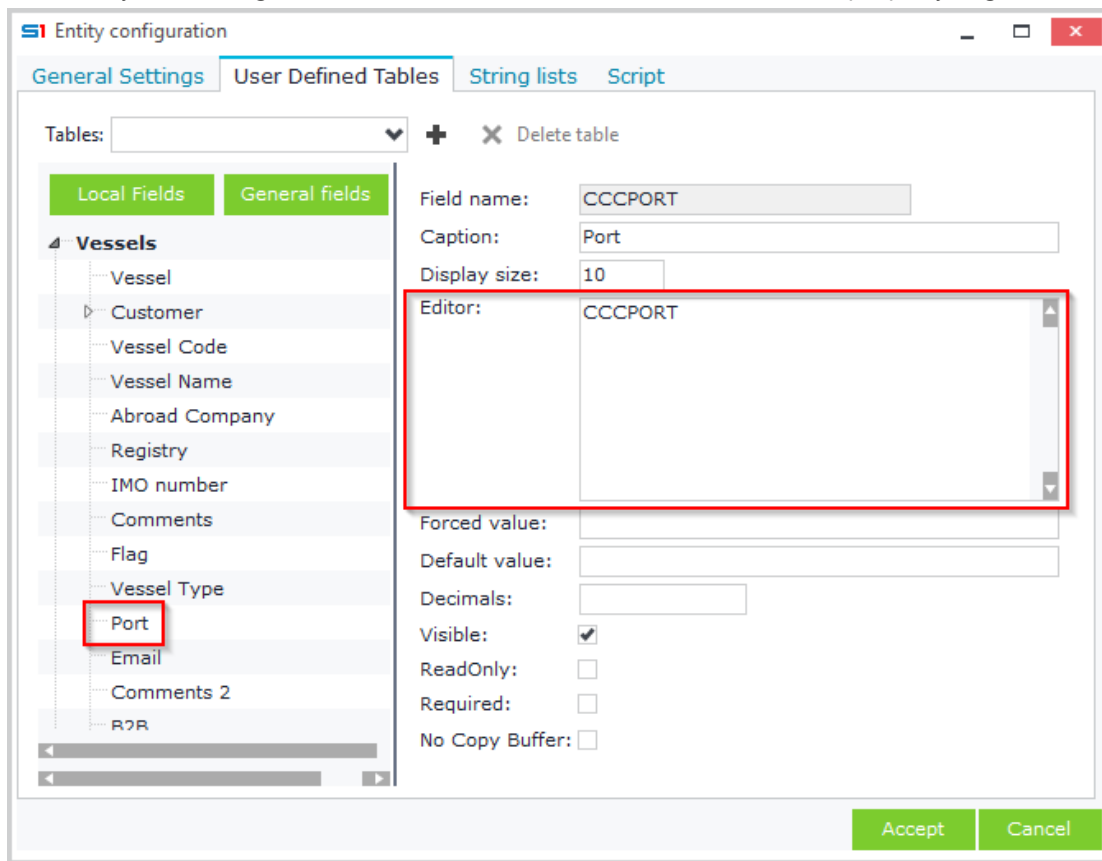


Figure C2.7

In running mode, the field will be displayed in the object form as in Figure C2.8.

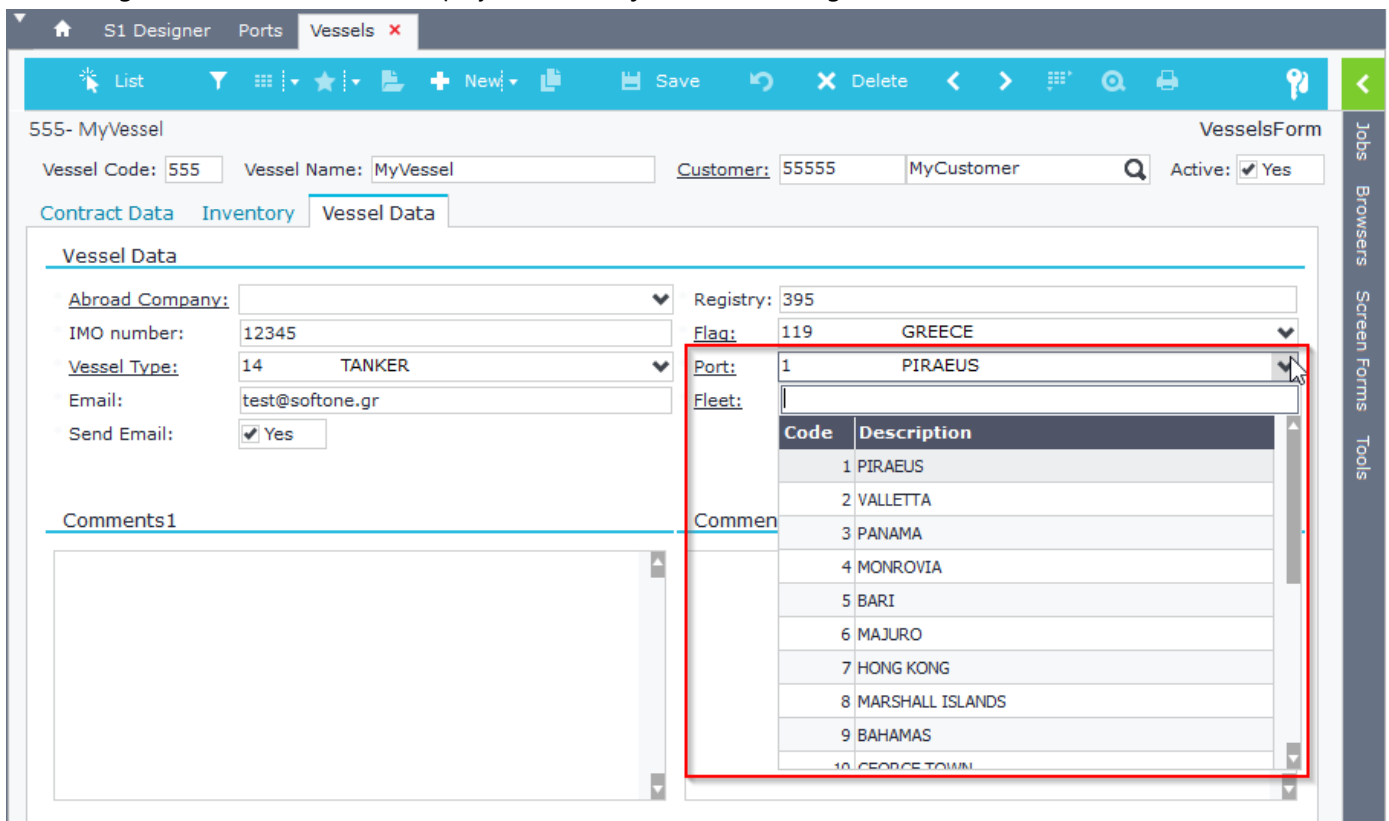


Figure C2.8

C.3 Child Tables

Child tables are ordinary database tables that are displayed through datagrid control inside object forms. They are detail tables to parent tables of SoftOne objects. That means that the relationship between them is “One to many”.

C.3.1 Design

The design of a child table uses the same principles as the design of any table, such as a table in memory. The only difference is that you need to add a field that links the child table to the primary key of the parent table (and stores it). It is recommended to name the link field of the child table as the primary key name of the parent table, so as to keep up with SoftOne logic and shorten object expressions, when you use it in editors or locate statements.

It is also advised to add a field named **LINENUM** (type integer) to all your child tables, that keeps the line numbering of the child table records per parent record. The column name LINENUM is predefined and it can generate auto-increment numbers as long as you enter the number **9999999** in the default value property of the field. After posting a record, SoftOne automatically fills fields named LINENUM with auto increment values (starting from 1). For every new parent record inserted, the line numbering of the child tables is restarted.

Note that this field is also used in **Web Service calls** ([SetData post method](#)).

Example of parent – child tables are the tables FINDOC(Documents) and MTRLINES (Document lines) where the field that links the two tables is FINDOC (primary key in table FINDOC) (Figure C3.1)

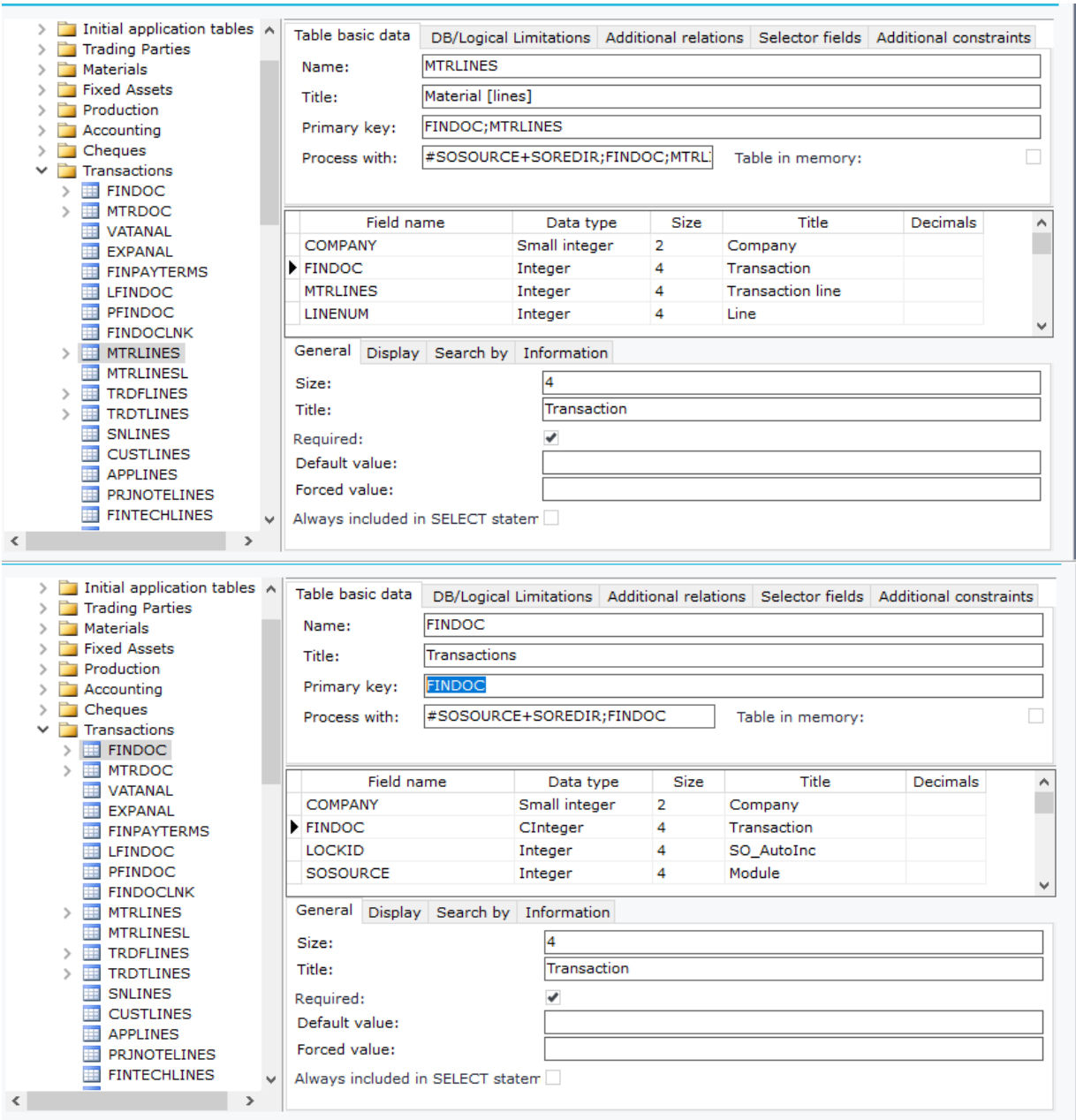


Figure C3.1

The design of a child table is the same as any other table. The only thing that you should keep in mind is to add a field that will store the data of the primary key field of the parent table.

The steps for creating the child table are the following:

- Right click the left pane of SoftOne designer and select "New table" from the pop-up menu.
- Enter the name of the table as it will be created in the database.
- Enter the title of the table as it will be displayed in SoftOne objects.
- Enter the primary keys separated by semicolons (;).
- Add the fields and select the property "Required" for the **primary key fields**

In the example of Figure C3.2 we have created a child table (CCCWSDETAIL) of the master table CCCWSMASTER (Figure C3.3). You are allowed of course to create child tables for any SoftOne table, not only for custom ones.

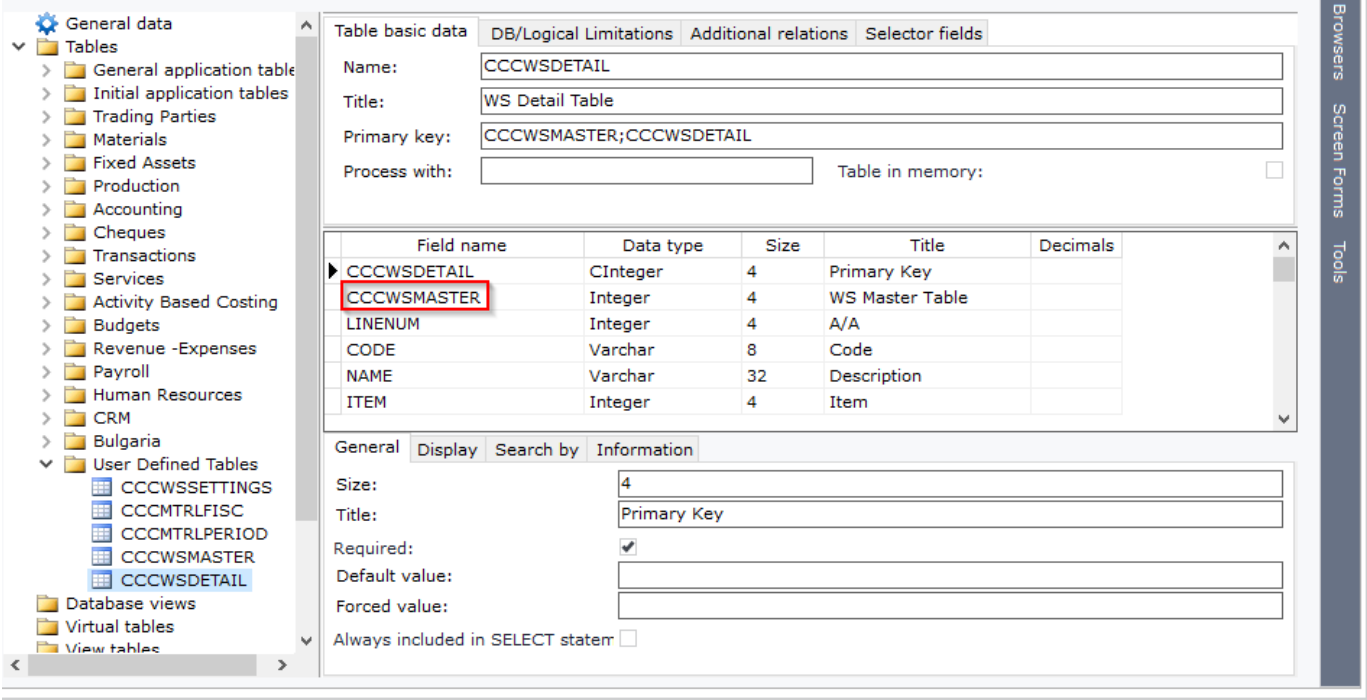


Figure C3.2

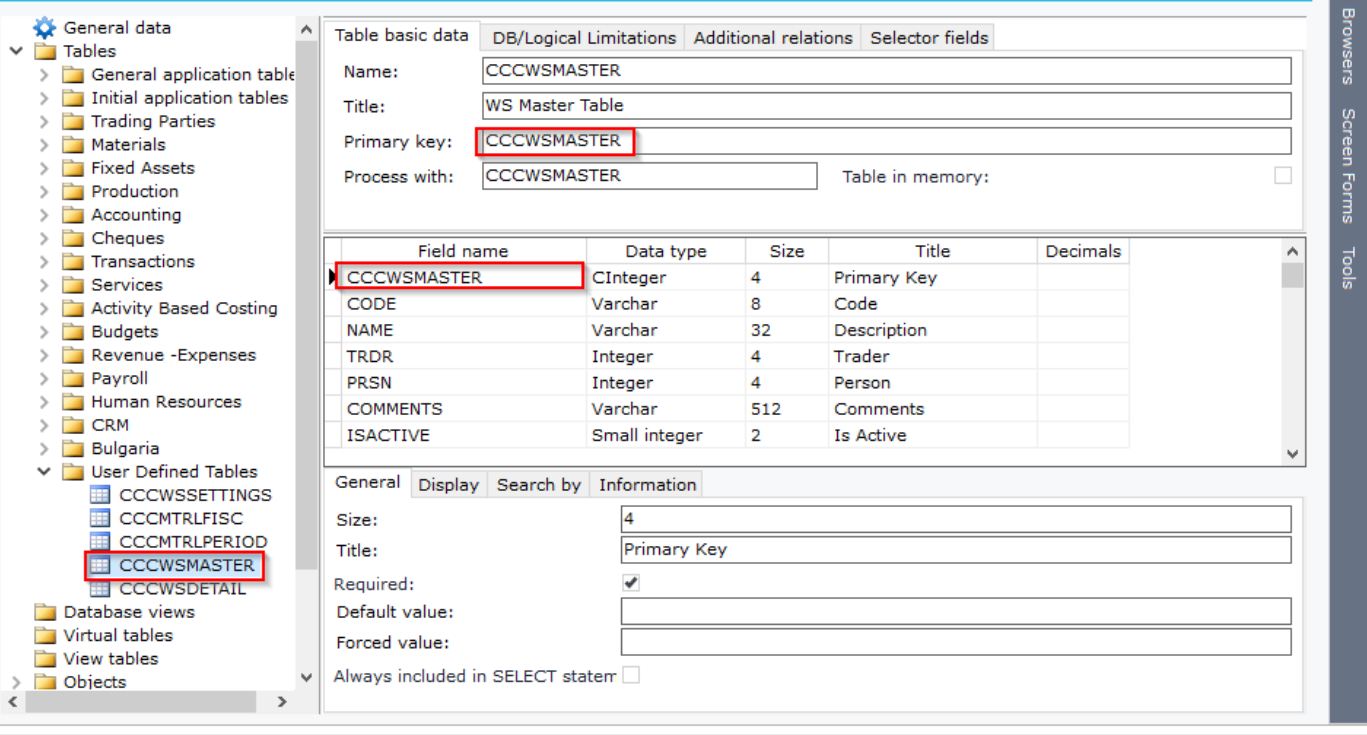


Figure C3.3

C.3.2 Process data in object form

Display a detail table in an object form using the following steps:

- Open the design mode of the object (e.g. CCCMYOBJECT) where you wish to display the table.
- Click on the **"Configuration"** button and move to the tab **"User Defined Tables"**
- Click on "Tables" drop down list and select the child table (Figure C3.4).
- Select the child table from the left panel and then in the **"Locate statement"** property enter the field expression that will be used to locate the data of the table. The locate statement expression may contain parent field values (:Field), user values (e.g. 51) or login parameters (e.g. :X.SYS.COMPANY). Syntax is the following:

ChildField1;ChildField2;ChildField3=:ParentField1;;ParentField2;Value1

Example: REFOBJID;SOSOURCE;XDOCTYPE=:PRJC;40;1

Note: If the link field of the parent-child relationship uses the same name in both parent and child tables, then you simply enter the name of the field inside the "Locate statement" property (Figure C3.4).

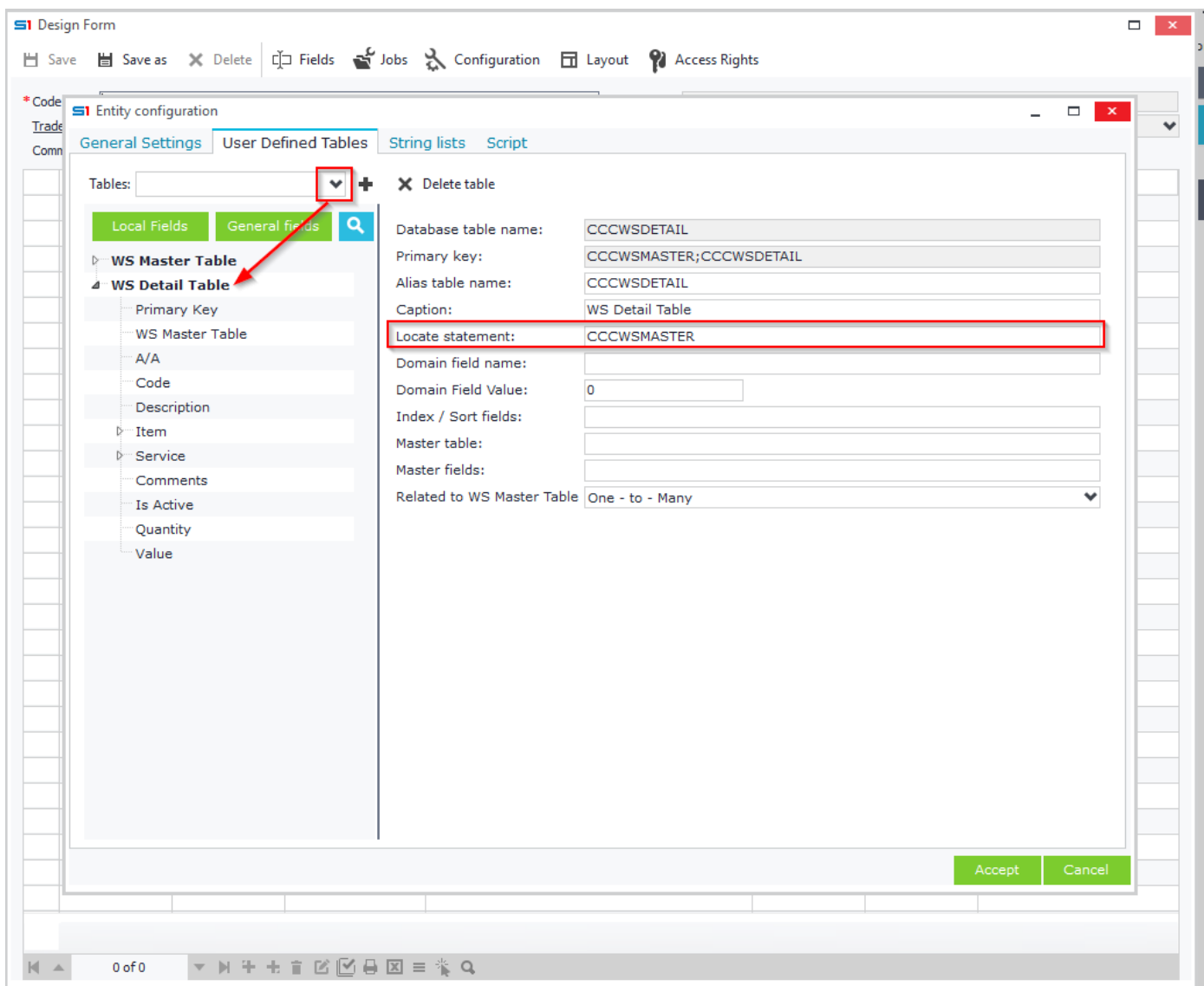


Figure C3.4

- The next step is to store the value of the parent table primary key in the corresponding child table field. Select also the fields mentioned in the **"Locate statement"** property and enter the appropriate values in the **"Forced value"** property so that they meet the expression of the locate statement.

Figure C3.5 stores the primary key value CCCWSMASTER of the parent table CCCWSMASTER in the field CCCWSMASTER of the child table CCCWSDETAIL.

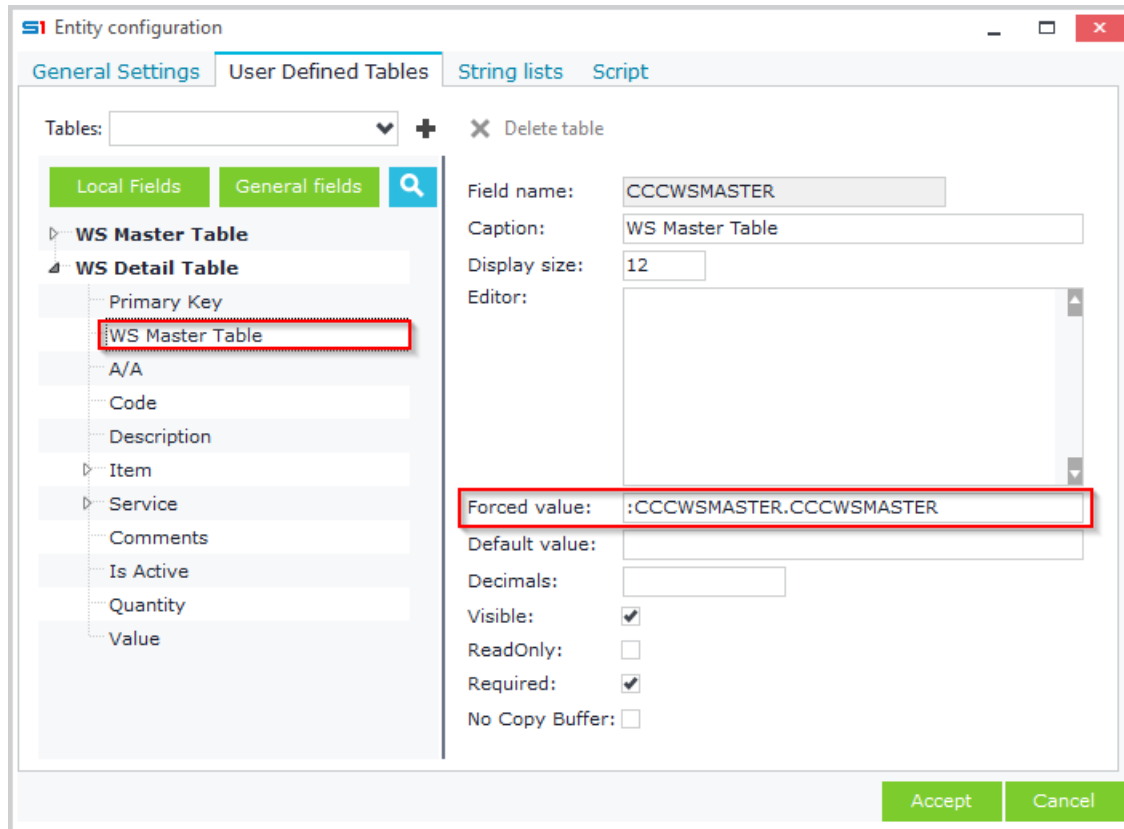


Figure C3.5

- Last step is to display the table inside the form using a grid control.
Right-click, create a new grid and then drag and drop the fields you want to be displayed (Figure C3.6)

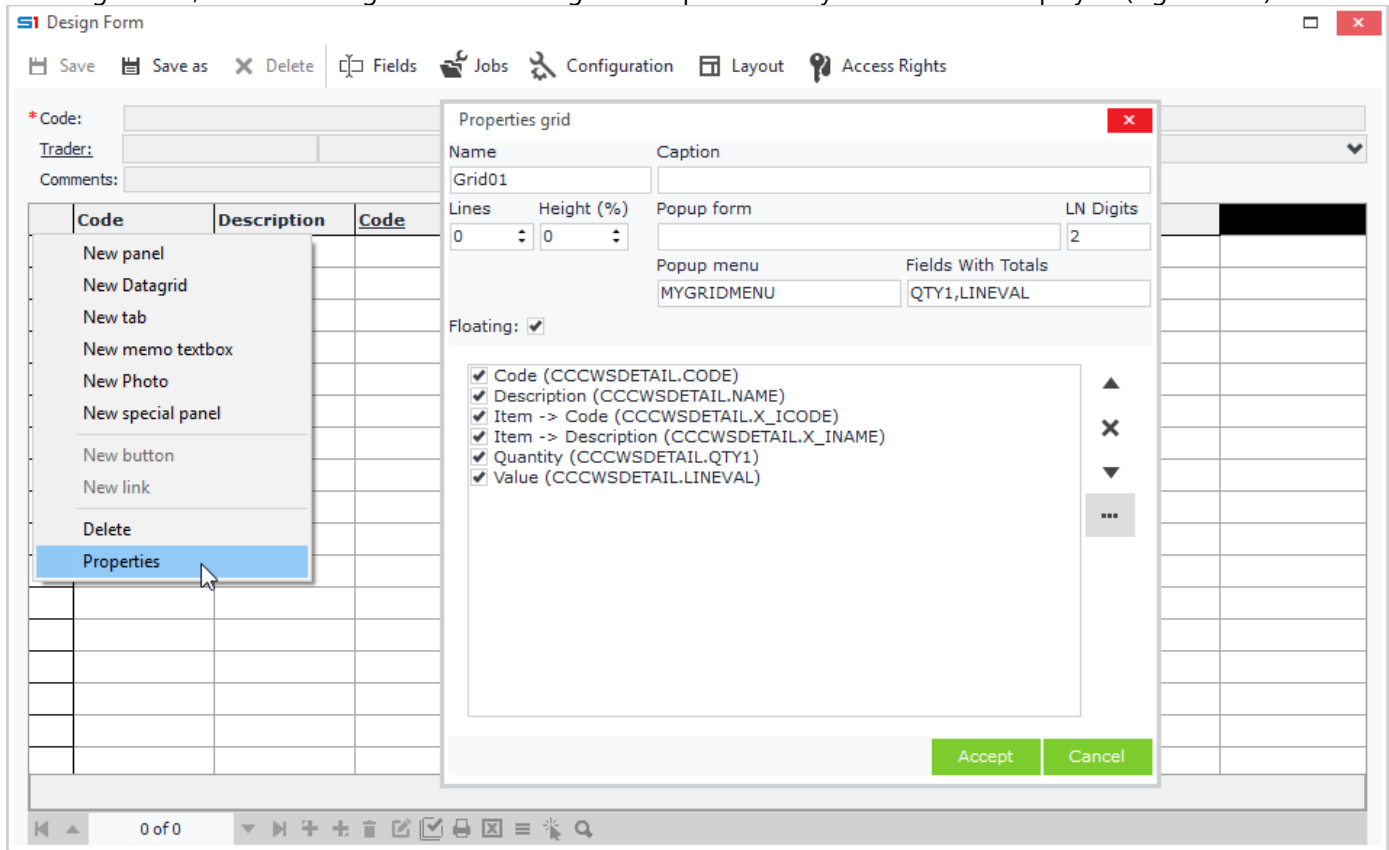


Figure C3.6

C.3.3 Performance Tips

It’s better not to use identity fields in child tables, because they are not processed very fast from database servers. Instead, you can use integer fields as primary keys that are auto filled with incremental numbers, following the steps below:

- Insert a new field in your parent table:
 - **Name:** LOCKID
 - **Data type:** Integer
 - **Preselected Value:** 0
- Create a primary key field in the child table using any name and add the following command:
:ParentTableName.LOCKID in its Forced Value property.

Parent table fields named **LOCKID** are responsible for filling automatically child table fields with incremental numbers, using an internal SoftOne process.

Figures C3.7 and C3.8 show the primary key field MTRLINES of the child table MTRLINES that uses the above command in the object Sales documents.

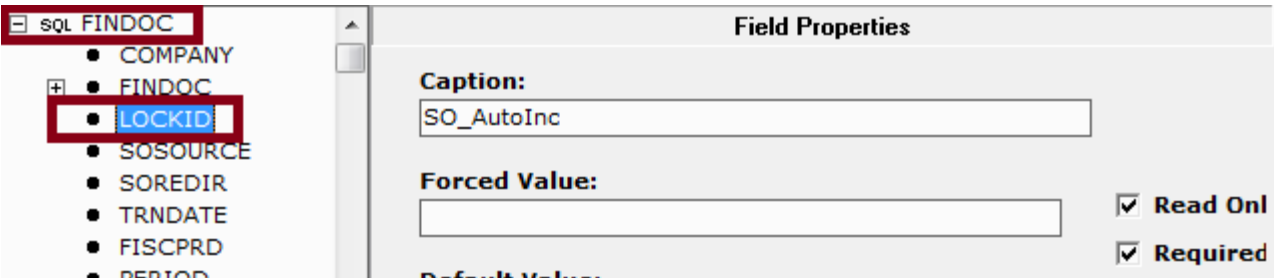


Figure C3.7

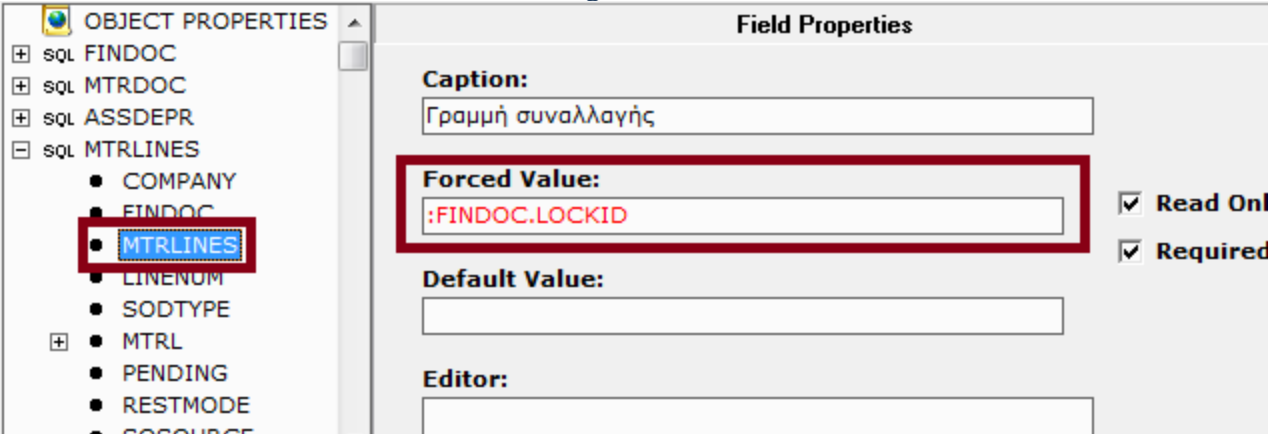


Figure C3.8

D. String Lists

String Lists represent a collection of keys and values. A key-value pair is an entry of a string list, where keys are unique. String lists are used as editors to numeric fields (integers, small integer) and provide fast lookups. An example of a field that uses a string list as selector, is the field “Tax Category” in the Customers entity, which displays the values “Normal, Is Exempted, Reduced” (Figure D1).

The main difference of tables and string lists is that string list data cannot be modified during runtime. They also display only two fields, and their data are always loaded in memory.

String lists can be created either from SoftOne designer or through the configuration window of an object form. The ones that are created through SoftOne designer can be used from any object of the application, while those created inside an object can only be used from fields within the current object ([Chapter 1 - A5. String Lists](#)).

The screenshot shows the 'Customers' form in SoftOne Designer. The form is divided into several sections: 'General data', 'Contact Info', 'Business Info', 'Sorting', 'Invoicing data', 'Transaction data', and 'Accounting'. The 'Tax Category' field is highlighted with a red box, and its dropdown menu is open, showing the following options: 'Normal', 'Is exempted', and 'Reduced'.

Figure D1

A string list is created in SoftOne Designer using the following steps:

- Right click the left panel of SoftOne Designer and select “**New string list**” from the pop-up menu.
- Enter the name of the string list in the textbox “**Name**”.
- Inside the “**Key**” column enter the key codes
- Add values for each key in column “**Value**”. Those values will be displayed in fields that use this string list in their editor property (Figure D2).

The screenshot shows the 'S1 Designer' window with the 'String Values' editor. The 'Name' field is set to 'DocumentLevel'. The table below shows the key-value pairs for the string list:

Nr.	Key	Value
1	0	Init Level
2	1	Middle Level
3	2	Last Level

Figure D2

String lists can be used as selector lists from numeric fields of any object, using the following steps (Figure D3):

- Design a form of an object (e.g. Sales Documents)
- Click on **"Entity Configuration"**
- Select a numeric field and enter the name of the string list prefixed by \$ (e.g. \$DocumentLevel)

In runtime the field will be displayed as in Figure D4.

Analysis of creating a string list inside a form can be found in [Chapter 1 - A5. String Lists](#).

String lists created inside forms are used in Field Editors without using the symbol \$, but only the name of the stringlist.

Note: All **SoftOne internal string lists** (e.g. SODTYPE, SOSOURCE) can be overridden through Soft1 Designer, if you create an identical string list – with the same name (e.g. SODTYPE) and add new values (e.g. 5001:MyType). These values will then be available at runtime (in fields that use the above string list as editor). Use this option carefully!

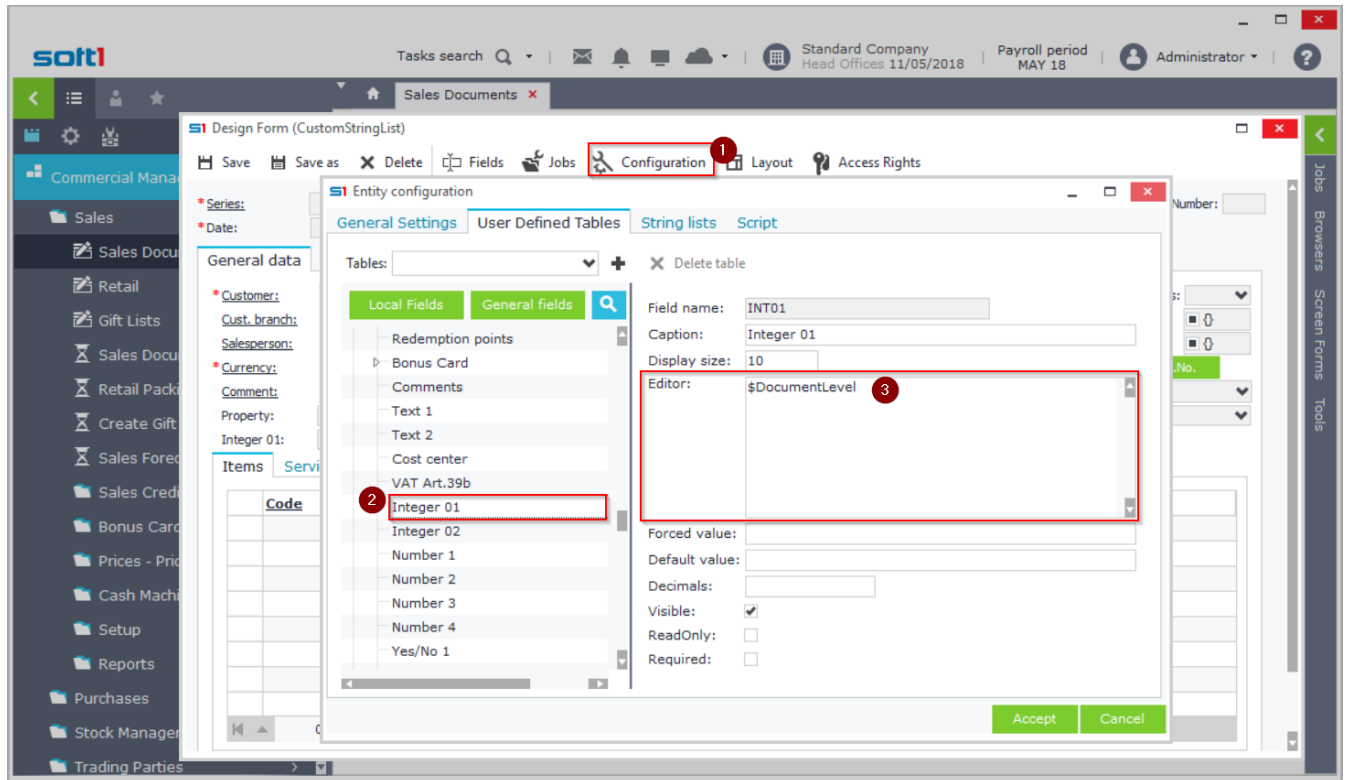


Figure D3

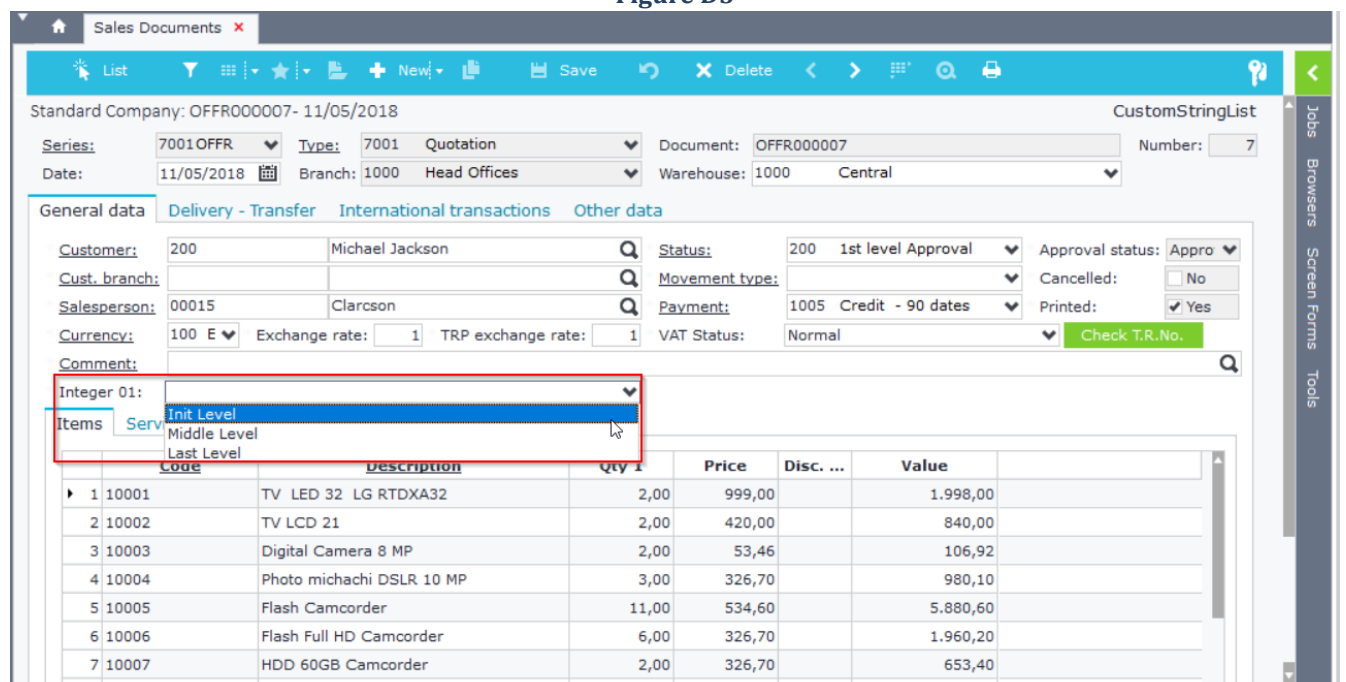


Figure D3

E. Database Views

Database Views are stored in database and can be used in the following cases:

- As editors in lookup fields (selectors), displaying data from SoftOne tables or tables from another database.
- As child “tables” of object forms in order to display entries from various tables.
- In custom object browsers for displaying data in report style with redirections to other objects (e.g. Sales Statistics).

E.1 Design

A database view is created using the following steps:

- Right click the left panel of SoftOne Designer and select “**New database view**” from the pop-up menu.
- Inside the textbox of the tab “**SQL**” enter the SQL statement and then click on the button “**Create Fields**”, to create the fields (Figure E1). Configure fields by entering titles, editors, etc.

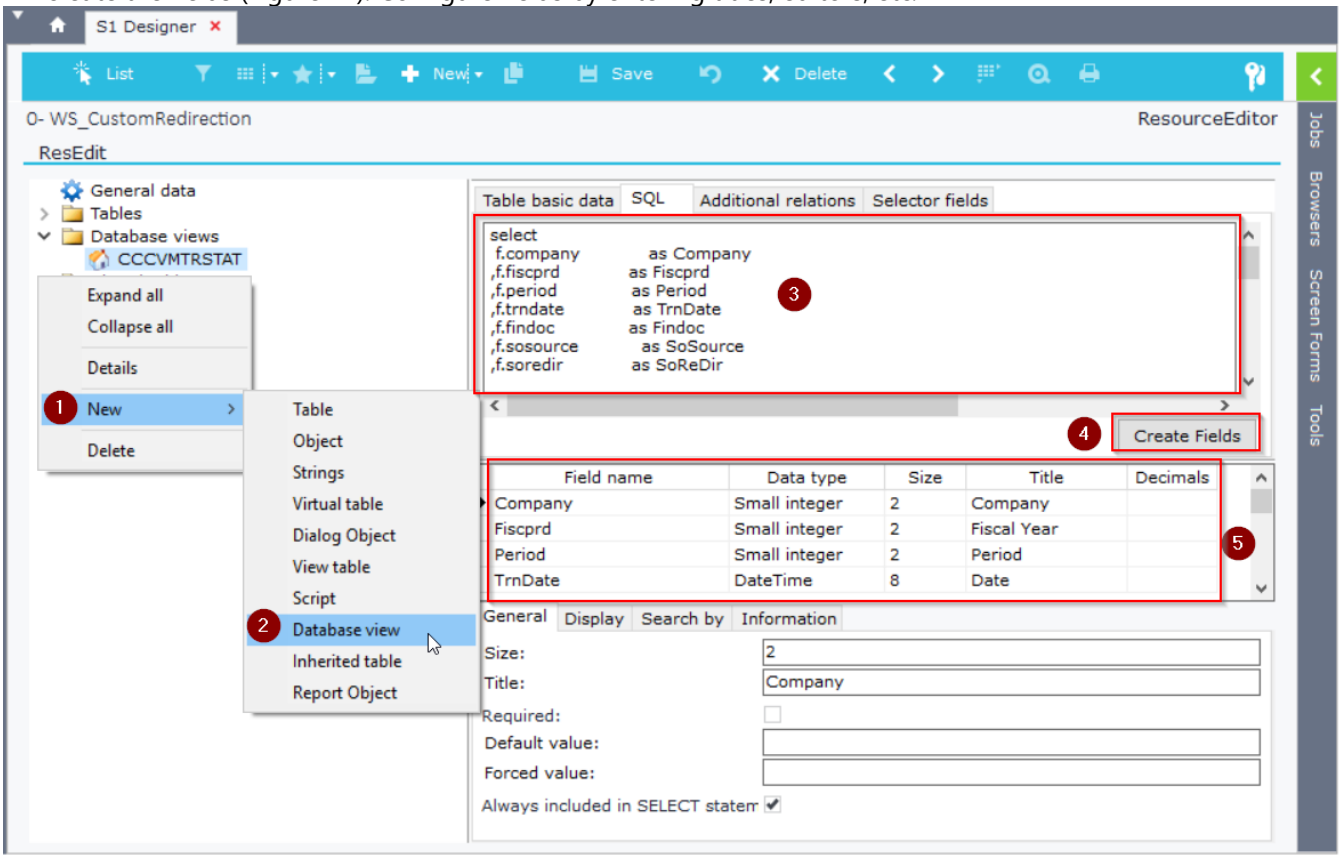


Figure E1

- The property “**Primary Key**” of tab “**Basic Data**” may be used in two different ways, either to locate a record when used as a child table of an object or to return a value when used as an editor from a selector field. In both ways you need to enter here the name of the fields separated by semicolons (Figure E2).
- Finally, in Tab “**Selector fields**” insert the fields that you wish to display if the specific database view will be used as lookup table in field editor (Figure E3).

After completing the creation of the database view the database must be synchronized.

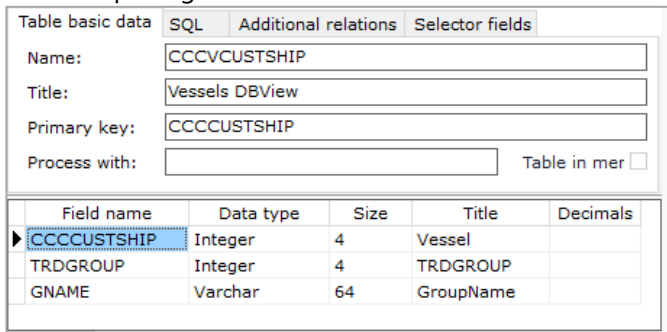


Figure E2

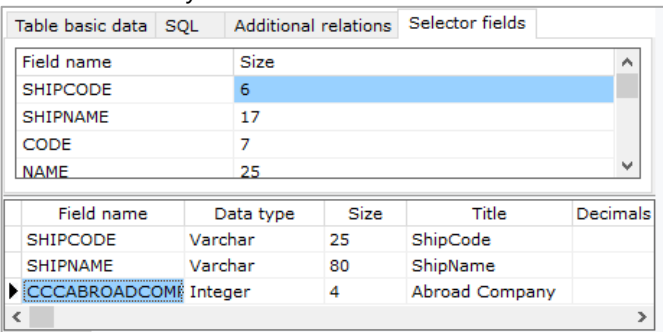


Figure E3

E.2 Advanced Browser Redirection

Database views can be used in objects in order to display data in browsers and pivot tables. This means that when users double click on a record/line of the browser they can be redirected to a specific record of an object. The expression formula for browser redirection to different objects is **#ObjectID;RecordID** and is fully explained in section [Browser Redirection](#).

In this section we will display the way of defining the redirection objects along with different forms and lists using script code.

In the example of the section [Browser Redirection](#) we created a database view that redirects the user to different objects (Sales, Purchases, etc.) according to the values of the fields SOSOURCE and SOREDIR. Suppose that we needed to redirect the user to specific forms of the above objects depending on the value of the field SERIES. In this case we could write a JavaScript code that checks the series (or anything else we prefer to check) and then return a command that opens the object in a specific form (and list). There are two way to express the command, either using the object name: **#ObjectName,FORM=xxx,LIST=xxx** or the object id: **#ObjectID,FORM=xxx,LIST=xxx**

The JavaScript function that returns the object command can be written inside the **"S1 Import"** tool (Tools – SDK – Customization tools – S1 Import). Create a new a record, type a code (e.g ReturnFindocForm) and description and then enter the JavaScript of Figure E4. The function RUN (can be any name) returns the object command that will be used for the redirection and it will be later called from the property **"Process with"** of the database view.

The script initializes a variable "vEditor" that will hold the object command (returned in line 32). Then it checks for the value of the field SOSOURCE (line 5) and also the SERIES value (lines 7,10) so as to create the command that redirects to different object forms per SERIES. In line 13 it uses the expression SOSOURCE+SOREDIR (as in [Browser Redirection](#) example) that redirects to the object using the default form of the user.

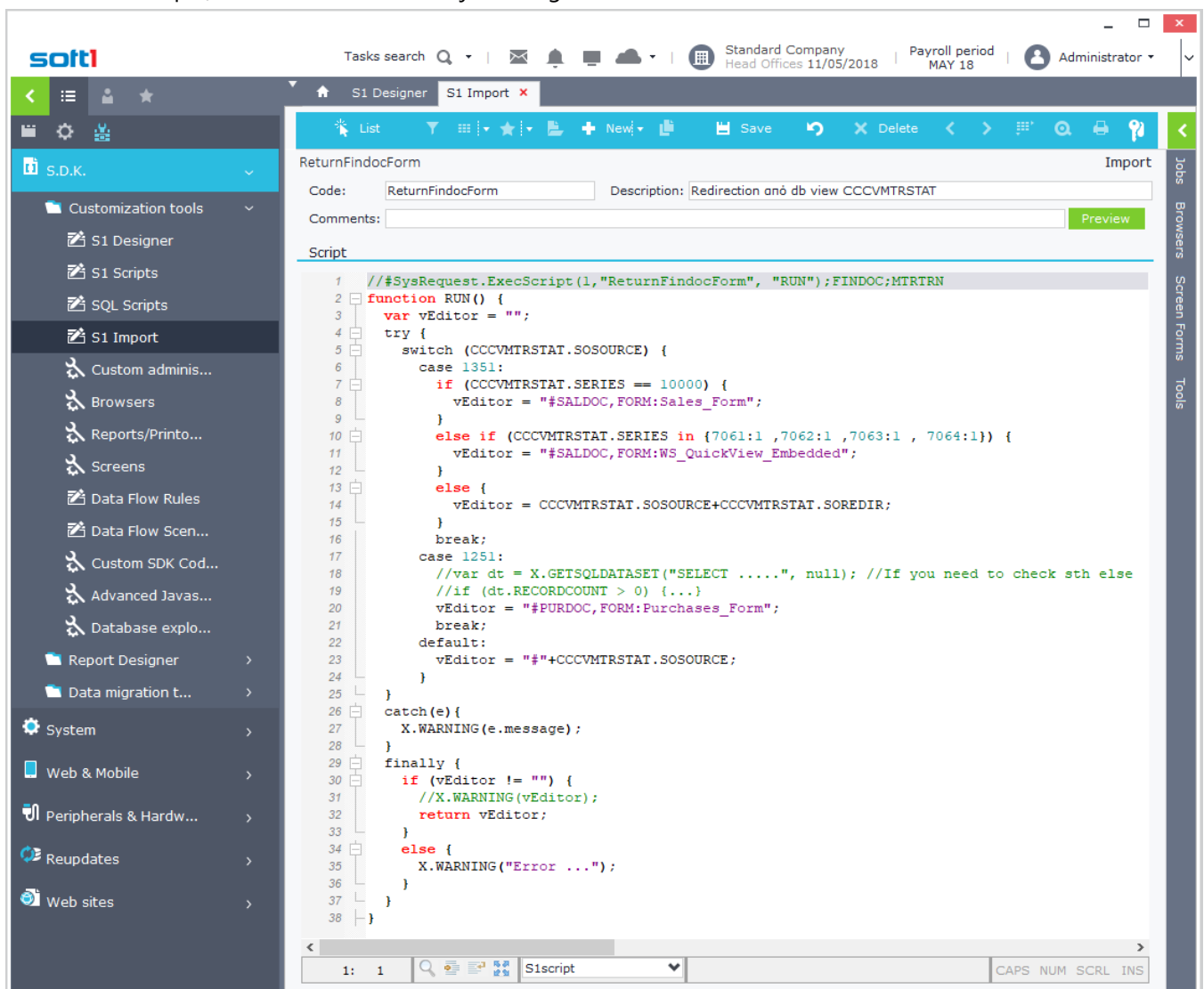


Figure E4

In SoftOne Designer, select the view and enter the following expression in the "**Process with**" property (Figure E5).

#SysRequest.ExecScript(1,"ReturnFindocForm", "RUN");FINDOC;MTRTRN

The part **#SysRequest.ExecScript(1,"ReturnFindocForm", "RUN")** is the one that executes the JavaScript function "**RUN**" of the "**S1 Import**" record with code "**ReturnFindocForm**". The other part is the record id of the redirection.

Figure E6 shows the database view at runtime and how it redirects to specific form per SERIES value.

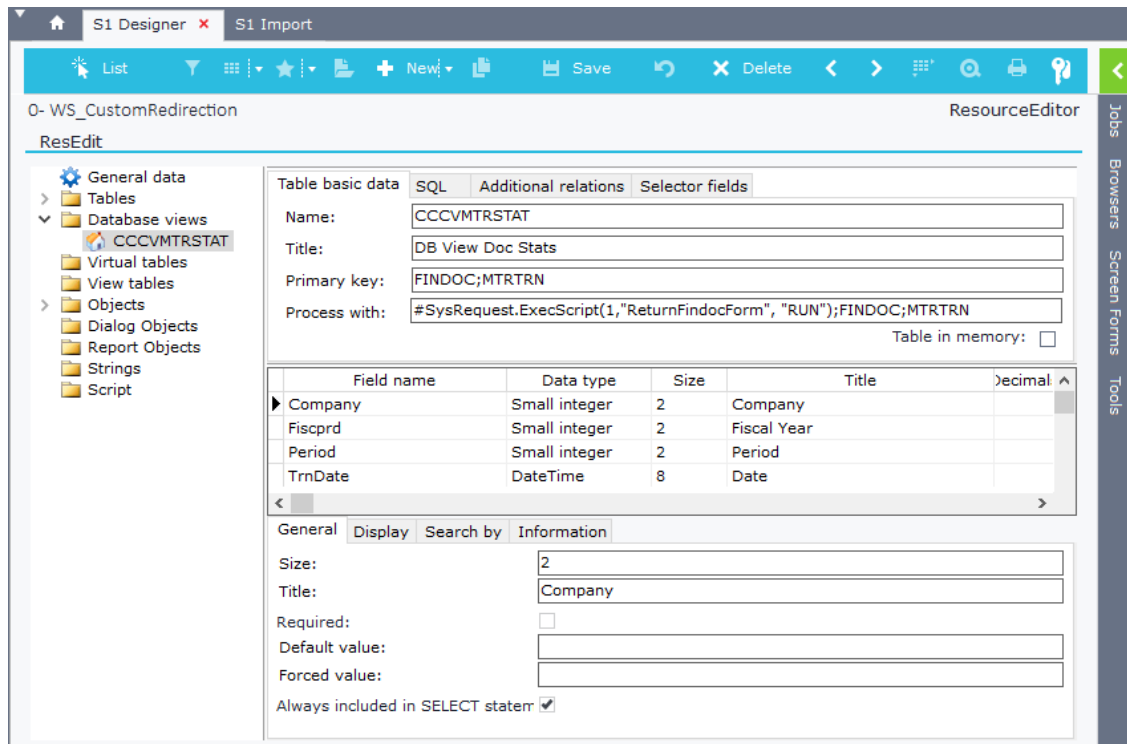


Figure E5

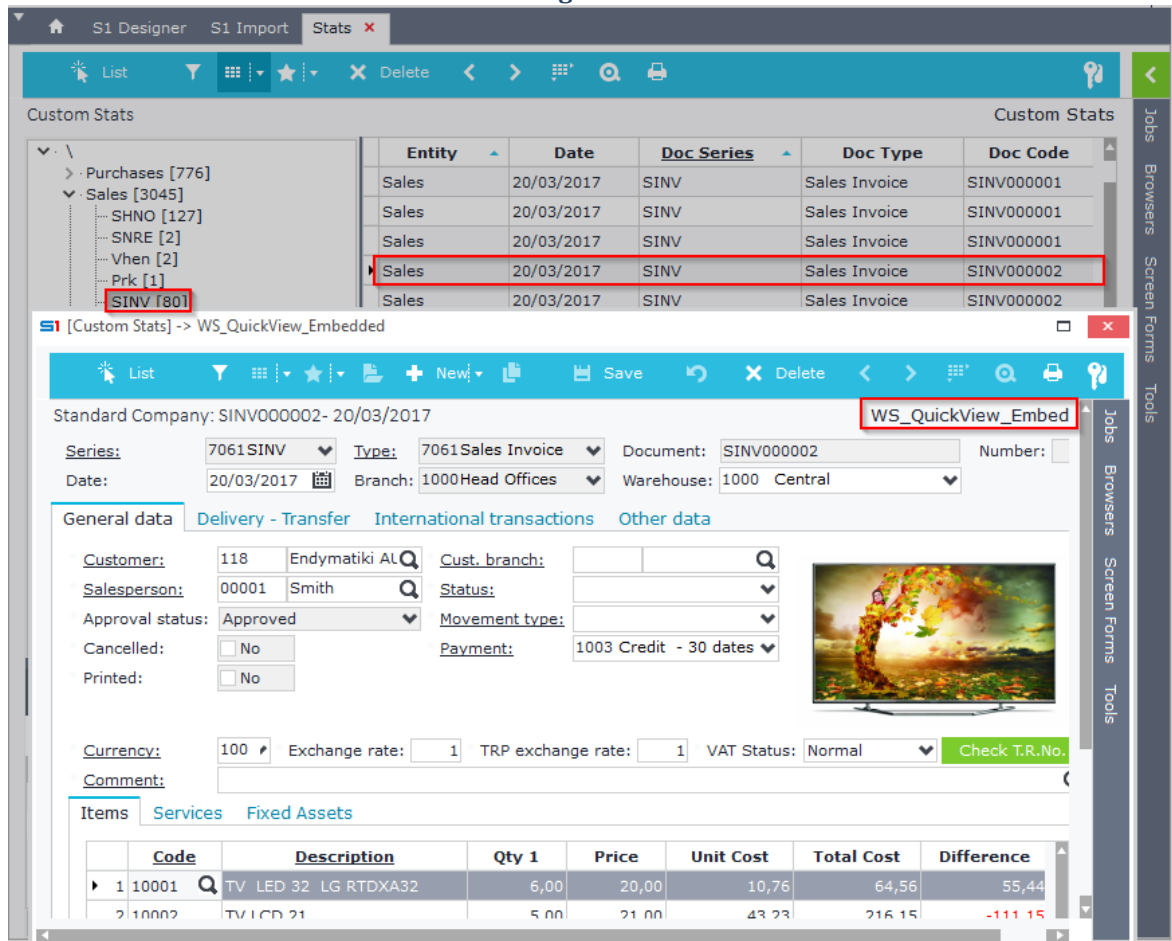


Figure E6

E.3 Examples

E.3.1 Editor in lookup fields (Selectors)

Create a database view that will be used as lookup list in field MTREXTRA.NUM01 of Items. The lookup list will display the code and the names of customers, as well as some financial data from table TRDFINADATA and return the id (TRDR) of the selected customer.

- Create a database view named **CCCViewCustomer** with primary key **TRDR**
- Enter the following SQL statement inside the tab "SQL" and click on the button "Create Fields" (Figure E3.1):

```
SELECT
    A1 . TRDR
    A1 . CODE
    A1 . NAME
    A1 . COMPANY
    $NVL (A2 . LBAL , 0) AS LBAL
    $NVL (A2 . LTURNVR , 0) AS LTURNVR,
FROM
    TRDR      A1 ,
    TRDFINDATA A2
WHERE
    A1 . TRDR  =A2 . TRDR
AND
    A1 . COMPANY=A2 . COMPANY
AND
    A1 . SODTYPE=13
```

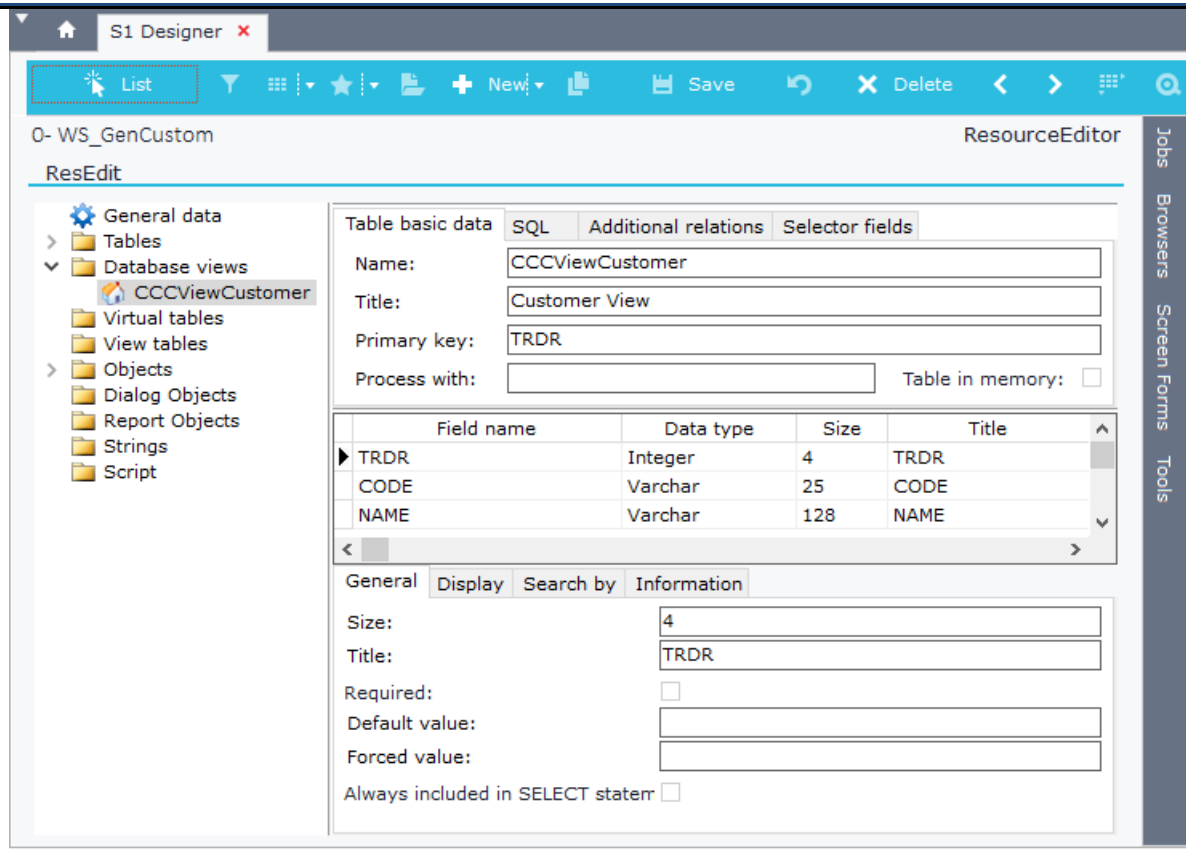


Figure E3.1

- Add the selector fields as in Figure E3.2, change the S1 Designer version, save and synchronize the database.

Field name	Size
CODE	7
NAME	20
LBAL	9
LTURNVR	9

Figure E3.2

In "Items" Configuration window, select NUM01 of "Item extra fields" and enter the name of the database view "**CCCViewCustomer**" in the Editor property. (Figure E3.3)

The screenshot shows the 'Entity configuration' window with the 'User Defined Tables' tab selected. On the left, under 'Item extra fields', the 'Number 01' field is selected. The right pane shows the configuration for 'NUM01':

- Field name: NUM01
- Caption: :X.ITEPPRMS.NUM01:15
- Display size: 14
- Editor: CCCViewCustomer
- Forced value: (empty)
- Default value: (empty)
- Decimals: (empty)
- Visible: ☒
- ReadOnly: ☐
- Required: ☐

At the bottom right are 'Accept' and 'Cancel' buttons.

Figure E3.3

The field will be displayed at runtime as in Figure E3.4.

The screenshot shows the 'Stock Items' configuration window at runtime. The 'General data' tab is active, showing fields for 'Text' and 'Number'. The 'Number' field is expanded, showing a table of data. The table has columns: CODE, NAME, LBAL, and LTURNVR. The data is as follows:

CODE	NAME	LBAL	LTURNVR
310	3rd District General Hospital	8.529,9	6.130
121	Alexis Chalkidis	-2.287,7299999999	14.440,75
304	Anagnostopoulos Ioannis (potential cust)	9.390,9	6.878,78
115	Anastasios Kontomari	1.308.733,60000000	1.097.669,59000000
281	Anastasios Martonis	34.324,6800000000	72.022,5
238	Andreas Petrides	-309.934,67999999	50.242,2000000000
248	Apostolos Panagiotou	4.366,35	4.226,3
112	Apple AE	101.684,82000000	89.134
184	Athena Kollias	347.103,58999999	281.404,47999999

Below the table is a 'Clear filters' button. To the right of the table are four 'Yes/No' checkboxes labeled 'Yes/No 1' through 'Yes/No 4', all set to 'No'.

Figure E3.4

E.3.2 Child Database View in Object

In Customers' form, display CRM actions inside a datagrid.

- Create a database view named CCCTRDRACTION with primary key TRDR and enter the following sql query:
- Click on "Create Fields" (Figure E3.5)

```
SELECT
    A.SOACTION,
    B.TRDR
FROM
    SOACTION A
LEFT OUTER JOIN
    TRDR B
ON
    A.TRDR=B.TRDR
```

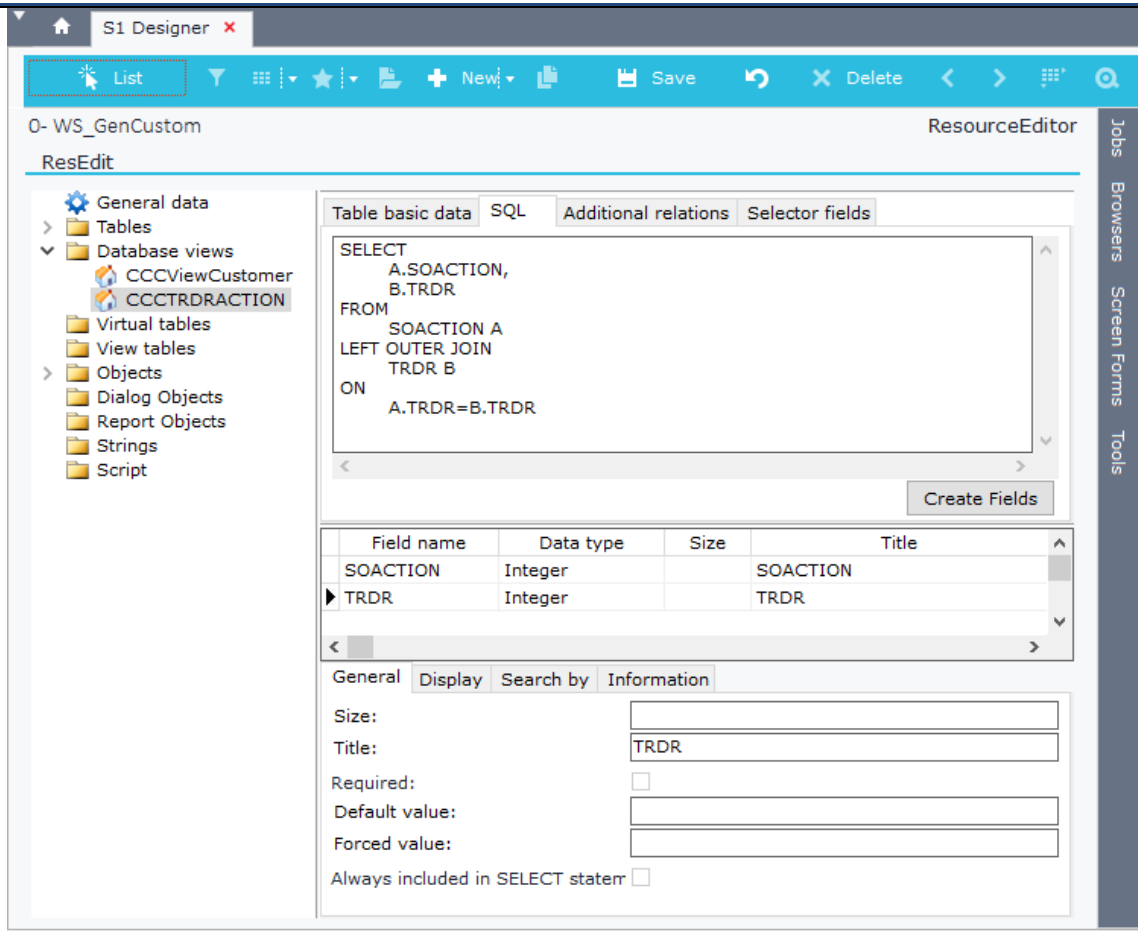


Figure E3.5

- Enter titles, display sizes in fields, change S1 Designer version, save the changes and synchronize the database.
- Inside Customer form Configuration window, select and add the view as in Figure E3.6.

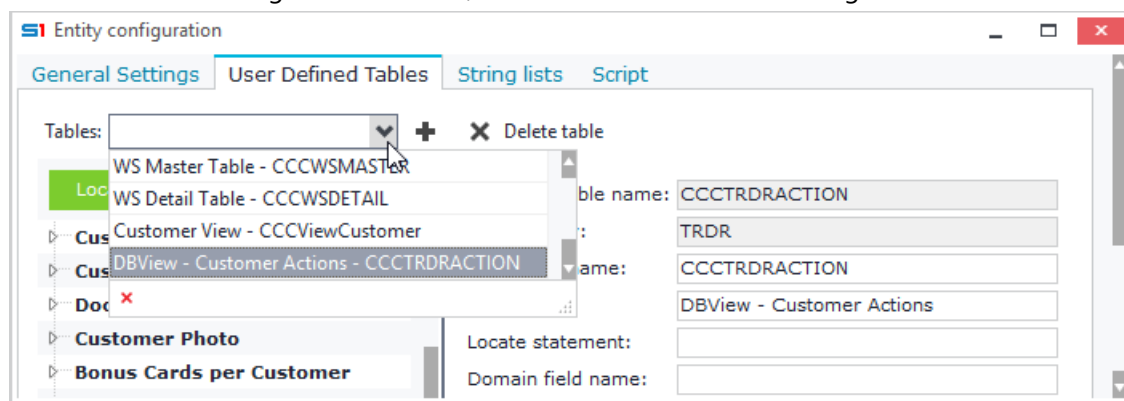


Figure E3.6

- Select the view (left panel) and set the locate statement, use either TRDR or TRDR=:TRDR.TRDR (Figure E3.7).

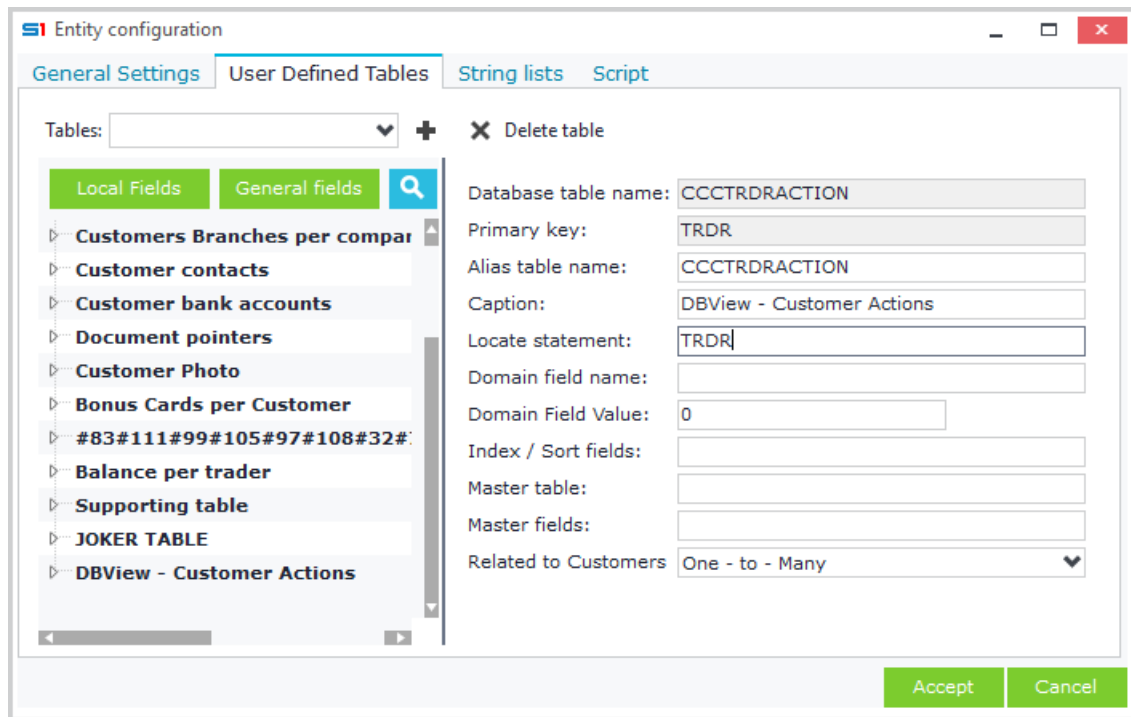


Figure E3.7

- SOACTION fields are displayed (linked) when you enter the editor SOACTION inside the field "CCCTRDRACTION.SOACTION" of the view (Figure E3.8). Notice that the table is linked to the field after you Accept the changes and reopen "Configuration".

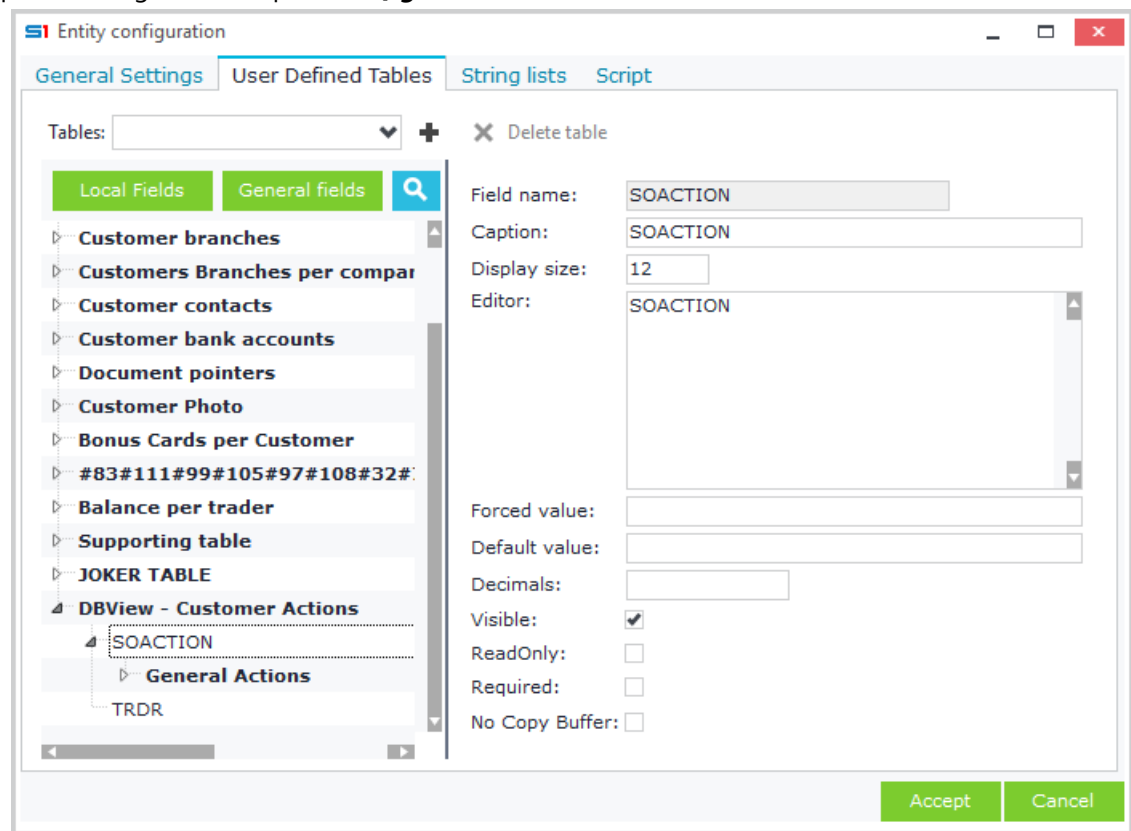


Figure E3.8

Chapter 6 – Database Designer

- Create a new tab and a new datagrid and drag and drop fields from the action table of the database view (Figure E3.9).

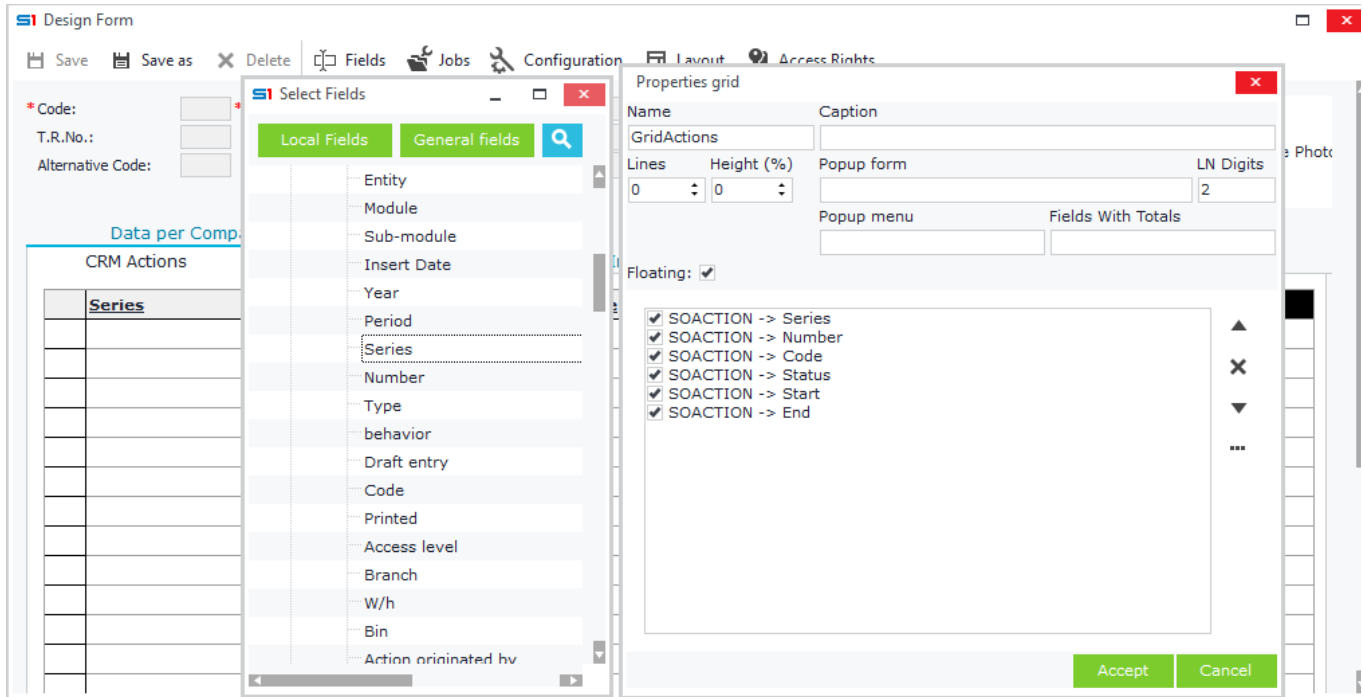


Figure E3.9

The datagrid will be displayed at runtime as in Figure E3.10.

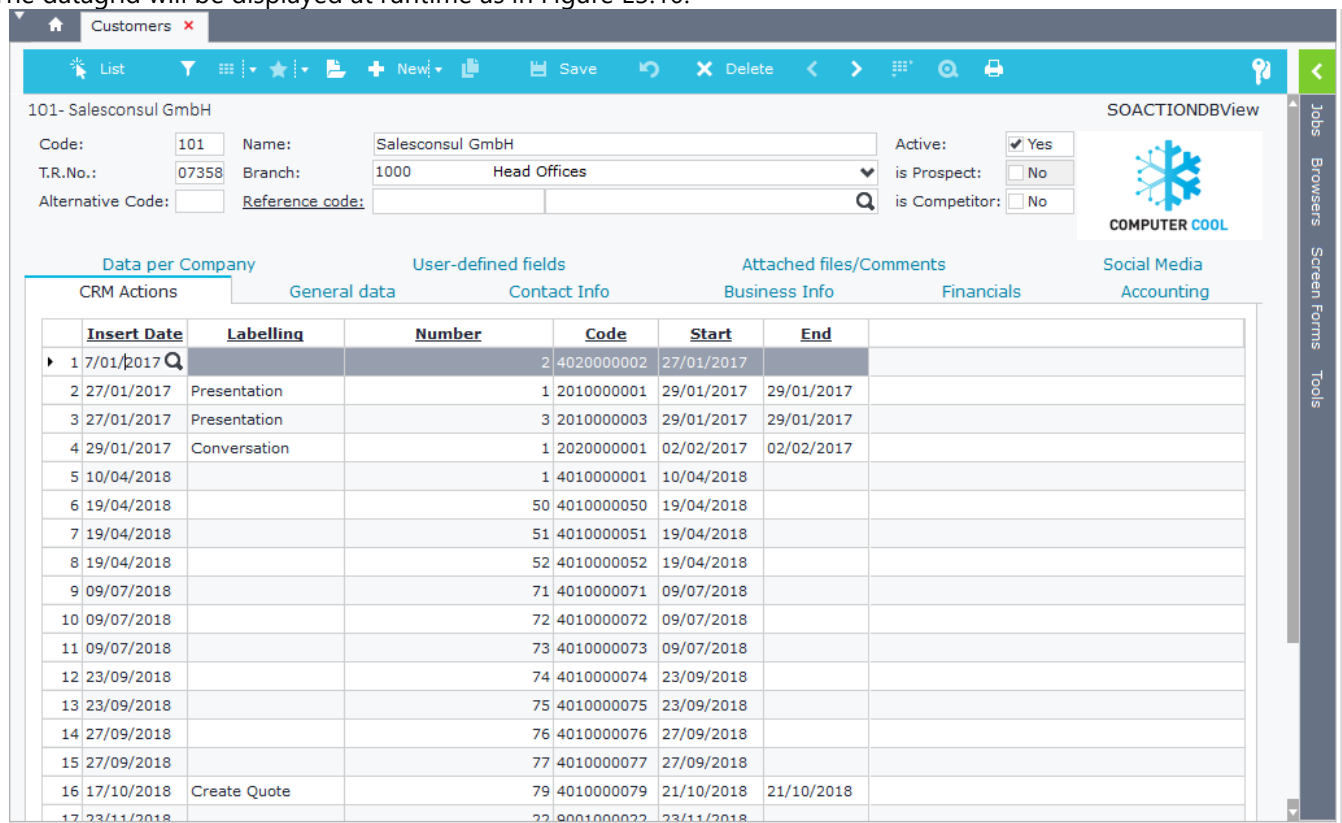


Figure E3.10

F. Objects

Business Objects are used for creating a user interface where data from tables, virtual tables, database views etc. can be processed. They function exactly as SoftOne objects (Customers, Sales documents, etc.), which means that they include all the tools that can be found in an object (EDA, Imports, Browsers, Forms, Printout Forms, etc.). Database objects (tables, database views, etc.) can be used inside form objects, as long as they are designed through SoftOne designer.

F.1 Design

Objects are created through the right click option **"New – Object"** of SoftOne Designer (Figure F1.1). Object names must use the prefix CCC.

Database objects (tables, database views, etc.) can be inserted through drag and drop (Figure F1.2). If you need to lock the focus of the object then use the button **"Lock Tree"**.

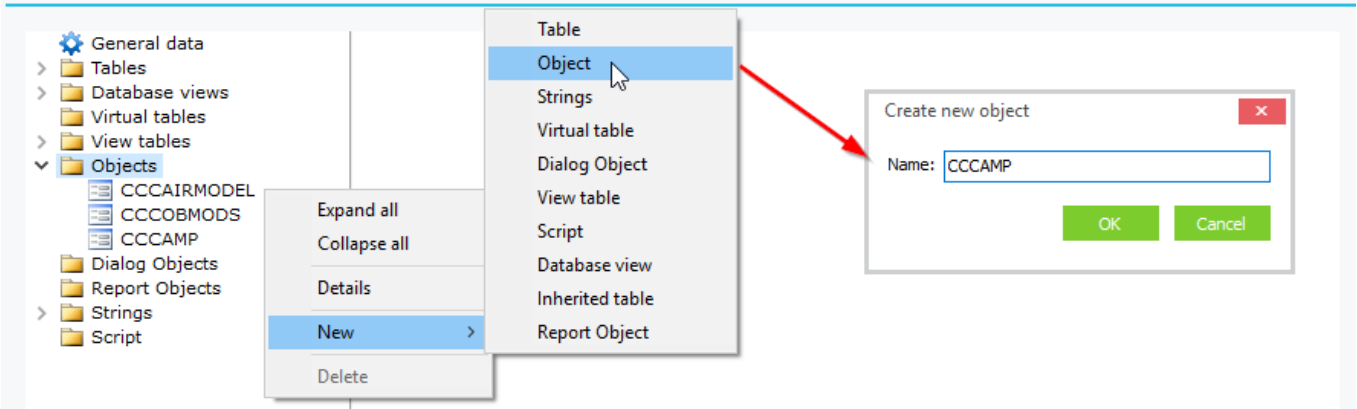


Figure F1.1

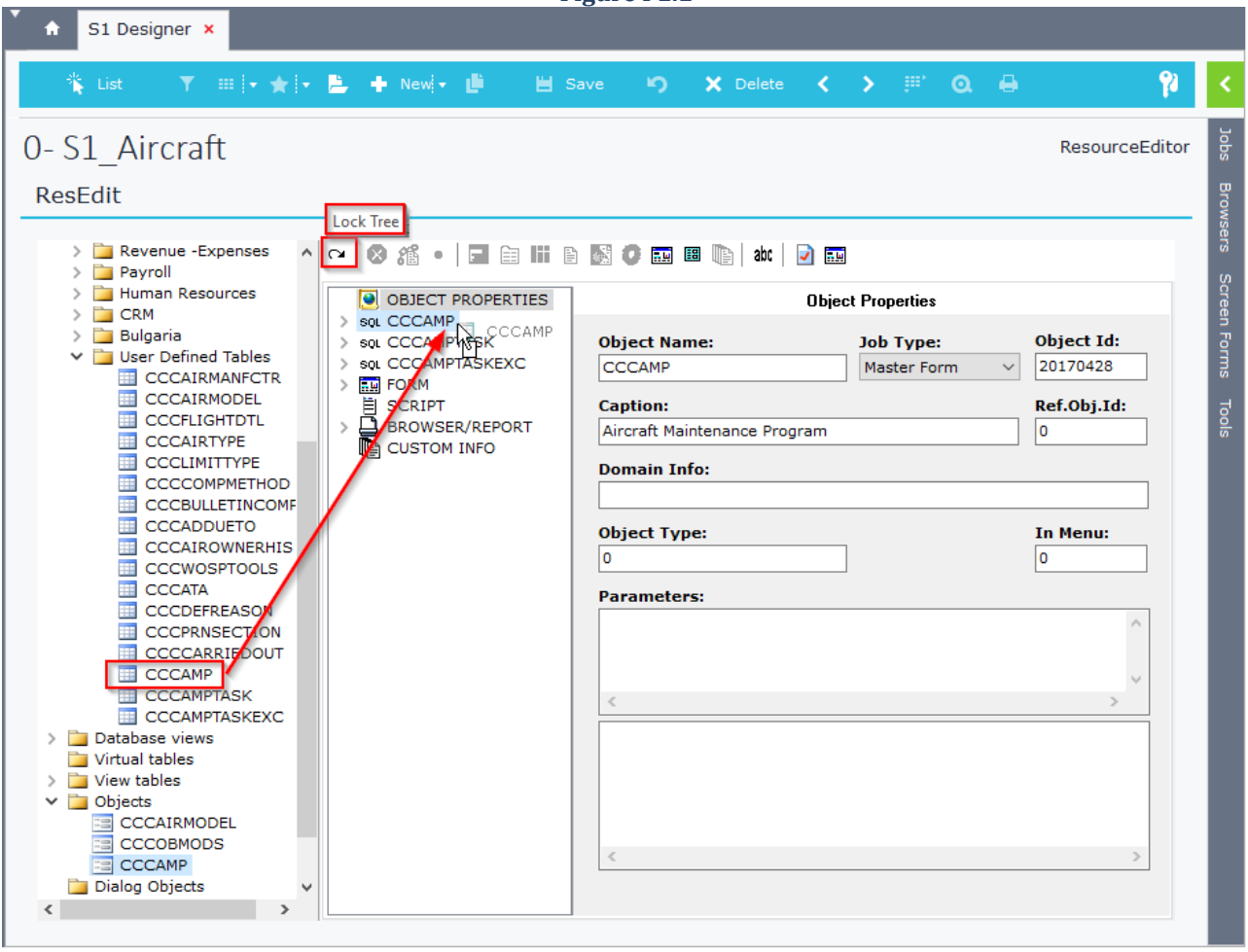


Figure F1.2

F2. Object Properties

Object properties are the following (Figure F2.1):

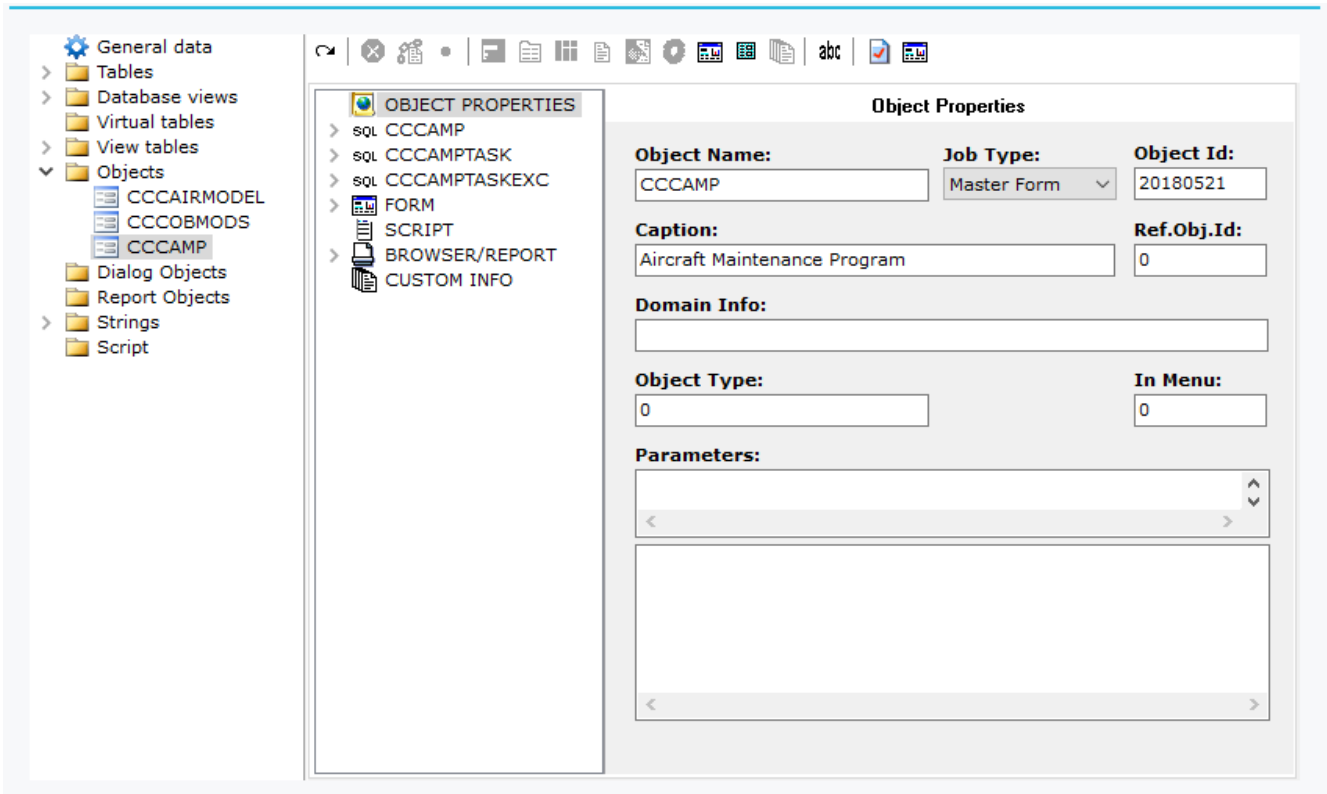


Figure F2.1

Object Name

Object name using CCC prefix.

Job Type

- **Master Form**
Creates a form object that uses all the available tools (Browsers, Forms, EDA, etc.)
- **Edit List**
Creates an edit list object that uses simple toolbar controls (Save, New, Delete). Data are displayed either in a datagrid (e.g. Customer Professional categories) or in a single record form using the command EDITOPTIONS=ONEROW (e.g. Customer Parameters).

Object ID

Declares the ID of the object that can be used from as a reference from other tools. For example printout forms that refer to this object are created through the menu command Templates.ObjectID. This ID can also be used in hyperlink redirection, as long as you have used it inside the table's "Process with" property.

Caption

Name that will be displayed in the default object form.

Ref.Obj.Id

Defines the reference object that displays your custom object in its "Relative jobs". For example, if you enter the number 51, then your object will be listed in the relative jobs of items. Object ids (SOSOURCE & SODTYPE) can be found in Annex [SODTYPE](#) and [SOSOURCE](#) tables.

Domain Info

Declares the field whose value is used to distinguish between different object records. Helpful when you want to create only one interface (object) and use it for different record types.

For example, there is only one object created for table AREAS, but there are different menu jobs for customers (SODTYPE=13), suppliers (SODTYPE=12), etc. The Domain Info property of the AREAS object is SODTYPE=SODTYPE(12,13,15,16,20,21) and the different objects created in the menu have the command AREAS.xxx, where xxx is the SODTYPE value. That is, for customer areas the menu job command is AREAS.13. Moreover, every new record adds the value of the menu job command (13) in the field SODTYPE, that is defined in Domain Info property of the main object.

Parameters

The following table displays the various commands that can be used as Object Parameters (Figure 2.2).

Note: These commands can also be used in custom forms ("General parameters" tab of "Configuration" window).

Object Parameters	
Command	Operation
AUTOLOCATE=id	Locates the record using the given id
BRMENU=StringList	Adds the StringList records in the Browser context menu
BROWSERONLY=1	Displays object only as a browser (e.g. SalesStatistics)
BUFEXCLUDE=S1FIELD1;S1FIELD2;	The SoftOne fields defined are excluded from Record Copy
CUSTOMBUFEXCLUDE=CCCFIELD1;CCCFIELD2;	The custom fields defined are excluded from "Record Copy"
CUSTOMBUFEXCLUDE=TABLENAME	All the fields of the defined table are excluded from "Copy"
NOGRIDTOOLBAR=1	Hides the toolbar of all the grids inside a form.
NOTOOLBAR=1	Hides the main toolbar of the given object.
NOSIDEBAR=1	Hides the sidebar (browsers, forms, tools) of the given object.
EDITOPTIONS=NOINSERT	Prohibits adding a new entry and hides the toolbar button "New"
EDITOPTIONS=NODELETE	Prohibits entry delete and hides the toolbar button "Delete" Example: EDITOPTIONS=NOINSERT,NODELETE
EDITOPTIONS=READONLY	Disables editing the data of an entry
EDITOPTIONS=NOBROWSER	Hides the browser of an object and the toolbar button "Browser" Example: EDITOPTIONS=NOBROWSER,NOINSERT,NODELETE
EDITOPTIONS=ONEROW	Displays an EditList object in "one record" mode
EDITOPTIONS=NOPRINTFORM	Disables Form printing and hides the toolbar button "Print Form"
EDITOPTIONS=NOPRINTBROWSER	Disables Browser printing
EDITOPTIONS=NOPRINTLABEL	Disables label printing (from browser right click menu)
PARENT=ObjectName (e.g.PARENT=SALDOC)	Inherits the properties of an Object (Tables, Script, etc.)
RELJOBS=StringList	Adds the StringList records in the Related Jobs of the Object

The screenshot shows the 'Object Properties' dialog box for an object named 'CCCMYOBJECT'. The left pane shows a tree view with categories: SQL, FORM, SCRIPT, BROWSER/REPORT, and CUSTOM INFO. The main area contains the following fields:

Object Name:	Job Type:	Object Id:
CCCMYOBJECT	Master Form	20000
Caption: My object		Ref.Obj.Id: 0
Domain Info:		
Object Type: 0		In Menu: 0
Parameters: EDITOPTIONS=NOINSERT,NODELETE		

Figure F2.2

The screenshot shows the 'Object Properties' dialog box for an object named 'CCCS1MVPRG'. The left pane shows a tree view with categories: SQL, FORM, SCRIPT, BROWSER/REPORT, and CUSTOM INFO. The main area contains the following fields:

Object Name:	Job Type:	Object Id:
CCCS1MVPRG	Master Form	20211221
Caption: Prg		Ref.Obj.Id: 0
Domain Info:		
Object Type: 0		In Menu: 0
Parameters: BRMENU=BRMENU BUFEXCLUDE=CCCS1MVPRGADS CUSTOMBUFEXCLUDE=CCCS1MVPRGLNS.REGPRICESP;CCCS1MVPRGLNS.REGPRICESF;		

Figure F2.3

F.3 Table Properties

Database objects (Tables, DB Views, etc.) used in SoftOne objects have the following properties (Figure F3.1):

Figure F3.1

Caption

Auto filled with the title of the table

Original Name

Auto filled with name of the table

Filled with Data from

Indicates the way table records are populated

- **SQL:** Use in Form objects (e.g. Customers, Sales documents, etc.).
- **Report:** Use in Report objects (e.g. Customer Balance report).
- **Dialog:** Use in Dialog objects (e.g. Customers reupdate) or filter tables in Report objects (e.g., Customer Balance report).

Locate

Defines the fields that will be used to locate the data of this table according to the parent table primary key. Parent table field data are defined using a colon following the name of the field (:FIELD). You have the option to use only the name of the linked field, if its name is the same in both tables (parent – child). Multiple fields and values are separated with semicolons (Figure F3.2).

The main syntax for the locate statement is:

CHFIELD1;CHFIELD2;CHFIELD3;CHFIELD4=;PRFIELD1;;PRFIELD2;Value1;Value2

For example, if the name of the parent table field is TRDR and the linked child field name is REFOBJID, then you have to enter REFOBJID=:TRDR

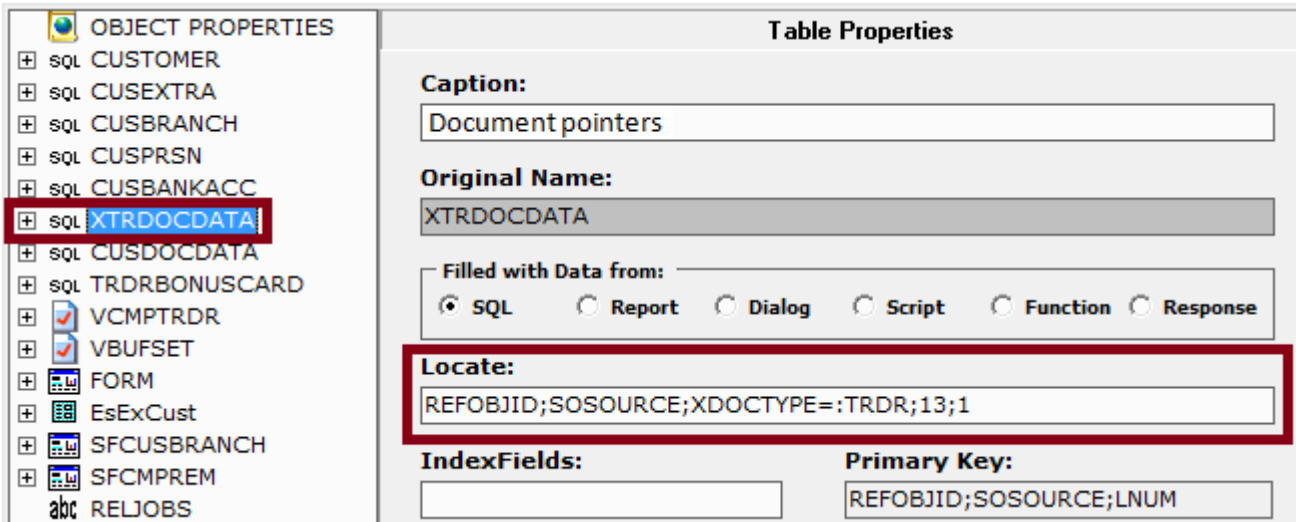


Figure F3.2

Index Fields

It is used in second child tables (child tables linked to child tables) to indicate the fields that define the child-to-child relationship. The example of Figure 3.3 shows the table SNLINES which is child table to MTRLINES, which is also a child table of the parent table FINDOC. Index field of the SNLINES table is the field MTRLINES.

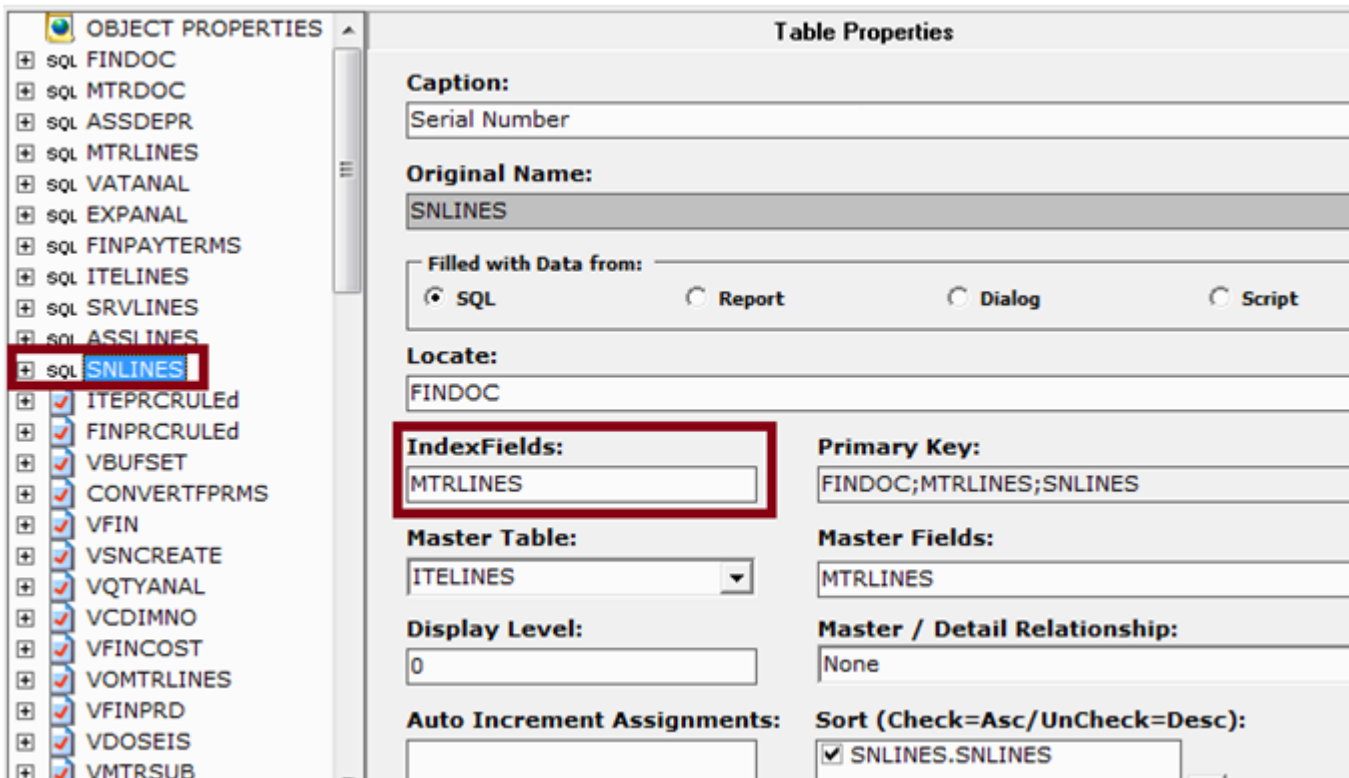


Figure F3.3

Primary Key

Read-only property that displays the primary keys of the selected table

Master Table

Use it in second child tables to indicate the name of the master child table that will be used for locating the records of the second child table.

Master Fields

Use it in second child tables to indicate the fields of the master child table that define the child-to-child relationship.

Display Level

Internal use only

Master / Detail Relationship

Indicates the master – detail relationship. For example, in Sales documents the table MTRDOC has “one to one” relationship with table FINDOC, while the table “MTRLINES” has a “one to many” relationship.

Auto increment Assignments

Internal use

Sort (Check=Asc/Uncheck=Desc)

Indicates the fields that will be used by default for sorting the records of the selected table/view.

F.4 Field Properties

Table fields inside objects have the following properties (Figure F4.1):

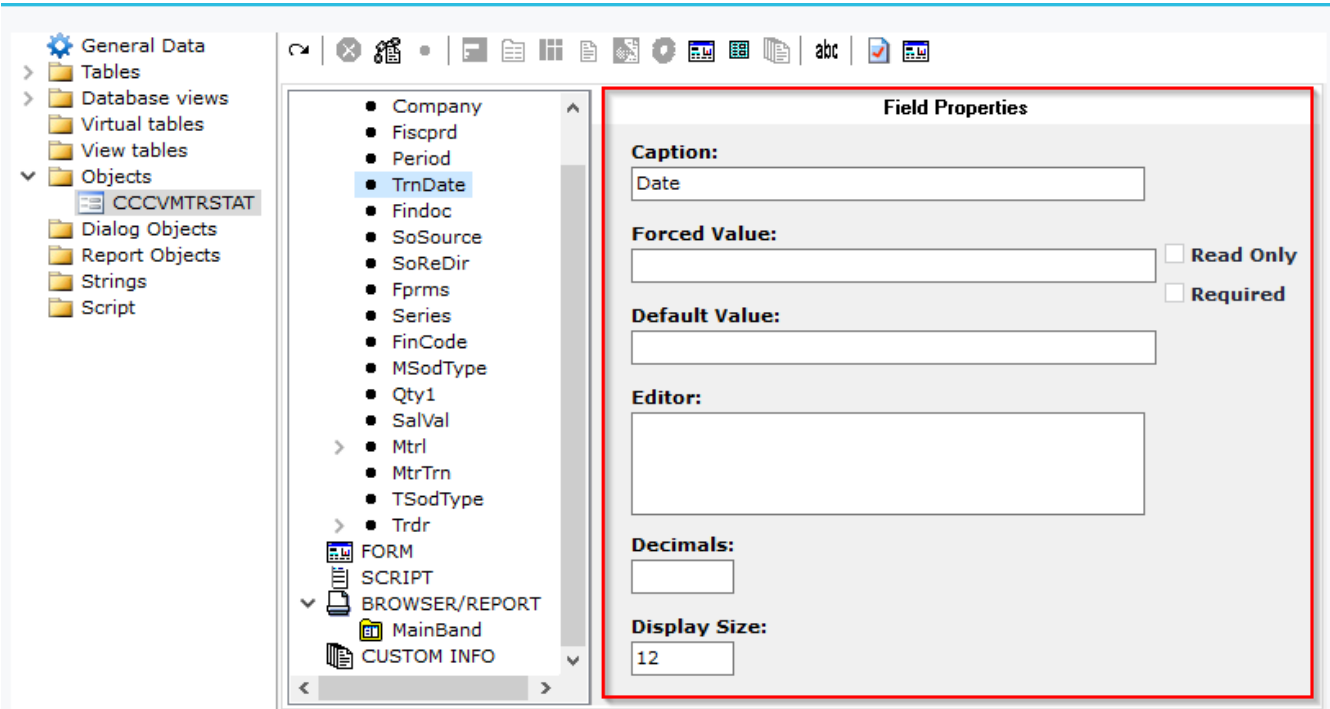


Figure F4.1

Caption

Sets the title of the field.

Forced Value

Indicates the forced value applied to the field. Table Field values are defined through the syntax **:TABLENAME.FIELD**. [System parameter values](#) are defined through the syntax **:X.SYS.PARAMNAME**

Default Value

Indicates the default value applied to the field. Syntax used is the same as Forced Value.

Editor

Changes the form control that displays the data of the field using [Editor Commands](#).

Decimals

Indicates the number of the decimal places. Specific commands (P,V,Q,%,C) can be used in order to define the use of the company defaults for decimal places. (see [Default decimal places](#))

Display size

Sets the default width size display of the field inside SoftOne, for use in datagrids or edit list objects.

Hidden field

If enabled then the field will not appear in the field list of the object in Browsers and Forms.

Display Level

Internal use only

Read Only

Specifies whether the contents of the field can change at runtime by users.

Required

Specifies whether a value for the field is required for the record to be saved (Mandatory field).

F.5 Calculated Fields

Apart from database fields, there is an option of creating calculated (virtual) fields that can be populated through form script code (JavaScript or VBscript) or used as command buttons. Calculated fields are created through right click on tables of the object and the selection of the option “Add calculated Field” (Figure F5.1).

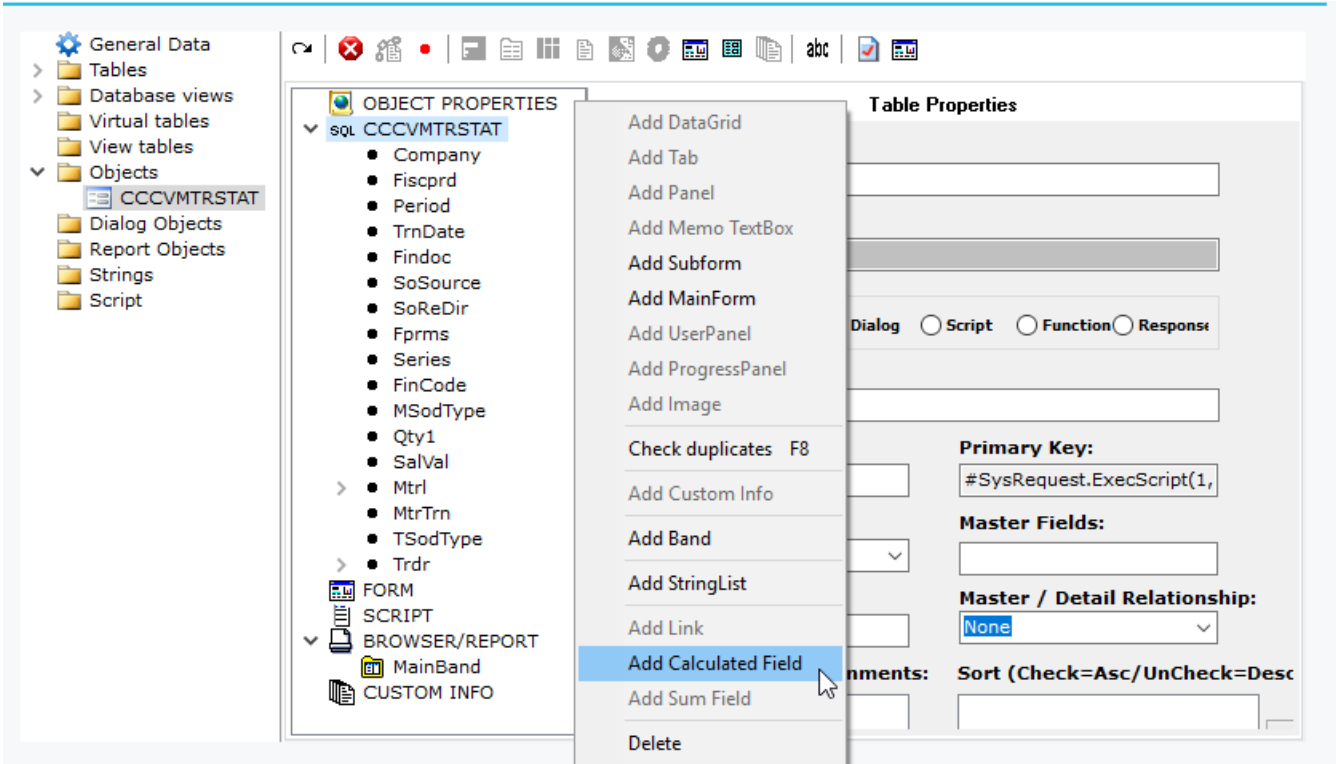


Figure F5.1

Calculated fields created in Report Objects can be used for assigning values to other fields, through the Formula property. Calculated field properties are the following:

Field Type: Indicates the type of the field (Varchar, SmallInt, Double, Date, Integer)

Size: Sets the size of the field

Caption: Sets the title of the field

Formula: Refers to Report Objects only. Enable the checkbox Formula. Then enter the preferred calculation in the Formula property. If the first character of the command is ^ then the calculation will take place after totals are calculated.

F.6 Linked Tables

Tables are linked to fields through the editor property. This gives you the ability to use any field of the linked tables inside custom designed forms or browsers, but these fields are not available when designing the **default** browsers or forms inside a custom object. For this reason, you need to use linked tables that make linked table fields available for designing default browsers and forms inside custom objects.

Design

Fields are linked to tables through the right click option “**Add Link**” (Figure F6.1). Click on “Lock Tree” button, drag a table from the left panel of SoftOne designer and drop it into the property “Link To Table”. Enable any of the available fields of the table as in Figure F6.2.

Another advantage of linked tables is that if you rename the selected fields by adding the prefix **X_** then you can refer to them as if they were database fields, so their names must be unique inside a table. This means that you can call a linked field through form script code, by only entering the name of the table and then the name of the linked field **TableName.X_Fieldname**.

Example

The example of Figure F6.2 links the field MTRL to the table LINEITEM. Reference to the field CODE of the linked table LINEITEM is achieved through the use of CCCDVXREOP.X_MCODE.

Note also, that the linked tables LINEITEM and TRDR use fields with same names (CODE, NAME). Uniqueness is done by using different field names in linked tables: X_MCODE, X_MNAME for table LINEITEM and X_TCODE, X_TNAME for table TRDR.

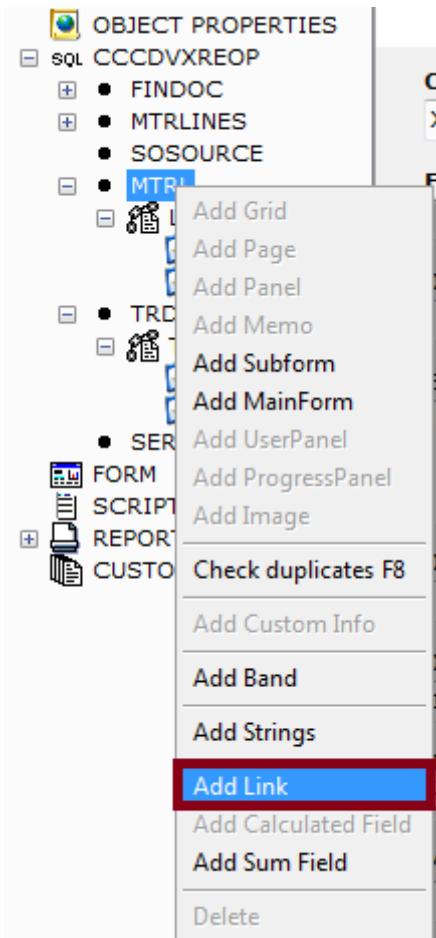


Figure F6.1

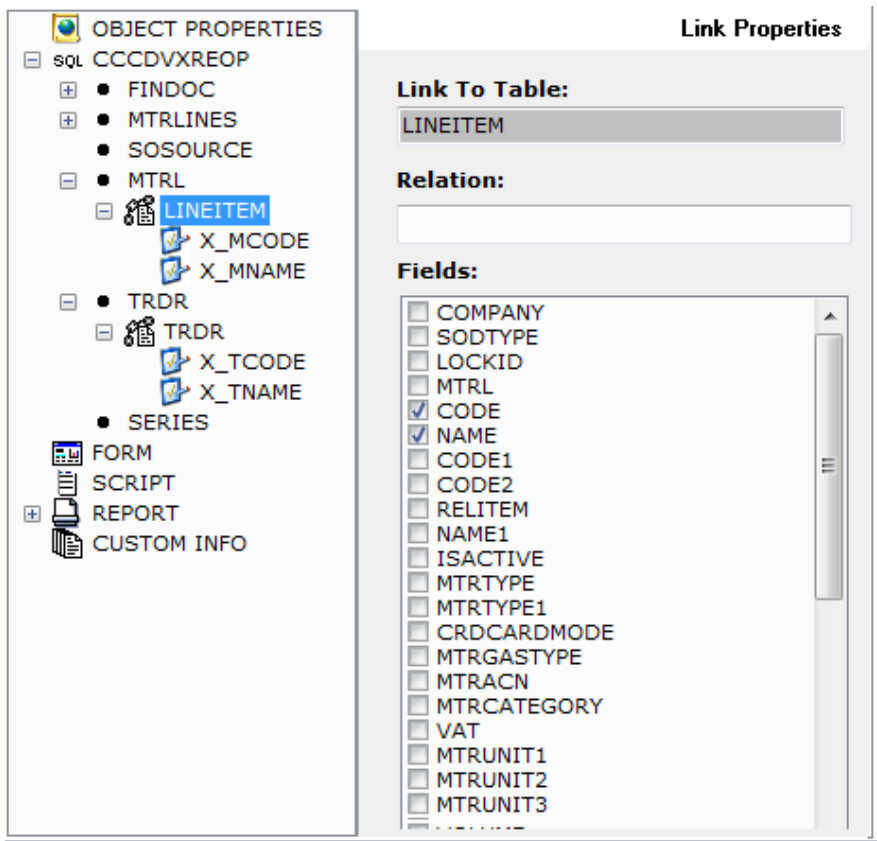


Figure F6.2

F.7 Browsers Design

Objects give you the option of designing default browsers, which cannot be deleted or updated at runtime. Use the button “Lock Tree” when needed and follow the steps below to create a default browser:

- Right click the “**REPORT**” node in the left pane of your custom object and select “**Add Band**”
 - Name the band, for example MyMasterBand (Figure F7.1)
 - Select a table of your object and drag & drop it into the property “**Primary Band Table**”.
- Note that default browsers must **always** have a Primary Band Table, for the object to work properly.
- “**Secondary Band Tables**” is used for browsers displaying master detail tables.
 - Drag and drop the fields (columns) that the browser will display at runtime in the tab “**Band Fields**”
 - Drag and drop the fields that will be used for sorting in the tab “**Sort Order**”
 - Drag and drop the fields that will be used for grouping in the tab “**Grouping**”
 - Finally, drag and drop the dialog filter fields in the tab “**Questions**”.

For each field in questions tab you may also use the following properties:

- **Field Name:** Name of the field. Fields can be added through drag and drop.
- **From Caption:** Title that will be displayed in the dialog textbox “From”
- **Default From Value:** Default value that will be displayed in the dialog textbox “From”
- **To Caption:** Title that will be displayed in the dialog textbox “To”
- **Default To Value:** Default value that will be displayed in the dialog textbox “To”
- **Forced Value:** Defines the forced value of the field

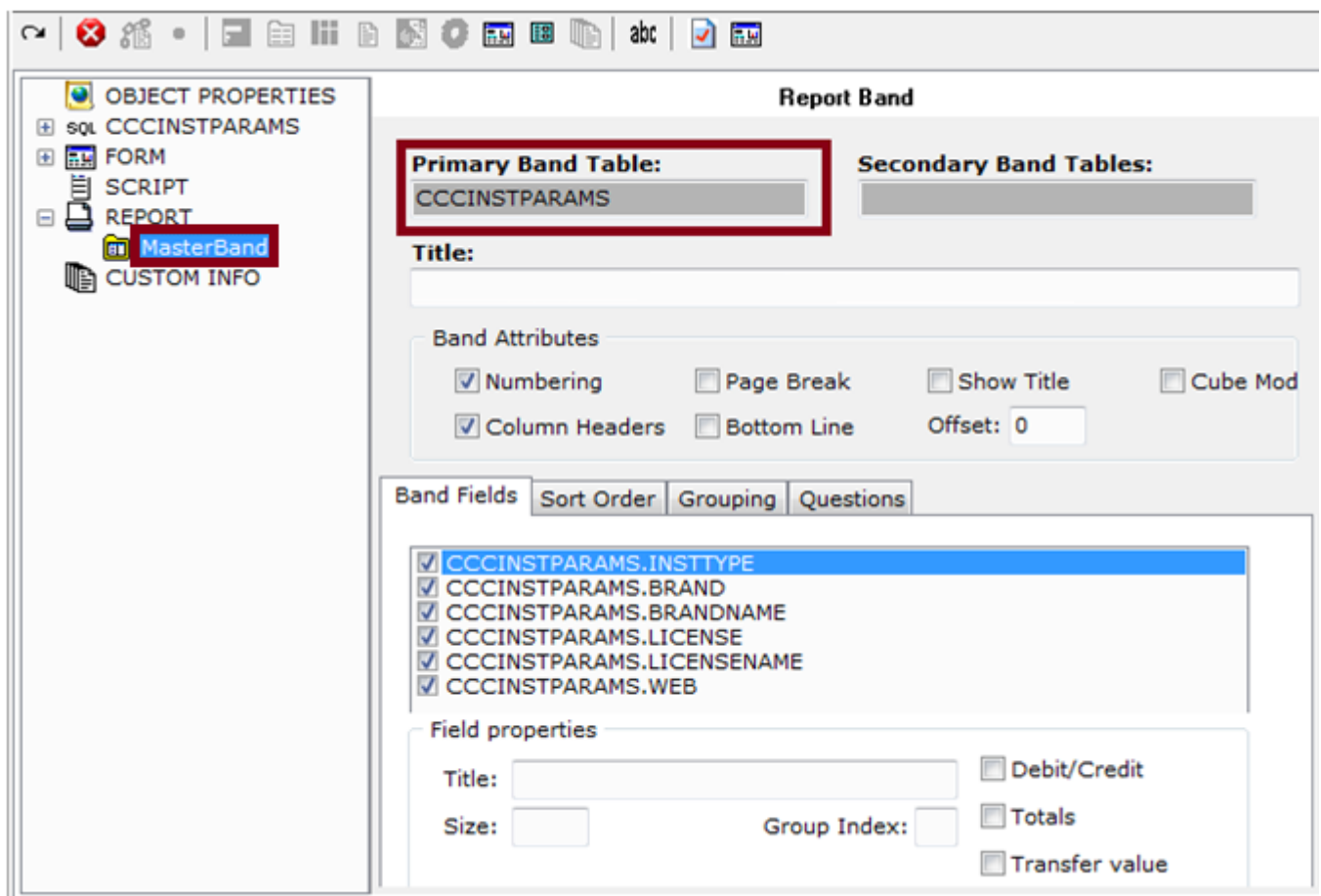


Figure F7.1

F.8 Forms Design

Default forms can be designed using the toolbar buttons or the right click menu options. Commands are the same as in internal objects form design (Figure F8.1).

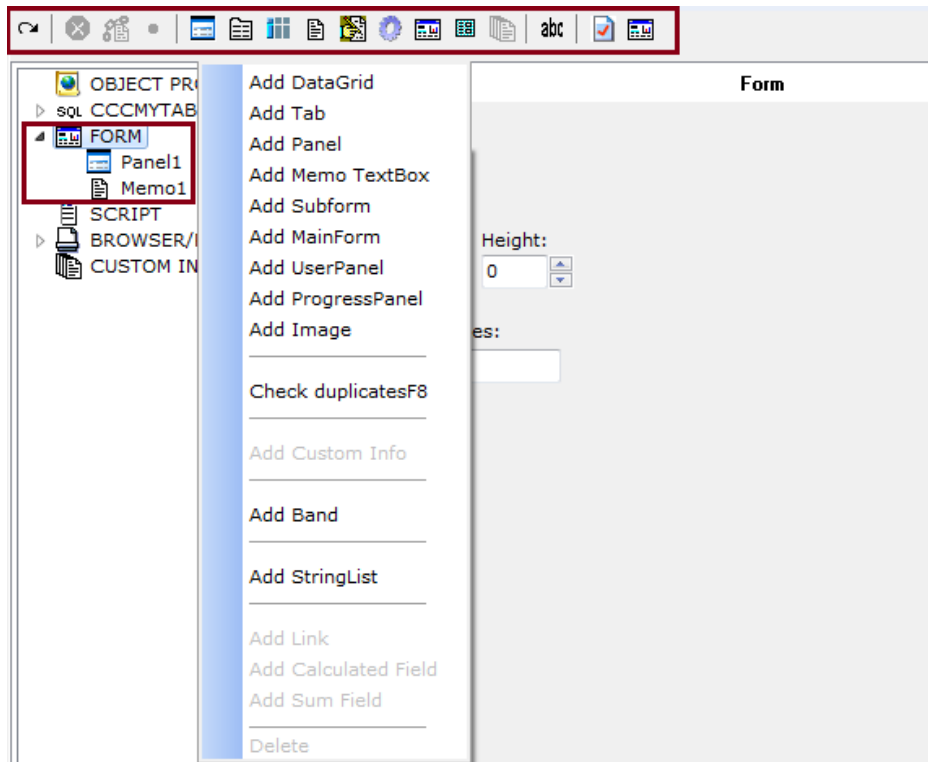


Figure F8.1

Add Datagrid: Adds a new datagrid. Fields are added through drag and drop.

There is also the option of adding a Totals line at the end of the datagrid. This is only available in datagrids created in custom objects. To enable this option, you only need to enter the field names that will be used in the Totals line, in the textbox **"Fields with Totals"** (Figure F8.2).

Figure F8.2

Add Tab: Adds a new tab. Properties are the same as in section [Screen Forms - Tabs](#).

Add Panel: Adds a new panel. Properties are the same as in section [Screen Forms – Panels](#).

Add Memo TextBox: Adds a new memo text box. Properties are the same as in section [Screen Forms – Memo TextBox](#).

Add SubForm: Adds a new subform. Properties are the same as in section [Screen Forms – SubForms](#).

Add Image: Adds a new image. Properties are the same as in section [Screen Forms – Image](#).

Check duplicates: Checks for controls that use the same name.

Add StringList: Adds a new string list. Properties are the same as in section [Screen Forms –String Lists](#).

Delete: Deletes the selected control.

There is also a toolbar button “**Test Form**” that lets you preview the form you have designed. After the preview, you can return to object design through the close button in the upper right corner (Figure F8.2).

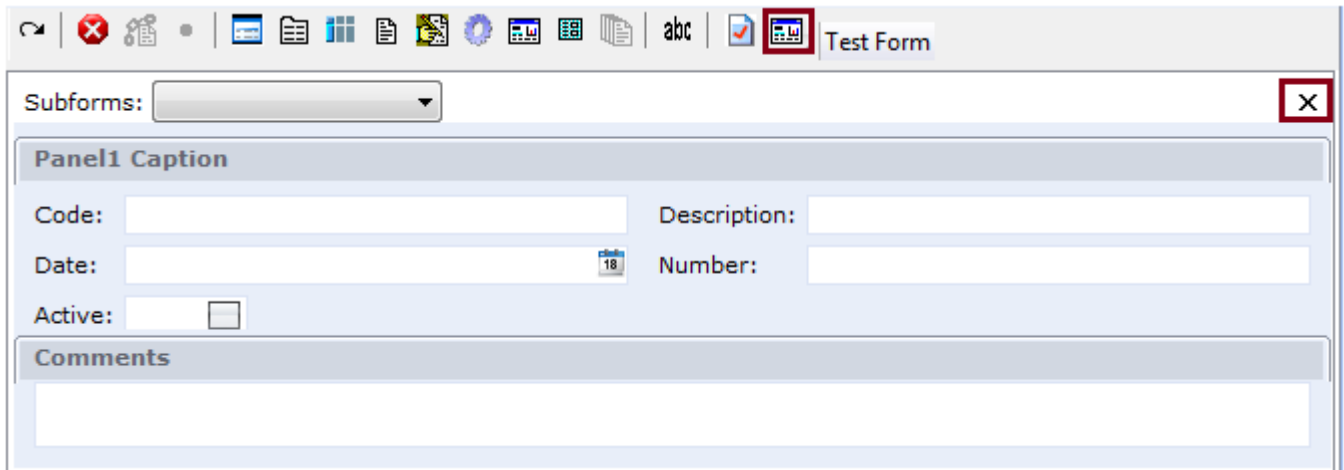


Figure F8.3

Fields can be added in a form control as long as you have enabled the toolbar button “Lock Tree”.

Inside the default form you can add a form script (JavaScript or VBscript) in the same way you add it through “Configuration” button in designed forms. Form script is added using the node “SCRIPT” (Figure F8.4).

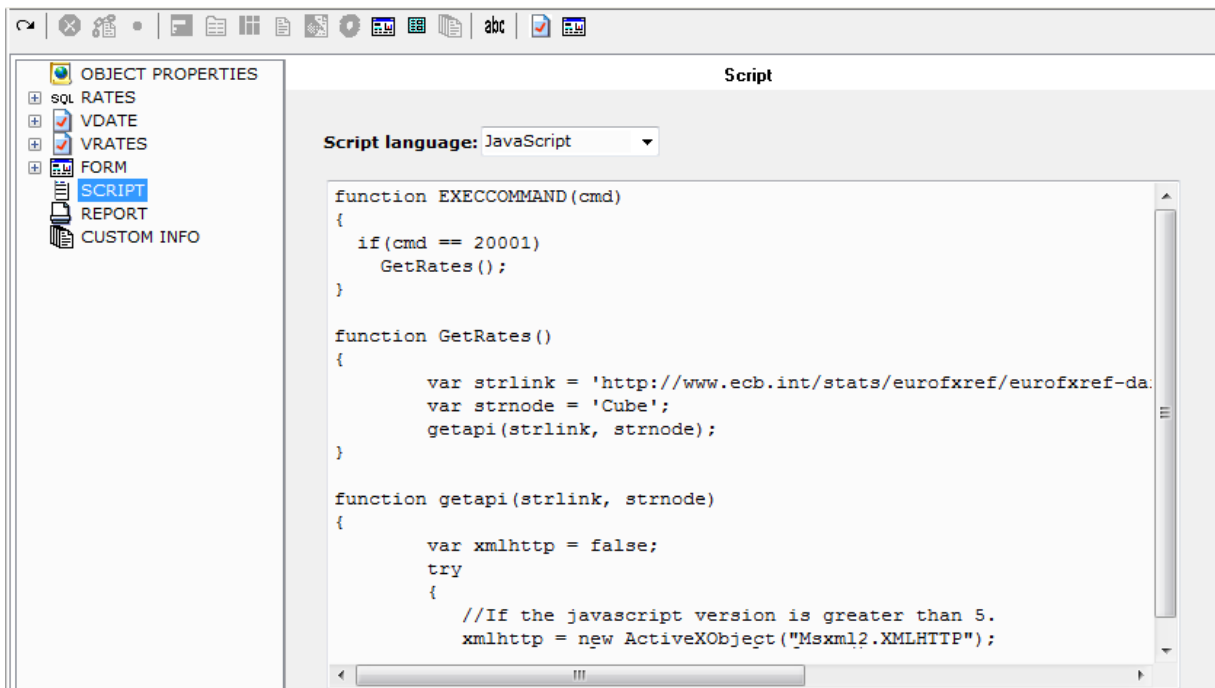


Figure F8.4

F.9 Display Object in Menu

Objects can be displayed in menus as jobs through the following steps:

- Save changes in SoftOne Designer.
- Synchronize the database if needed. Objects design do not need synchronization.
- Create a new job in the menu from the right click option "New job".
- Click on the button next to the textbox "Action/File" (Figure F9.1).
- Select your object from the list.

Alternatively, you can manually select "Object process" as Job type, enter the name of the object in "Action/File" and then the name of the job.

The object, at runtime, will be displayed as in Figure F9.2. Notice that all the tools are available.

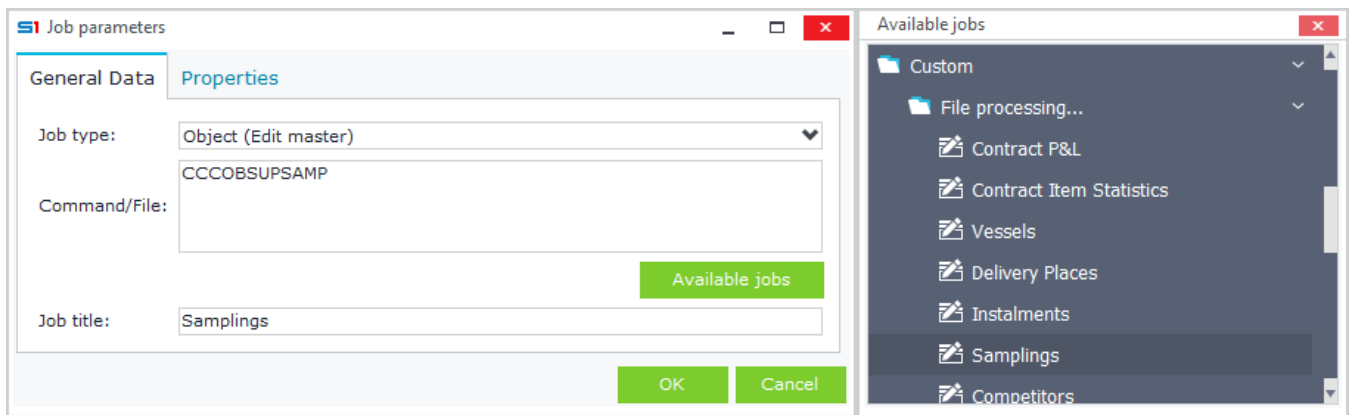


Figure F9.1

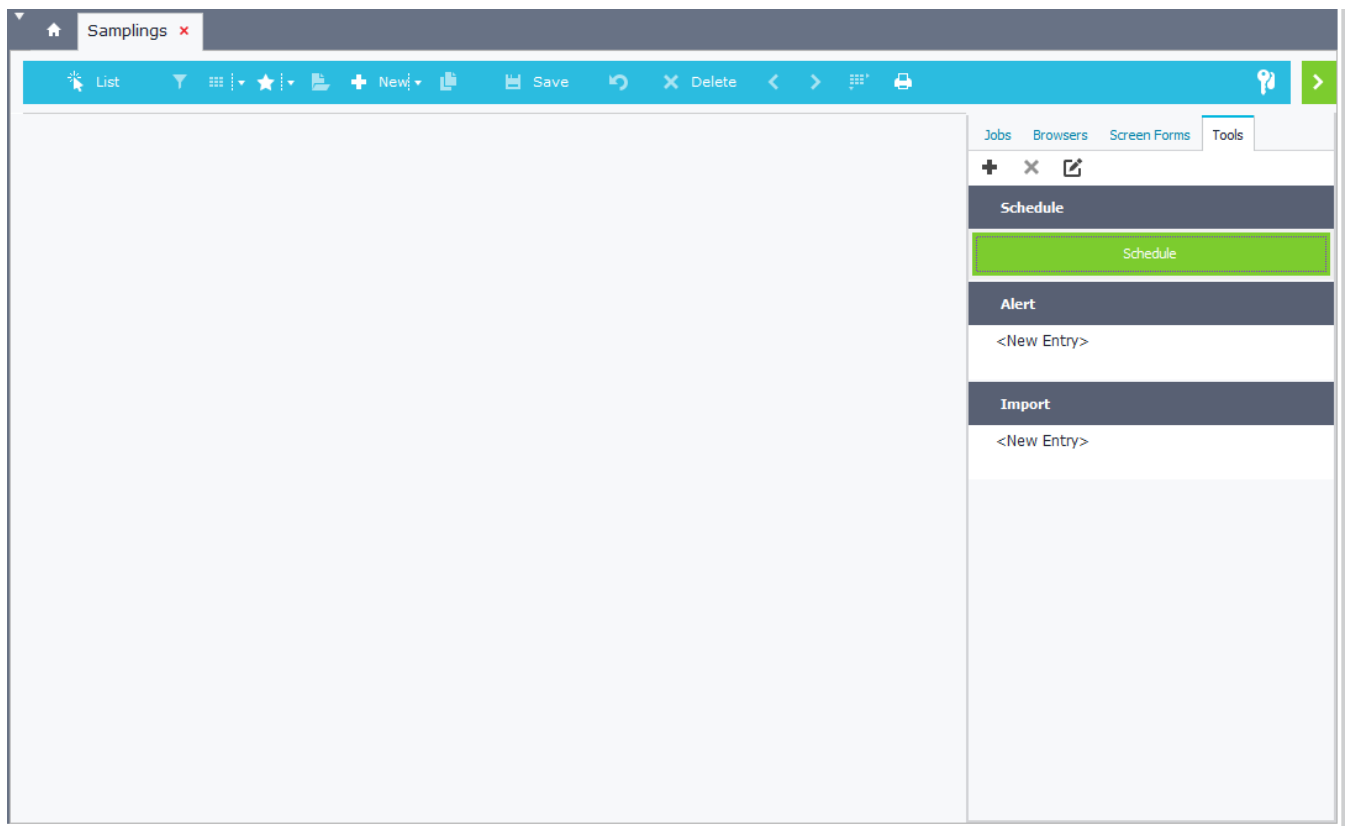


Figure F9.2

F.10 Printout Forms

Printout forms that use tables and data from custom objects you have created through SoftOne designer can be created using the following steps:

- Create a new job in the menu and select “Object process” as Job type.
- In “Action/File” enter the command “**TEMPLATES.ObjectID**”, where ObjectID is the unique ID of your object (Figure F10.1).
- Enter a job title and execute the job.

The printout form will be displayed as in Figure F10.2, allowing you to select from any of the available table fields that have been defined in your object.

The image shows a 'Job parameters' dialog box with two tabs: 'General Data' and 'Properties'. The 'Properties' tab is selected. It contains the following fields:

- Job type:** A dropdown menu showing 'Object (Edit master)'.
- Command/File:** A text box containing 'TEMPLATES.20010'.
- Job title:** A text box containing 'Samplings Printout forms'.

There is a green button labeled 'Available jobs' to the right of the 'Command/File' field. At the bottom right, there are 'OK' and 'Cancel' buttons.

Figure F10.1

The image shows the 'Design printout forms' interface. A 'Select Fields' dialog box is open, displaying a list of fields for the 'ITEM SAMPLINGS, (CCCITEMSAMPLE)' object. The fields include:

- Code, (Indicator), (CCCITEMSAMPLE)
- Item, (Numeric), (MTRL)
- Supplier, (Numeric), (TRDR)
- Date, (Date), (DATE)
- Qty, (Numeric), (QTY)
- Final price, (Numeric), (PRICE)
- Whouse, (Numeric), (WHOUSE)
- Comments, (Alphanumeric), (REMARKS)
- Status, (Numeric), (CCCSTATUS)
- LINENUM, (Numeric), (LINENUM)
- Currency, (Numeric), (SOCURRENCY)
- Upd_User, (Numeric), (UPDDUSER)
- Upd_Date, (Date), (UPDDATE)
- Supplier Item code, (Alphanumeric), (M)
- Supplier Item description, (Alphanumeric)
- Sampling User, (Numeric), (SEARCHUS)
- Valid from, (Date), (VALIDATEFROM)

The main window shows a grid for designing the printout form. The grid has a header row with columns labeled '0', '40', '50', and '60'. The grid is divided into sections by red lines. The 'Design printout forms' tab is active, and the 'Active' checkbox is checked.

Figure F10.2

F.11 Object Examples

F.11.1 Object using custom table

The following example creates an object (CCCCUSTSHIP) that uses the custom table “Vessels” (CCCCUSTSHIP). Create the Object through the right click option in Database Designer. Then, drag the custom table CCCCUSTSHIP from the left panel and drop it under the Object Properties. Enter CCCCUSTSHIP in the Object Name. Enter the caption of the object, Vessels (Figure F11.1).

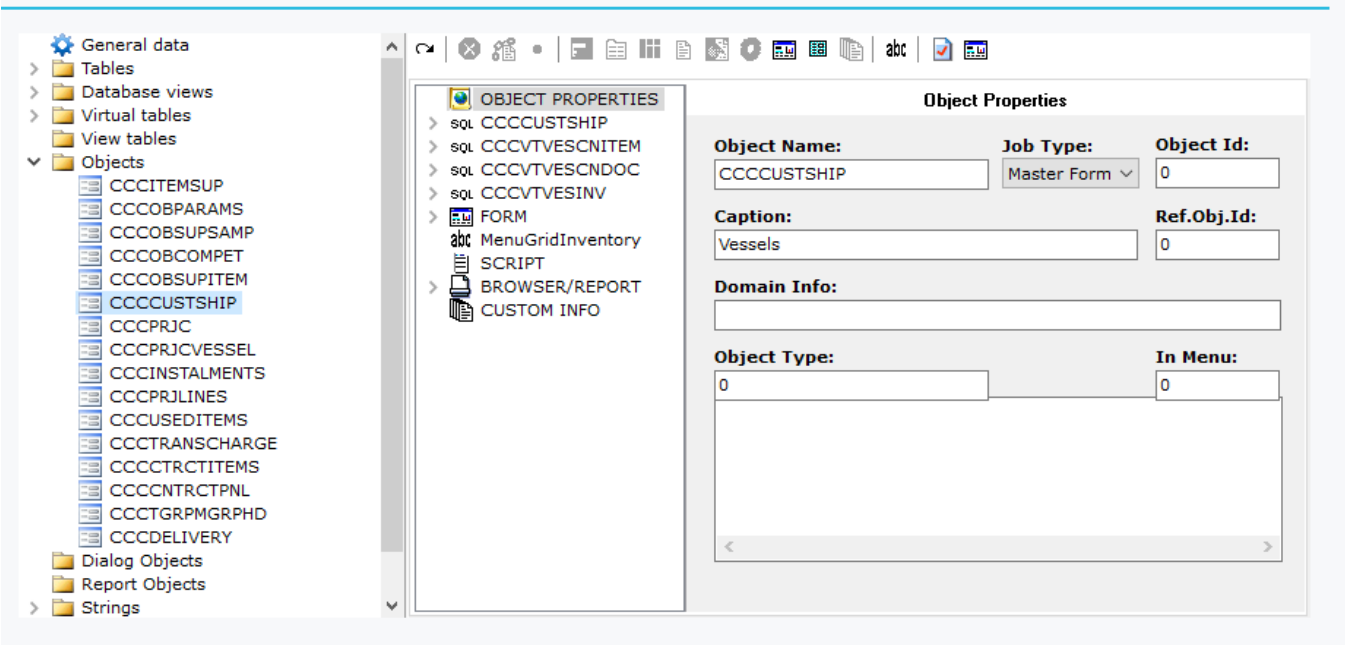


Figure F11.1

Create a new Band under the Report node and drag and drop the table CCCCUSTSHIP in the “Primary Band Table” property. Drag and drop fields from the table in the “Band Fields” tab to be displayed as columns in the default browser (Figure F11.2).

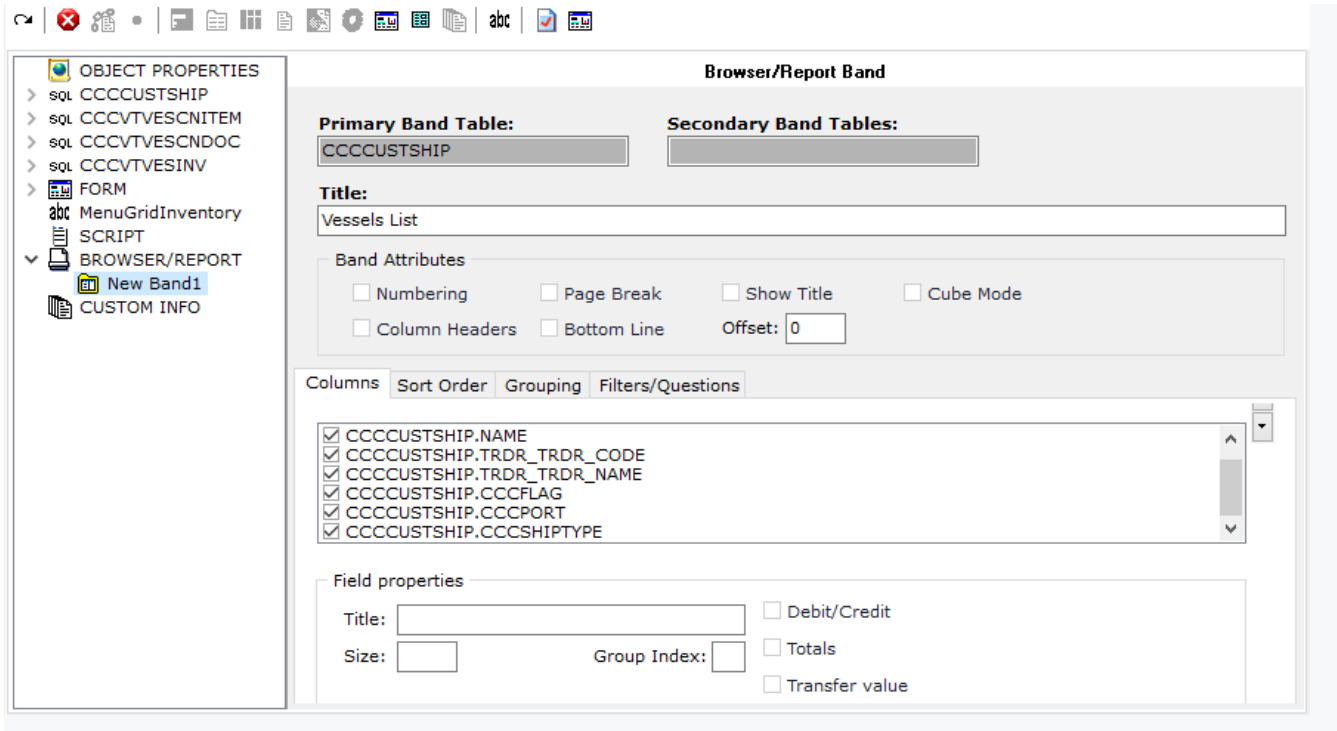


Figure F11.2

Create Panels under the Form node and add fields in the "Available Fields" property.
Add any other tables, virtual tables etc. and add memo textboxes, grids and context menus as in Figure F11.3.
Finally, create a new job to display the object in the menu (Figure F11.4)

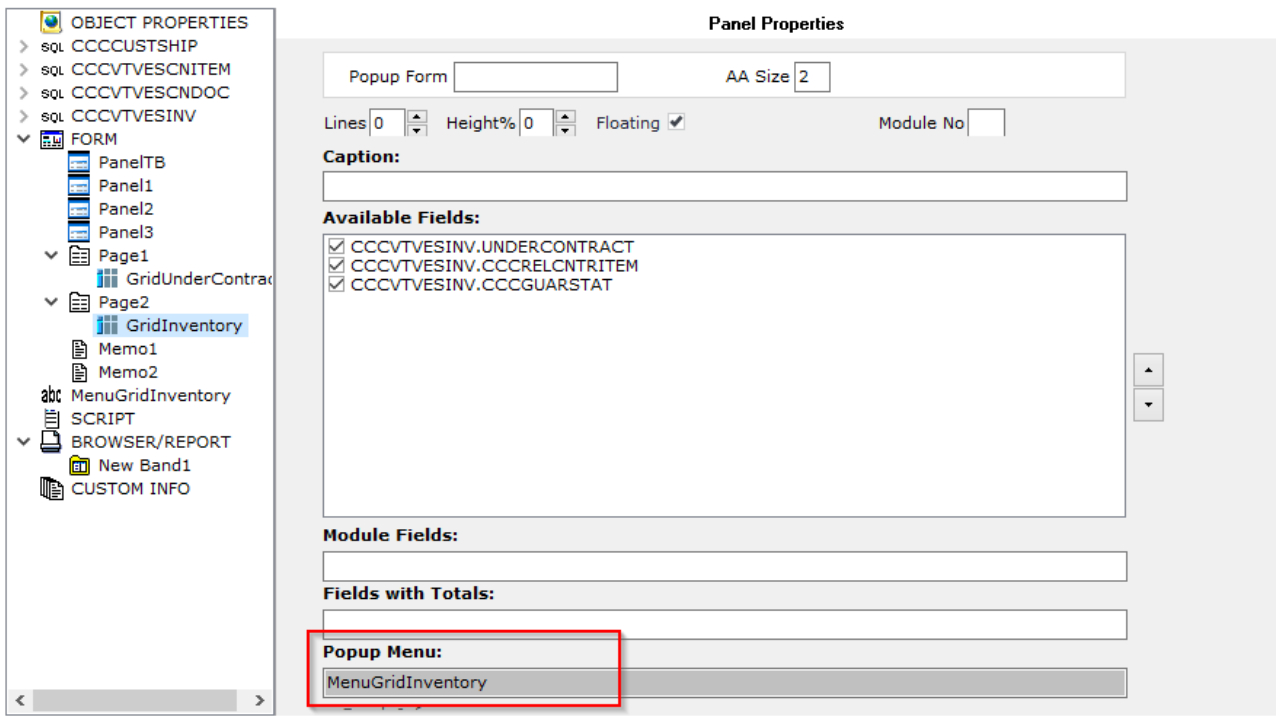


Figure F11.3

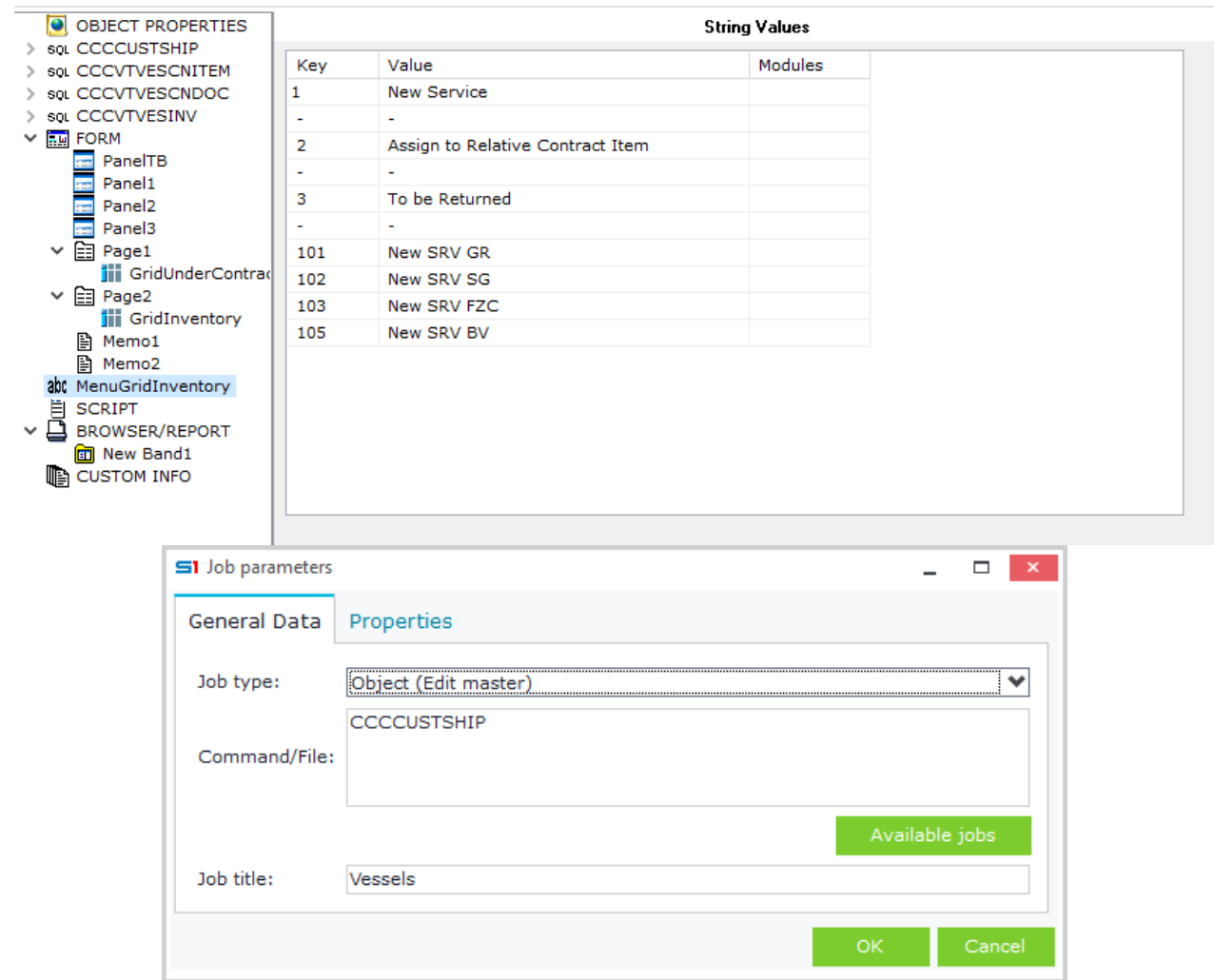


Figure F11.4

F.11.2 Object using database view

The following example creates an object (CCCVMTSTAT) that uses the custom database view "CCCVMTSTAT" created in the section [Browser Redirection](#).

- Create the Object and enter the name CCCVMTRSTAT.
- Drag and drop the database view CCCVMTRSTAT under the Object Properties.
- In Parameters textbox, enter the command BROWSERONLY = 1 (Figure F11.5), that displays the object in browser mode only (Form design is not available).

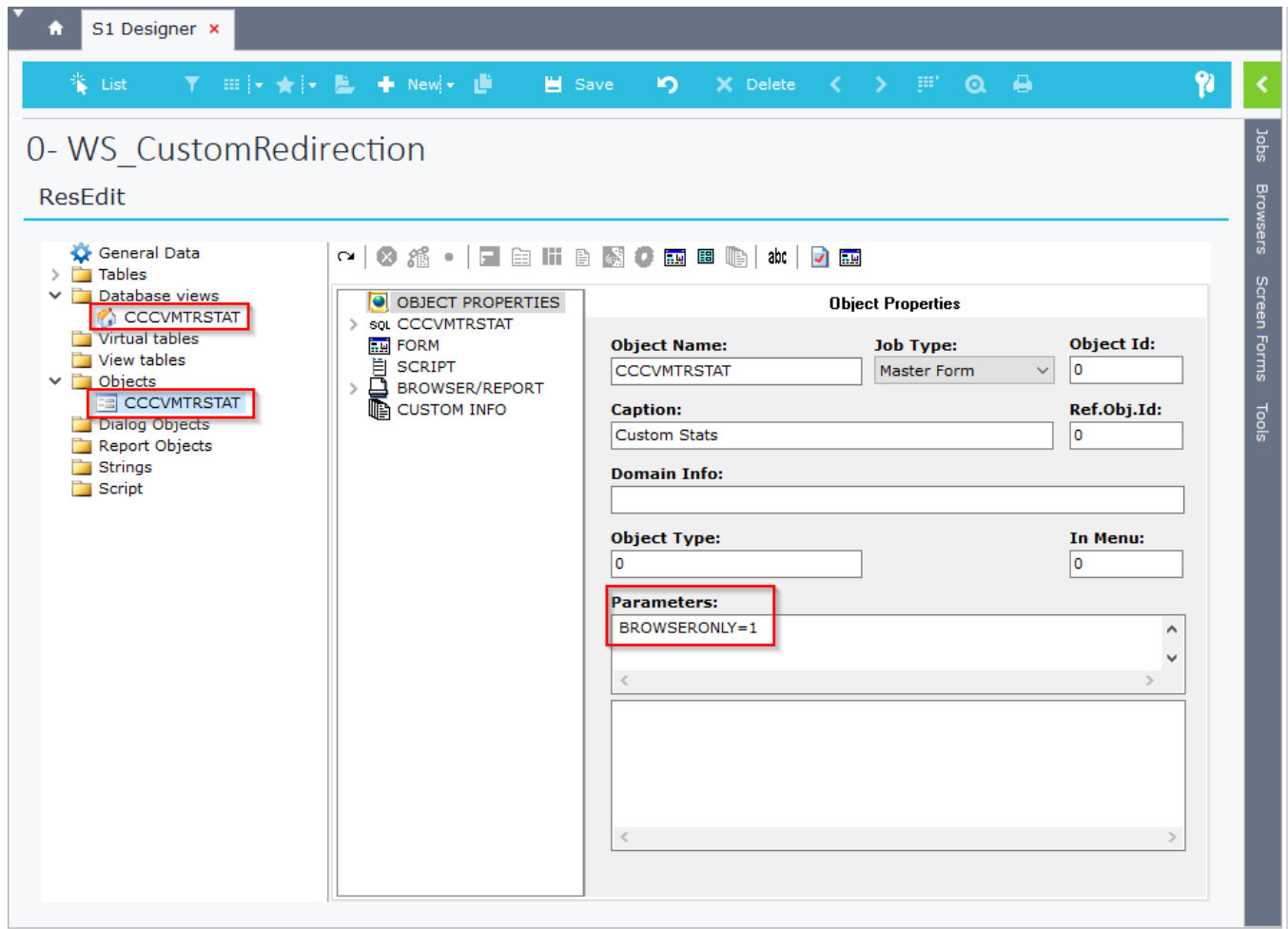


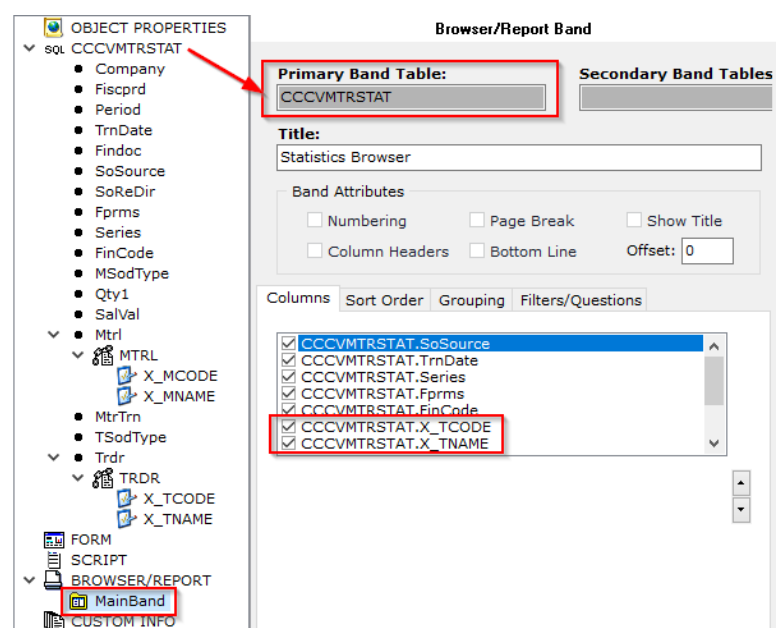
Figure F11.5

Add a new Band in "Browser/Report" and drag n' drop the DB View CCCVMTRSTAT in "Primary Band Table".

If "Primary Band Table" is left blank, then the object will not open and an error will raise.

Drag n' drop any fields inside "Columns", "Sort order", "Grouping" and "Filters" of the band (Figures F11.6 and F11.7).

If you need to add fields that are linked to a table (e.g. X_TCODE, X_TNAME) then you have to right click the field, add the linked table and then select the fields (Figure F11.6). It's better to rename the linked fields and add the prefix X_ because this makes them available for reference from Form script code as if they were fields of the table (e.g. CCCVMTRSTAT.X_TCODE).



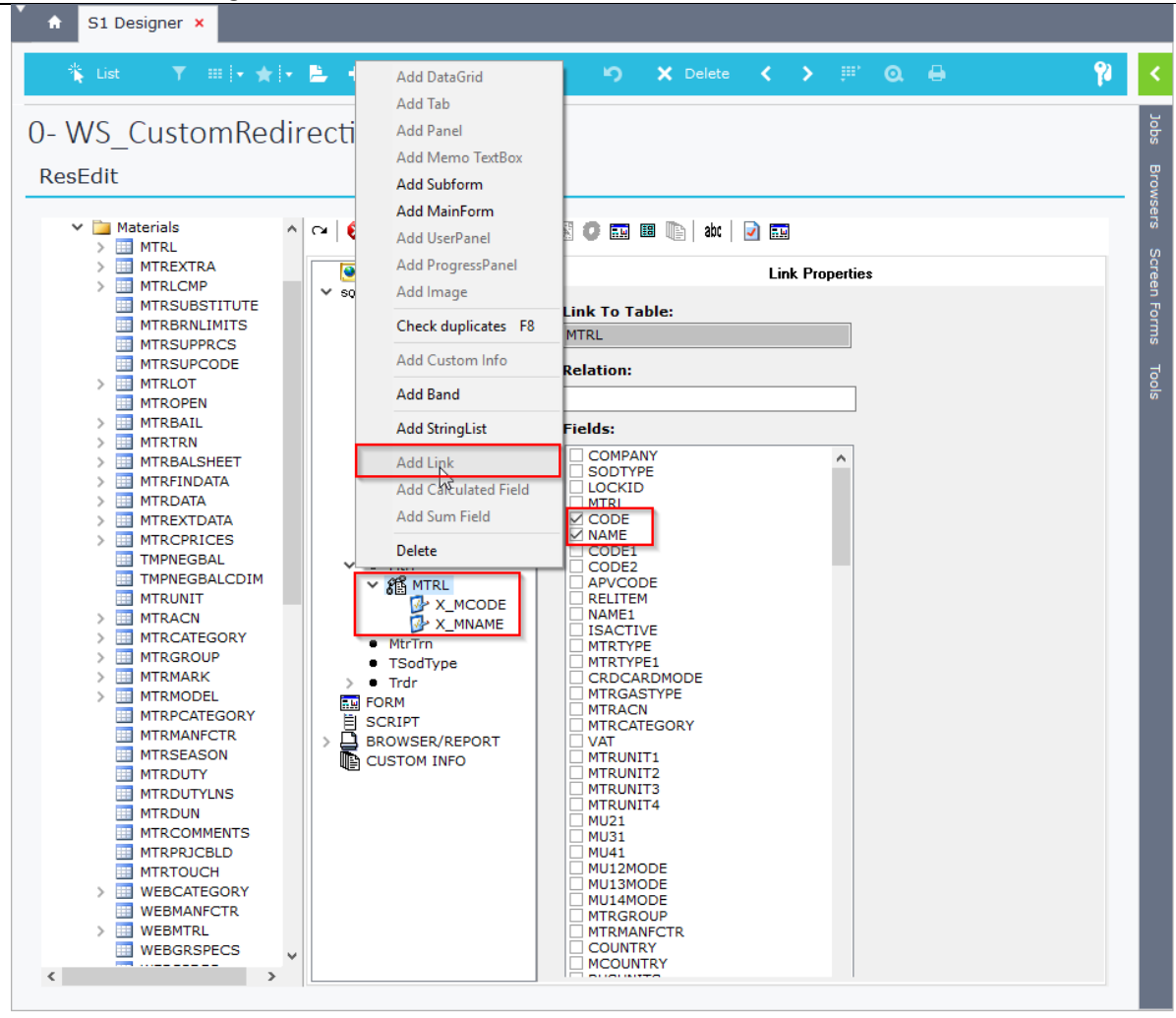


Figure F11.6

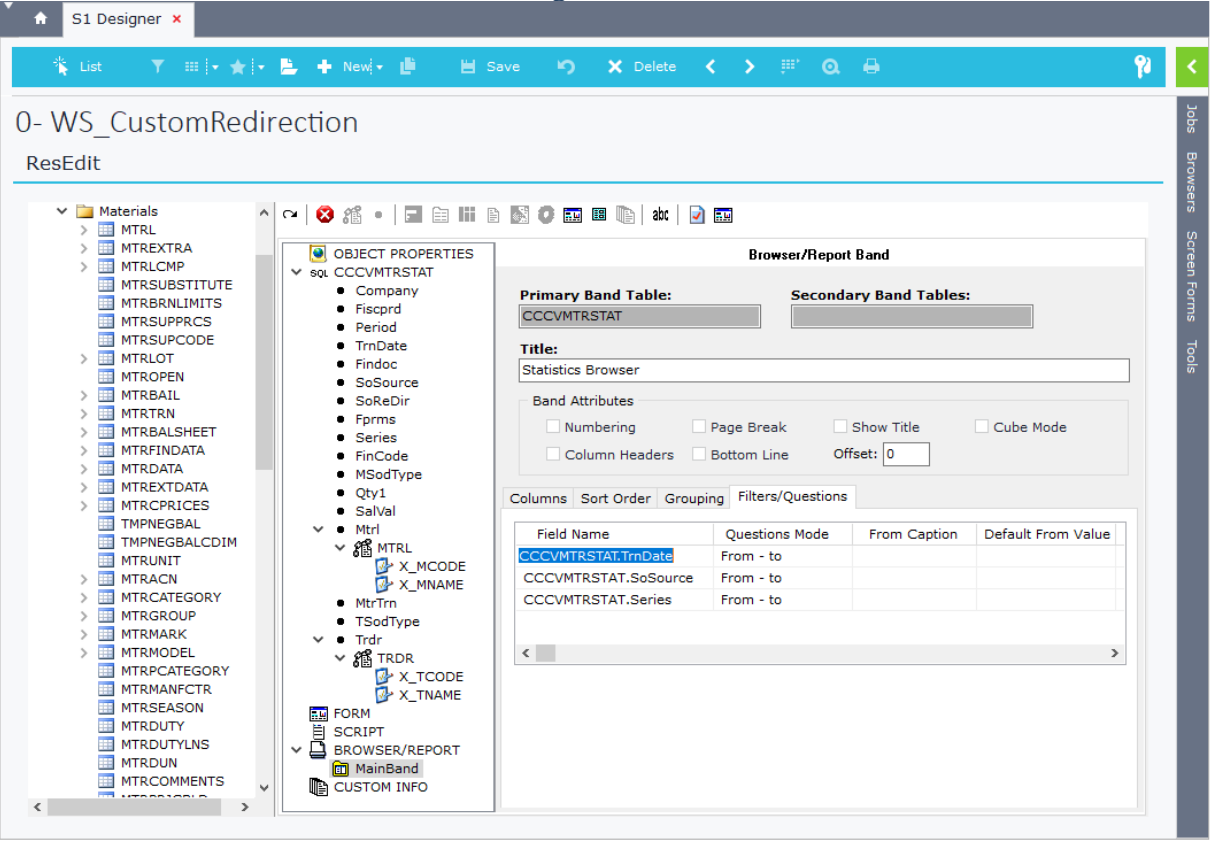


Figure F11.7

G. Virtual Tables

Virtual Tables are cached memory tables (datasets), whose fields are created only in SoftOne designer and not in the database. They are populated through script code and can be displayed in datagrids, in Report Objects and in Dialog Objects. The main difference with database views is that Virtual Tables can be populated at runtime and are also editable, while database views are populated through the data retrieved from the views of the database.

G.1 Design Virtual Table

Virtual tables are created through the right click option “New – Virtual Table” of SoftOne Designer. Fields are created the same way as in tables. That is, you can use all the properties (Editors, Default values, etc.). Virtual Tables are inserted in object forms, as tables (through Configuration window). They are populated through script code and their columns can be displayed in form controls (e.g. datagrids) through drag and drop.

G.2 Virtual Table Example

The following example uses a virtual table in sales documents that displays the lines of Item sets and provides the user the ability to edit, delete or add new records. Changes are saved in Item sets.

- Create a Virtual Table through right click pop up menu in SoftOne Designer.
- Insert the fields ITELINESMTRL, MTRL, QTY, PRICE as in Figure G1.
- Enter ITEM in the editor property of the field MTRL, to link the field with Items.

Field QTY will be used to display the item sets quantities.

The screenshot displays the SoftOne Designer interface. On the left, a sidebar contains a tree view with folders for General Data, Tables, Database views, Virtual tables (containing CCCSPCLINES), View tables, Objects, Dialog Objects, Report Objects, Strings, and Script. The main workspace shows the configuration for the virtual table 'CCCSPCLINES'. The 'Table basic data' tab is active, displaying the following fields:

Field name	Data type	Size	Title	Decimals
ITELINESMTRL	Integer	4	Doc Line Item	
MTRL	Integer	4	SPCS Item	
QTY	Float	8	Qty	
PRICE	Float	8	Price	

Below the table, the 'General' tab is selected, showing properties for the 'MTRL' field:

- Size: 8
- Title: Qty
- Required: ☐
- Default value:
- Forced value:
- Always included in SELECT statem: ☐

At the bottom, the 'ResEdit' window is open, showing the 'Editor' property for the 'MTRL' field set to 'ITEM'.

Figure G1

- After synchronizing the database, open the sales documents object in design mode.
- Use the configuration window to add the virtual table, through the “User defined tables” tab (Figure G2).

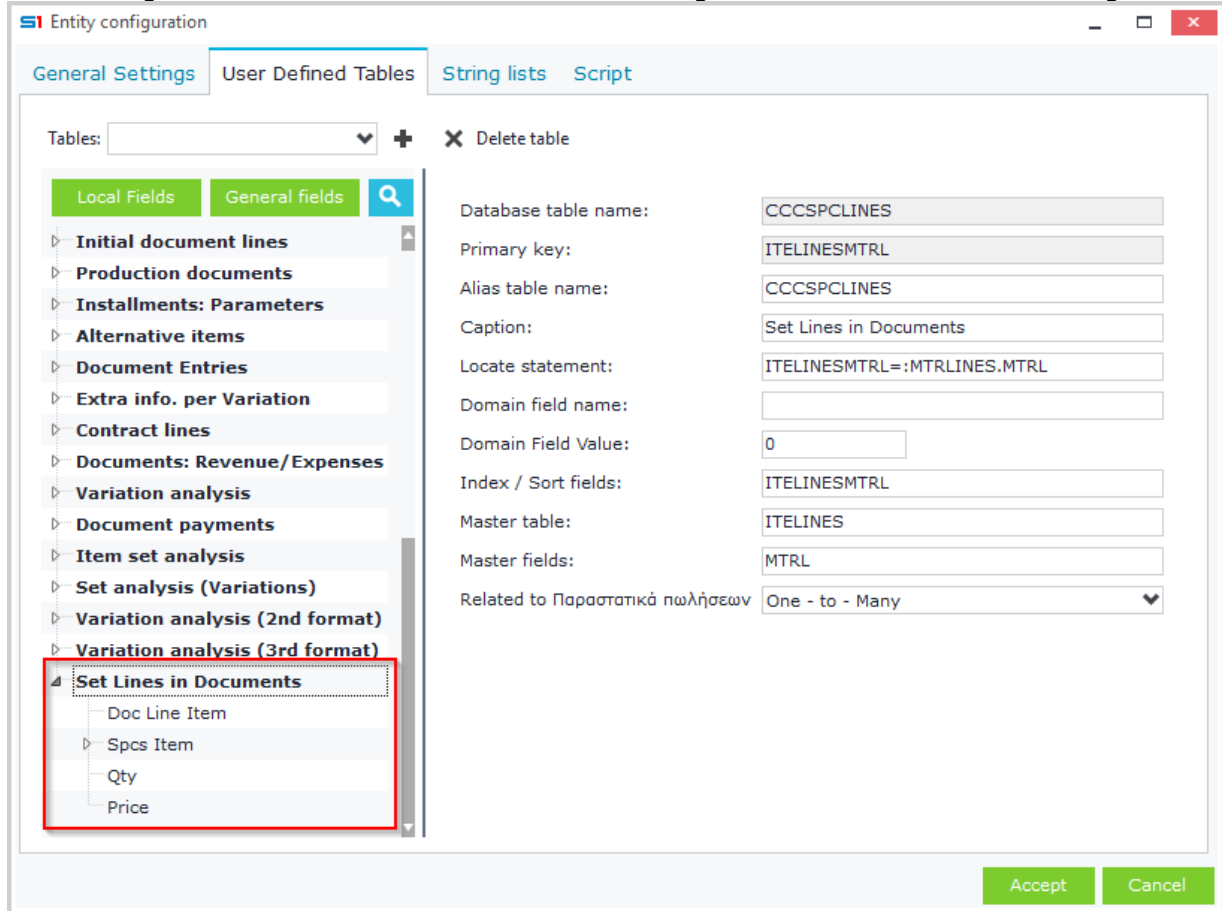


Figure G2

In the tab “Script” of the configuration window enter the code below, which populates the virtual table with the lines of the item set. It also uses the command X.SETPROPERTY('MERGECHANGELOG', 'True'), to save the changes and deactivate the “Save” button.

```
function ON_LOCATE() {
    ITELINES.DISABLECONTROLS;
    try {
        ITELINES.FIRST;
        while (!ITELINES.EOF()) {
            str = "SELECT MTRL,QTY1,PRICE FROM SPCLINES WHERE SPCS=(SELECT SPCS FROM MTRL
WHERE MTRL=:1) ";
            ds = X.GETSQLDATASET(str, ITELINES.MTRL);
            ds.FIRST;
            while (!ds.EOF()) {
                CCCSPCLINES.APPEND;
                CCCSPCLINES.ITELINESMTRL = ITELINES.MTRL;
                CCCSPCLINES.MTRL = ds.MTRL;
                CCCSPCLINES.QTY = ds.QTY1;
                CCCSPCLINES.PRICE = ds.PRICE;
                CCCSPCLINES.POST;
                ds.NEXT;
            }
            ITELINES.NEXT;
        }
    } catch (e) {
        X.WARNING(e.message);
    }
    finally {
        ITELINES.ENABLECONTROLS;
        X.SETPROPERTY('MERGECHANGELOG', 'True');
    }
}
```

In order to update the Item sets object with user changes, enter the code below, that runs in AFTERPOST of Sales Documents and locates the Object of Item Sets, deletes and re-enters all lines of the Set based on the lines of the Virtual Table of the sales documents.

Notice that DISABLECONTROLS is not used in ITELINES, because we need to iterate the virtual table CCCSPCLINES, which is child table of ITELINES (Locate Statement ITELINESMTRL=:MTRLINES.MTRL – Figure G2). If we use ITELINES.DISABLECONTROLS then CCCSPCLINES records will not be located while we loop ITELINES records.

```
function ON_AFTERPOST() {
    ChangeSetLines();
}

function ChangeSetLines() {
    //ITELINES.DISABLECONTROLS; //Disable controls should not be used when iterating child of child table
    ITELINES.FIRST;
    while (!ITELINES.EOF()) {
        strqry = 'SELECT TOP 1 SPCS FROM SPCS WHERE SPCS=(SELECT SPCS FROM MTRL WHERE MTRL=:1)';
        ds = X.GETSQLDATASET(strqry, ITELINES.MTRL);
        ds.FIRST;
        if (!ds.EOF()) {
            try {
                ObjSPCS = X.CreateObj('SPCGPS');
                ObjSPCS.DBLocate(ds.SPCS);
                TblSPCS = ObjSPCS.Findtable('SPCGPS');
                TblSPCLines = ObjSPCS.Findtable('SPCLines');
                if (TblSPCS.SPCS == ds.SPCS) {
                    TblSPCS.Edit;
                    TblSPCLines.FIRST;
                    while (!TblSPCLines.EOF()) {
                        TblSPCLines.DELETE;
                    }
                    CCCSPCLINES.FIRST;
                    while (!CCCSPCLINES.EOF()) {
                        TblSPCLines.APPEND;
                        TblSPCLines.MTRL = CCCSPCLINES.MTRL;
                        TblSPCLines.QTY1 = CCCSPCLINES.QTY;
                        TblSPCLines.PRICE = CCCSPCLINES.PRICE;
                        TblSPCLines.POST;
                        CCCSPCLINES.NEXT;
                    }
                    ObjSPCS.DBPost;
                }
            }
            finally {
            }
        }
        ITELINES.NEXT;
    }
}
```

Finally, create a new datagrid in the form and add the columns of the Virtual Table as in Figure G3.

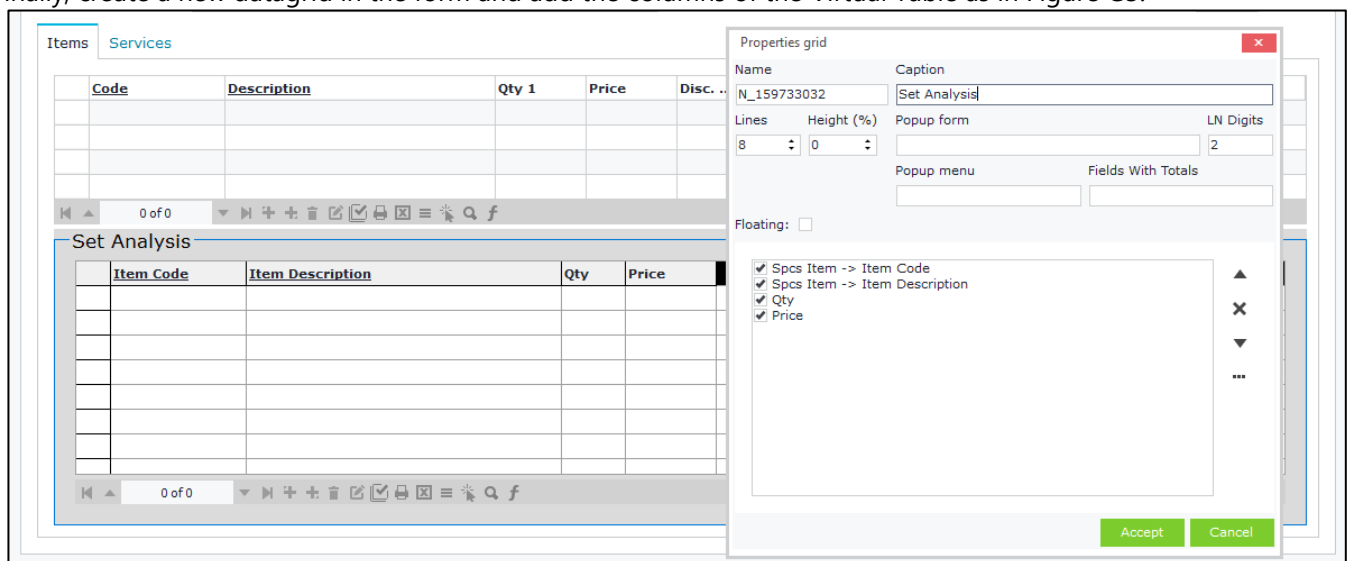


Figure G3

H. Report Objects

Report Objects are used to create dynamic reports. Their difference with the designed reports is that they give you the option of changing at runtime the column widths, captions, etc. depending on the filters.

H.1 Design Report Object

Reports using Report Objects are designed using database tables, virtual tables or view tables. In Report objects, Form is used for dialog filters, while Report displays the columns.

If you use virtual tables then they must be populated through form script. Additionally, you have to add the following commands in the object parameters:

CODE=SoScriptCodeClt

SCRIPTNAME=MyScript, where MyScript stands for the script that will be used for displaying the columns of the report.

Create a new Report Object through the right click option of SoftOne Designer. Select **"Report"** in the "Job Type" property of the object. (Figure H1.1)

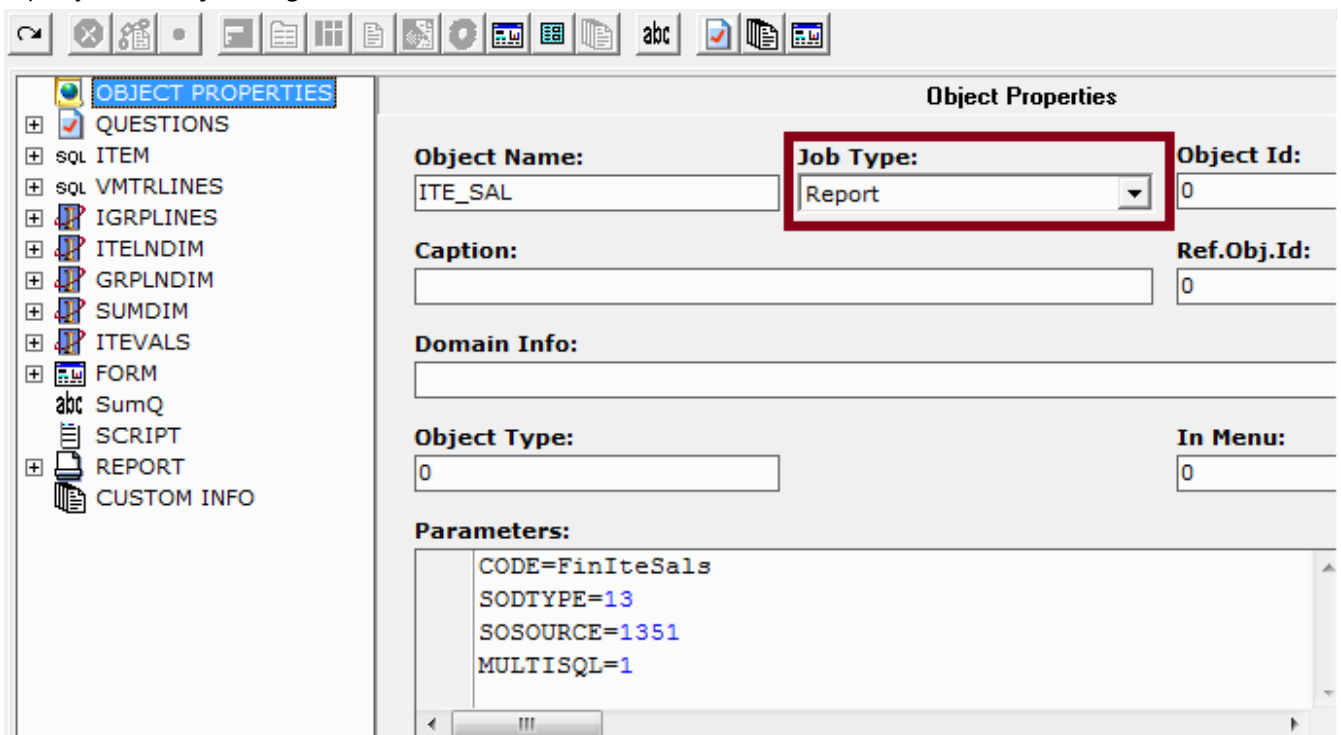


Figure H1

Use a virtual table for the dialog filters and select **"Dialog"** in its "Filled with Data from" property (Figure H2).

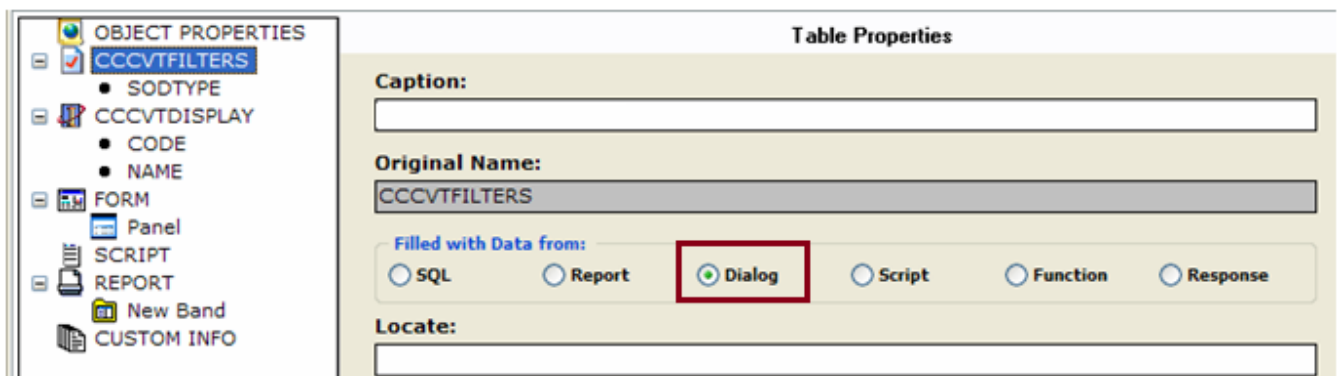


Figure H2

For tables that will be used as report columns you have to use the "SQL", "Report" or "Function" property. You don't have to fill in the Locate property in the tables that are used for report columns (Figure H3).

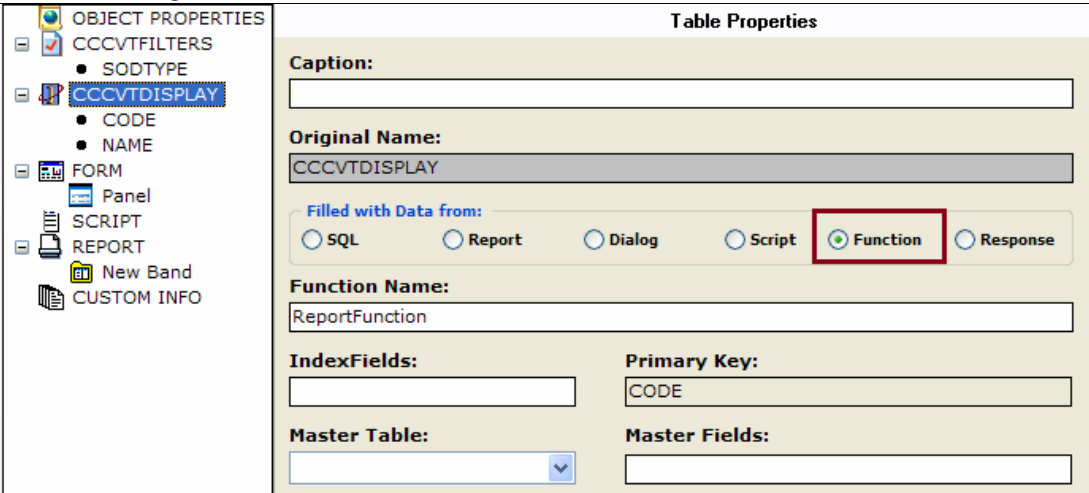


Figure H3

Cumulative fields can also be defined inside the report using the prefix **SUM_** followed by the name of the field. The initial value of the field can also be defined, through the “Forced Value” property. In the example of Figure H4 the field SUM_DEBIT will display the cumulative sum of the field DEBIT and will also have as initial value the value of the field FBALDATA.TOBO.

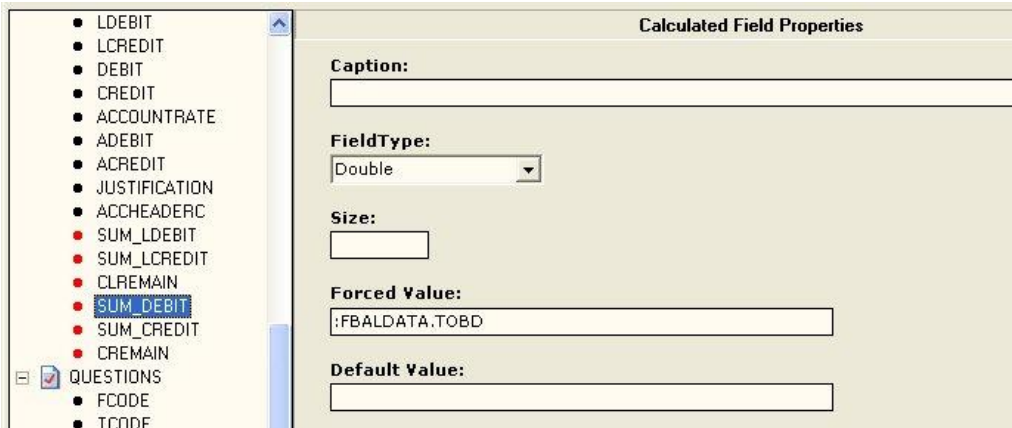


Figure H4

Under the Report node of the object properties add a new band and insert the table, whose fields will be used as report columns, in the “Primary Band Table” textbox (Figure H5).

The Band Attributes are the following:

- Numbering:** Auto increment number for each line of the report
- Page Break:** Change page on record
- Offset:** Number of empty characters
- Column Headers:** Displays the column headers (captions)
- Bottom Line:** Graphical line that separates records

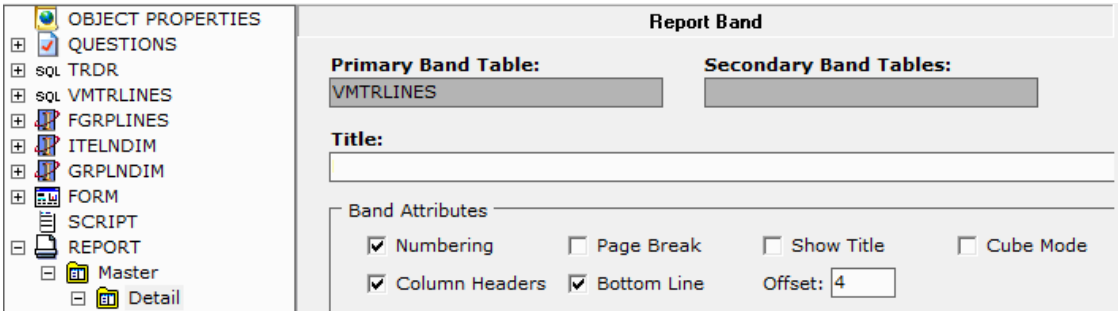


Figure H5

Fields entered in the tab “**Band Fields**” of the form node have the following properties (Figure H6):

Title: Column header caption.

Master headers are declared using the following steps in the "Title" property of the first column:

ColumnName@MasterName:NumberOfColumns

- **ColumnName:** Name of the table field
- **MasterName:** Sets the name for the master (merged) column
- **NumberOfColumns:** Number of merged columns for the master column

Size: Sets the width size of the column

Debit / Credit: Calculates the debit / credit according to the value sign

Totals: Displays the totals for a field. Mandatory for fields used in cumulative fields.

Transfer Value: Totals will be printed in each page.

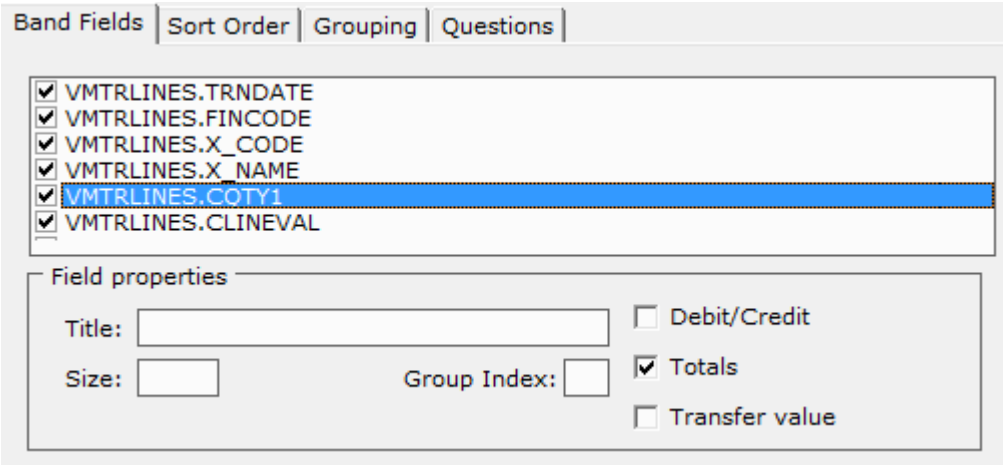


Figure H6

The following function SetBandXProperty is used for changing the attributes of the field and has the following parameters:

function SetBandXProperty(Params1,Params2,Params3,Params4)

Params1: Band

Params2: Band Caption

Params3: Property Type (1=Caption, 2=Width.Decimals, 4=Visible, 7=HasSums, 8=DebCred, 9=Transfer)

Params4: Property Value

H.2 Report Object Example

The following example creates a Report, that displays data from different SQL queries, depending on the filter selection. That is, using the filter Sodtype (Module) and selecting either Customers or Items, the respective data are displayed, with concurrent change of the captions and the number of columns.

Create a virtual table named CCCVTFILTERS, which will be used for the filters of the report.

This report uses one filter only, so the filter table will have only one field, SODTYPE (Title: Module) displaying only the customers and items, thus, you must enter in Editor property the command \$SODTYPE(W[13,51]) (Figure H3).

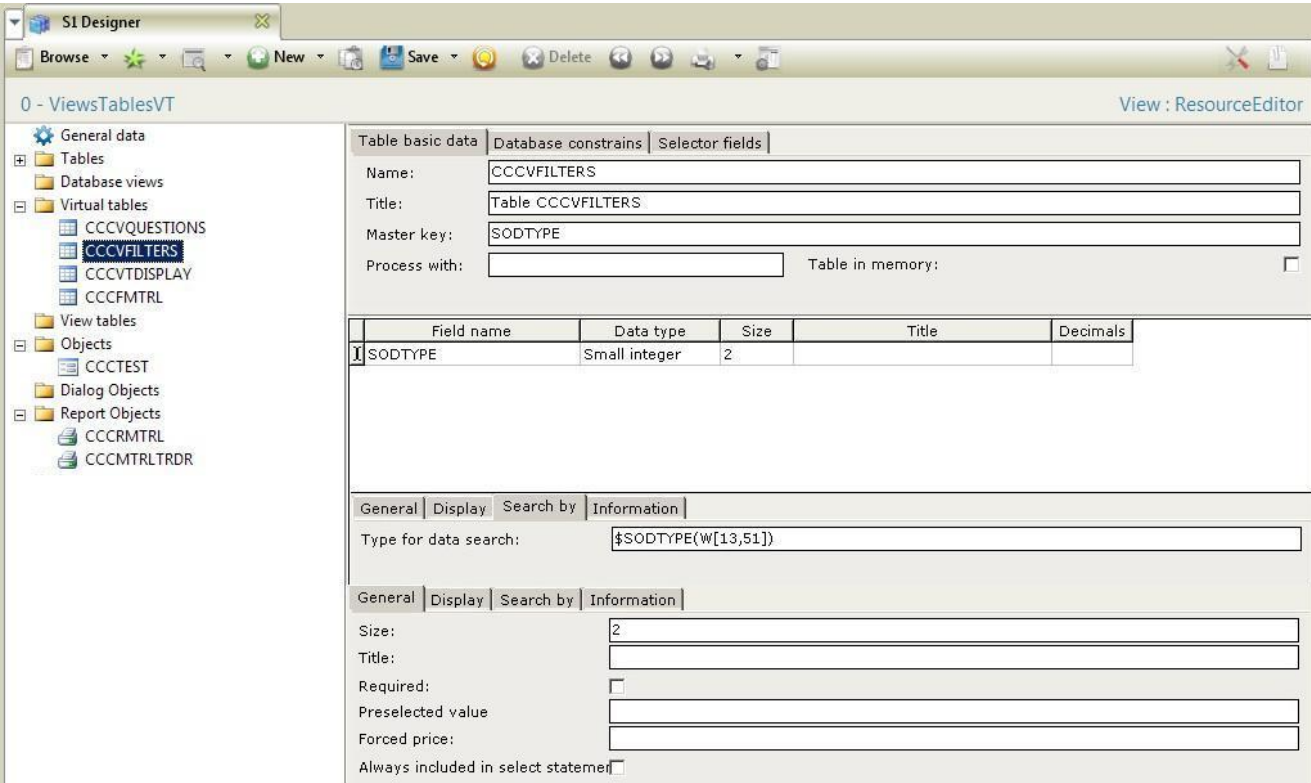


Figure H3

Createa virtual table named CCCVTDISPLAY, which has the columns that will be displayed in the report (FigureH4).

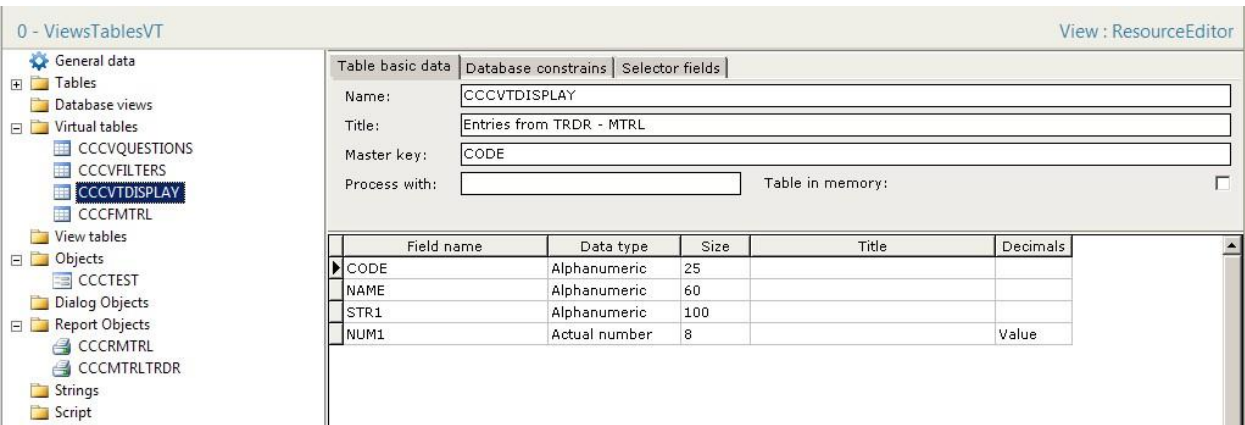


Figure H4

Create a Report Object and enter the parameter commands (Figure H5):

CODE=SoScriptCodeClt

SCRIPTSOURCE=SOIMPORT

SOIMPORT=SOIMPORT,CODE,SOIMPORT

SCRIPTNAME=CCCMTRLTRDR (the script that will be executed and create the report)

Insert the SBSL script as in Figure H6, which in this example executes different SQL queries to display columns according to the "Module" filter.

The commands used in CallPublished to change the attributes of the report grid are shown in Figure H7.

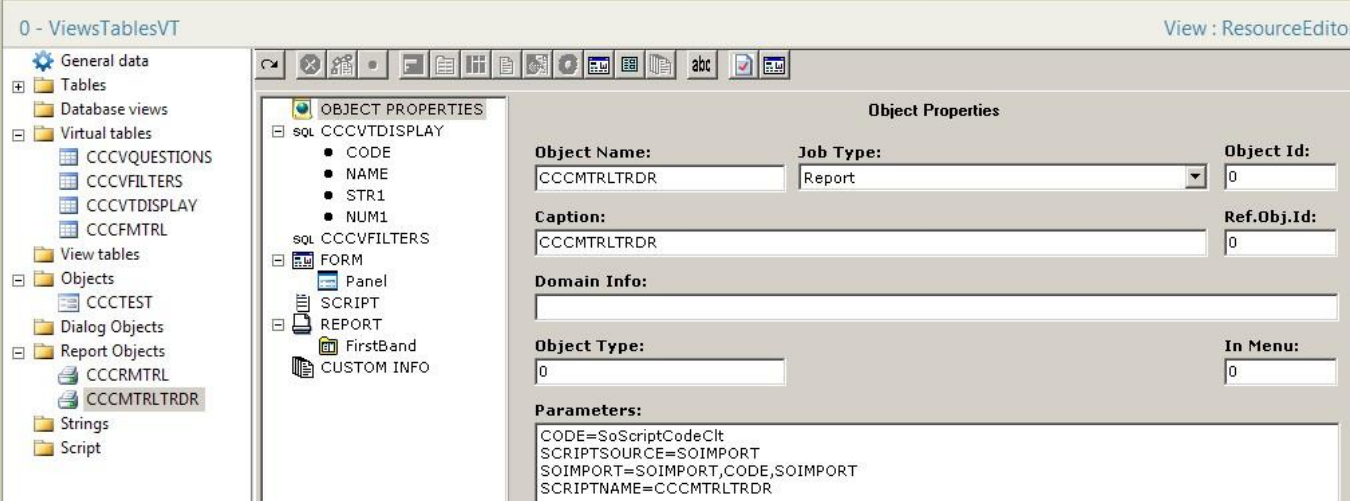


Figure H5

Name:	CCCMTRLTRDR	Name:	CCCMTRLTRDR
Comments:			

```
var vSQL,x,vQuestions,vSodtype,vBand;

Connect Xplorer DBDemo
{
  connect();
}

function Initialize()
{
  vBand=CallPublished('ModuleIntf.GetBand', VarArray(XModule,'FirstBand',2));
}

function OnStartReport()
{
  vQuestions=CallPublished('ModuleIntf.GetDataSet',VarArray(XModule,'CCCVTFILTERS',2));
  vSodtype=CallPublished('ModuleIntf.GetFieldValue',VarArray(vQuestions,'SODTYPE',2));

  if(vSodtype=13)
  {
    x=CallPublished('ModuleIntf.SetBandXProperty', VarArray(vBand, 'CCCVTDISPLAY.NUM1', 4, 0, 4)); //VISIBLE FALSE
    x=CallPublished('ModuleIntf.SetBandXProperty', VarArray(vBand, 'CCCVTDISPLAY.NAME', 1, 'Ερωτηµα', 4)); //CHANGE CAPTION
    x=CallPublished('ModuleIntf.SetBandXProperty', VarArray(vBand, 'CCCVTDISPLAY.STR1', 1, 'Διεύθυνση', 4)); //CHANGE CAPTION
  }
  if(vSodtype=51)
  {
    x=CallPublished('ModuleIntf.SetBandXProperty', VarArray(vBand, 'CCCVTDISPLAY.NAME', 1, 'Περιγραφή', 4)); //CHANGE CAPTION
    x=CallPublished('ModuleIntf.SetBandXProperty', VarArray(vBand, 'CCCVTDISPLAY.STR1', 1, 'Barcode', 4)); //CHANGE CAPTION
  }
}

function ReportFunction(ADataSet)
{
  var vSelect;
  if(vSodtype=13)
  {
    vSQL= ' SELECT CODE,NAME,ADDRESS FROM TRDR WHERE COMPANY=1000 AND SODTYPE=13';
    x=GetQueryDataSet('CCCVTDISPLAY', 'DBDemo', vSQL, Null );
  }
  if(vSodtype=51)
  {
    vSQL= ' SELECT S1.CODE,S1.NAME,S1.CODE1,S2.QTY1 FROM MTRL S1, MTRDATA S2 WHERE S1.MTRL=S2.MTRL AND S2.FISCPRD=2009 '
    + 'AND S1.COMPANY=S2.COMPANY AND S1.COMPANY=1000 AND S1.SODTYPE=51';
    x=GetQueryDataSet('CCCVTDISPLAY', 'DBDemo', vSQL, Null );
  }
}

{
}
```

Figure H6

Caption = 1 Width = 2 Decimals = 3 Visible = 4 FieldType=9
HasSums=7 DebCred=8 Transfer=9

Figure H7

7. EXTRA TOOLS

- A. [Auto Login from Windows Shortcut](#)
- B. [Maximum Entries per Module \(Select Top\)](#)
- C. [Menu Jobs](#)

A. Auto Login from Windows Shortcut

The application allows you to automatically login to a specific database, with a specific user, company and branch, using only a Windows shortcut.

A.1 Create a Windows Shortcut

Autologin works by adding the switch /XCO:xcopath to SoftOne application shortcut. First you need to create a shortcut for SoftOne file Xplorer.exe and then alter its "target" textbox by entering the path of a XCO file that includes the database login information.

You can copy the XCO file in the SoftOne folder and then add the target command as in Figure A1.

C:\Soft1\Xplorer.exe /xco:"C:\Soft1\MyCompany.XCO"

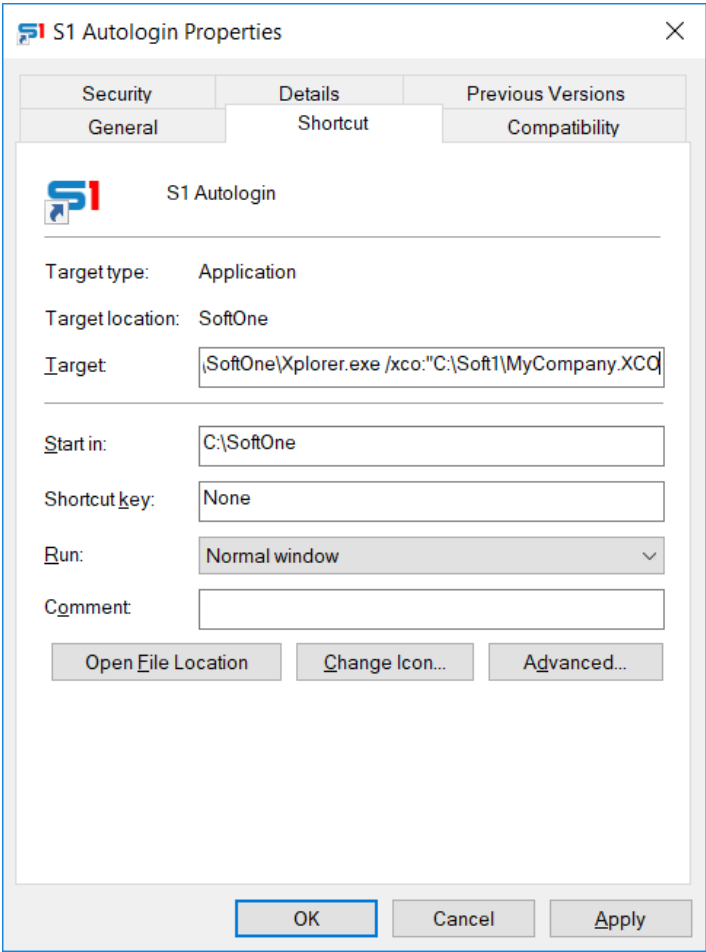


Figure A1

A.2 Configuration of XCO connection file

A.2.1 On-Premise Installations

Auto login is activated only if you manually add a section [LOGIN] inside the connection file XCO using the following commands.

XCO File
<pre>[LOGIN] USERNAME=xxx PASSWORD=xxx COMPANY=xxx BRANCH=xxx</pre>

A.2.2 Cloud Installations

Auto login in cloud installations is done by adding a section [LOGIN] inside the Params.cfg file, as in on-premise installations and also adding the command **GO=1** in order to work properly.

Params.cfg File

```
[SAAS]
SaasInstallation=SaasInstName (use any name)

[SaasInstallation]
USERNAME=xxx (serial number of the installation)
PASSWORD=xxx
DONTSHOWAGAIN=1

[LOGIN]
USERNAME=xxx
PASSWORD=xxx
COMPANY=xxx
BRANCH=xxx
GO=1
```

Auto login in cloud installations can also be done using a .xco file as in on-premise installations, by adding a [SAAS] section as in the following example (A.2.3 – MyLocalXco.xco) and using also the switch /xco:C:\...\MyLocalXco.xco in the application shortcut.

A.2.3 Schedule Jobs – Cloud Installations

Job scheduling is done using a txt file that describes the SoftOne job, as in the following example (MyJob.txt). The path of this file is used inside the application shortcut after the switch **/execute**.

Example: C:\...\xplorer.exe /execute:C:\...\MyJob.txt

The Params.cfg file is not needed, because its sections are described inside the xco file.

Note that when you use autologin for scheduling SoftOne jobs it is preferred to use the switch **/noversion** in the application shortcut, so as to avoid job interruptions when new versions are released. In this case, the SoftOne folder that is used for scheduling jobs can be manually synced.

Attention: Inside the xco file, always use the **plain text** password of cloud installation and not the encrypted one.

MyLocalXco.xco

```
[SAAS]
SERVER=SAAS.AZURE.ONCLOUD.GR
USERNAME=xxx (serial number of the installation)
PASSWORD=xxx (plain text - not encrypted)
DONTSHOWAGAIN=1

[LOGIN]
USERNAME=xxx
PASSWORD=xxx
COMPANY=xxx
BRANCH=xxx
GO=1
```

MyJob.txt

```
TYPE=BATCH
OBJECT=CLIENTIMPORT,SCRIPTNAME:MYJOB
XCOFILENAME=MyLocalXco.xco
```

A.3 XCO file Commands

In XCO file you can also add other commands, which affect the way SoftOne is displayed.

HIDEBAR = 1

Hides the main toolbar (Back – Forward - History - Menu selection...)

HIDEMENU = 1

Hides the user menu. It is usually combined with the command that displays an object after autologin (e.g. EXEC=RETAILDOC)

HIDEXPLORER = 1

Minimizes the application to Windows tray. It is usually used with the object you wish to display after login, hiding the form of the application (e.g. EXEC=SALDOC)

EXEC = Objectname

Opens the "Objectname" object after autologin (e.g. Sales, Customers, Items, e.t.c.).

You can also use any of the EXEC commands when opening an object ([Menu Job Parameters](#)).

The following example opens the Retail Documents object in a new entry, in Fullscreen and hides the Related jobs.

EXEC=RETAILDOC[FORM=RETAIL, AUTOEXEC=2, MAXIMIZE=1, RELJOBS=0]

B. Maximum Entries per Module (Select Top)

B.1 Overview

The utility "Select Top" can be used to specify the number of entries that can be displayed per object. This means that by adding the command "ObjectName = x" to this tool, all the fields that link to this object (editors) including the browser of the object, display only the first x rows that meet the condition filters.

The above operation is done by executing the "SELECT TOP x" clause in the SQL statements of the object primary table. This can improve the performance of the application as long as it is used in a proper way.

B.2 Object Usage

"Select Top" tool is opened by inserting a new job in the menu, using as job type "System Tools" and command "**ACMD:ACSELECTTOP**" (Figure B2.1).

When running the above job, a blank window will be displayed, where you can define the objects that the "Select Top" will be applied, as well as the number of the entries that will be displayed (Figure B2.2).

After completing the above process, you can check that is working properly by opening one of the objects defined inside the "Select top" tool and running its browser. Note that it displays only the number of records that you have set in the "Select Top" window. You can also view the SQL statement through the SQL Monitor tool.

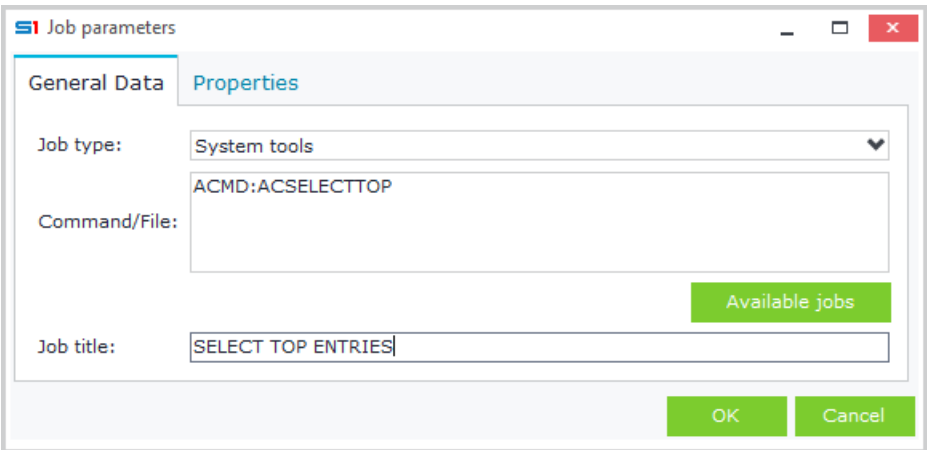


Figure B2.1

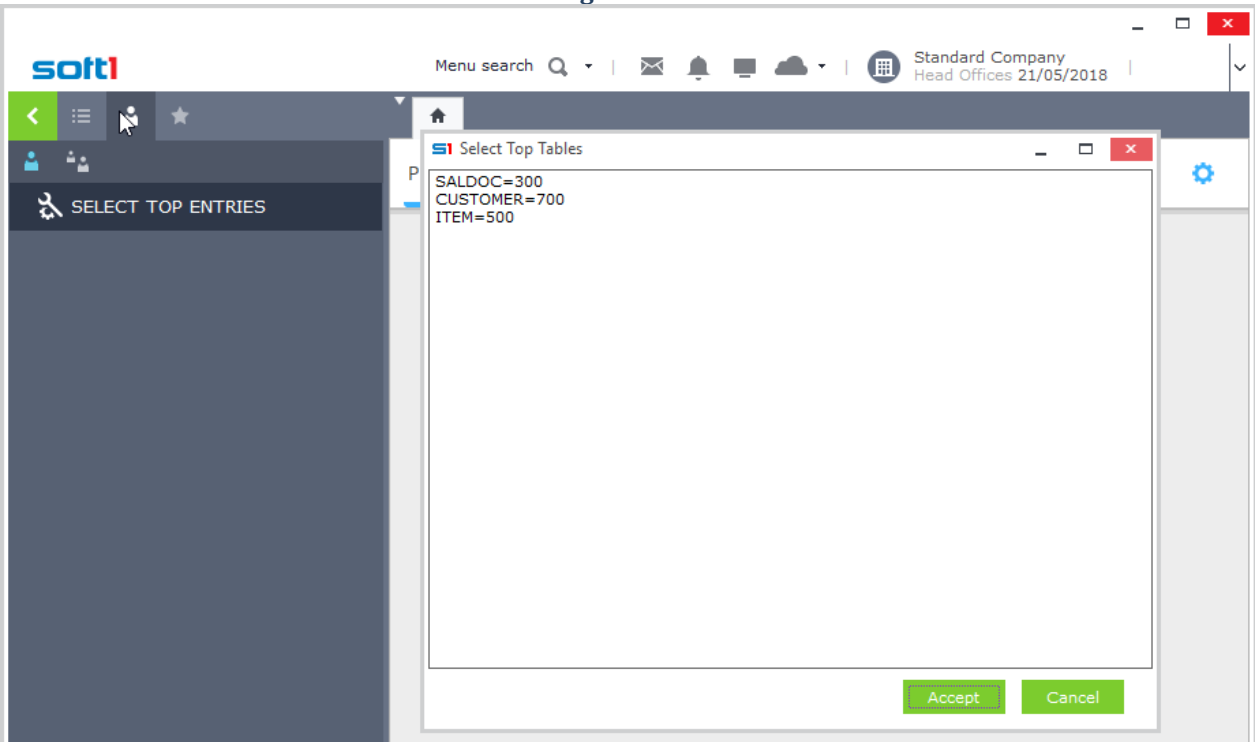


Figure B2.2

Notice that when you run a browser or a selector that uses any of the above objects then the SQL statement will use the SELECT TOP entries defined in the previous step (e.g. ITEM – Figure B2.3).

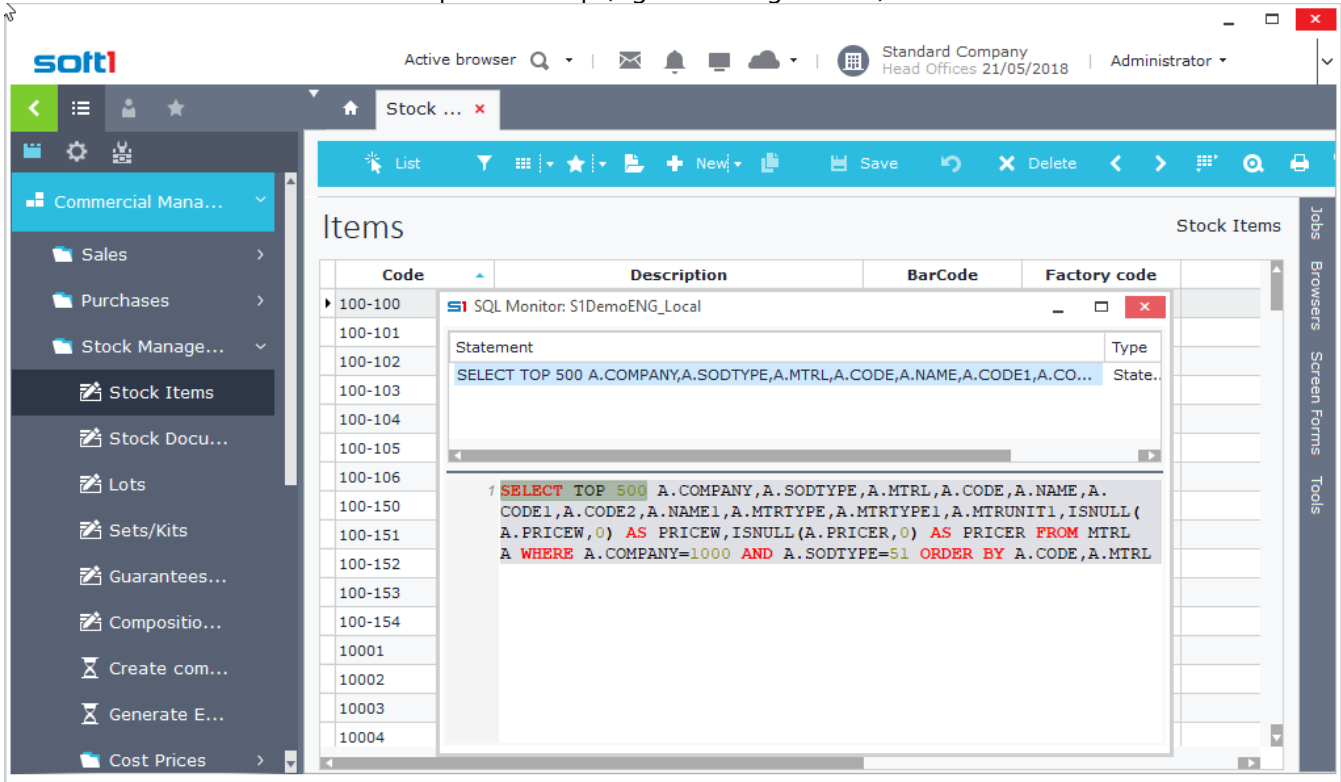


Figure B2.3

B.3 Browser Usage

Objects inserted in the tool **"Select Top"** affect all the controls that the object is displayed (browsers, reports, editors). However, there are a lot of times where you need to deactivate it or change the number of select records in a browser or report. To achieve this, you need to create a filter that lets the user select the number of records that will be displayed (null = All records). Follow the steps below:

- Create a user-defined field and type the command **SELECTTOP** in column "Name".
- Select **"Question"** in the column **"Operation"** and **"Number"** in column **"Field type"** (Figure B3.1).

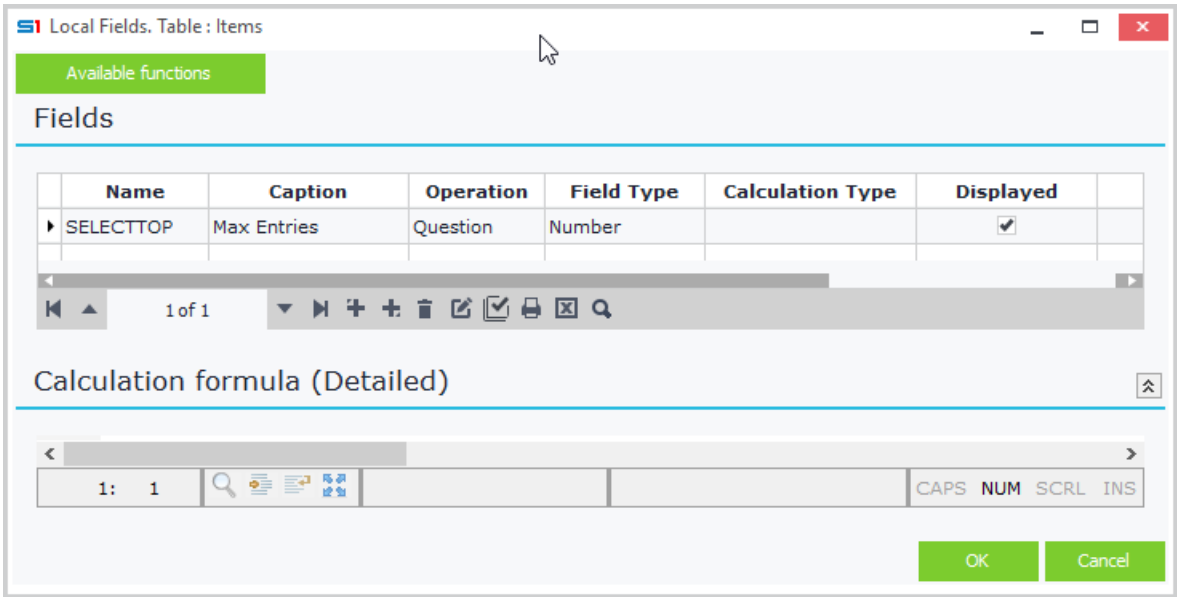


Figure B3.1

- Insert the defined field in the filters of the browser.
- Open the object and notice that the browser filters are displayed as in Figure B3.2.

Figure B3.2

- Enter a value in Filter “Max Entries” and run the browser (Figure B3.3).

Code	Description	BarCode	Factory code	Descript
100-100	Orange juice11111111			
100-101	Lemon juice			
100-102				
100-103				
100-104				
100-105				
100-106				
100-150				
100-151				
100-152				
100-153				
100-154				
10001				
10002				
10003	Digital Camera 8 MP	220.31.2.014	220.31.2.014	
10004	Photo michachi DSLR 10 MP	220.31.2.015	220.31.2.015	
10005	Flash Camcorder	220.31.2.016	220.31.2.016	
10006	Flash Full HD Camcorder	220.31.2.015	220.31.2.015	
10007	HDD 60GB Camcorder	220.31.2.015	220.31.2.015	
10008	Digital Photo Frame 8 "	221.21.2.017	221.21.2.017	
10009	Home cinema 1000 Watt	221.31.2.018	221.31.2.018	

Figure B3.3

All the records of the object are displayed if you leave the “Max entries” filter field **blank**.

Note: The “*SELECTTOP*” user-defined field can be used in any browser or report even if there is no reference for its object inside the “Select Top” tool (Section B.2).

C. Menu Jobs

The application allows you to configure the menu by creating jobs that open objects (Sales documents, Customers, Items) using specific browsers, forms, templates etc. and not the default ones. They also give you the option of adding some extra commands such as locking the Toolbar, opening in insert mode, use specific template etc.

These jobs might be used in custom group or user menus in a way that each group or user can be set to view different browsers and forms for the same object.

User Menus are saved in the blob field SODATA of the table CSTINFO (CSTTYPE=7).

```
Select * from cstinfo where sotype=802 and sodefault=0 and csttype=7 and cstname=name
and cstinfo=0 and name='UserName'
```

C.1 Create Job

The easiest way to create a job with a specific browser and form is to open an object, select the desired browser and form and then drag the tab and drop it in the menu (left pane).

Alternatively, you can do the following:

- Right-click the menu panel and select "New Job" (Figure C1.1).
- In "**Job type**" select "File Processing" for application objects or "Table processing" for memory tables. Job parameters are discussed in the next section.
- In "**Order/File**" type or select an object (i.e. ITEM). For memory tables add the symbol \$ before the name of the table (e.g. \$DISTRICT).

The tab "Properties" allows you to set if the job will open automatically on login and if users are permitted to close the job (Figure C1.2).

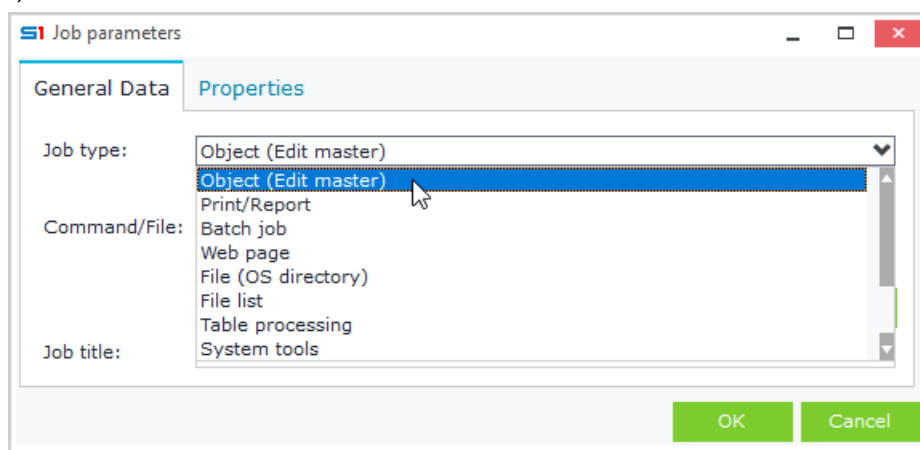


Figure C1.1

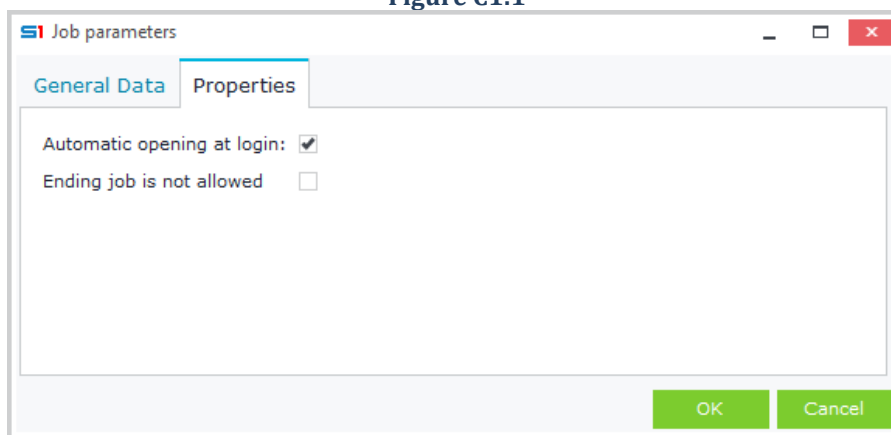


Figure C1.2

C.2 Job Types

The available job types are the following:

Object (Edit Master)

It is used to open an application object (e.g. SALDOC)

Table processing

It is used to call a memory table (e.g. \$DISTRICT). Memory tables do not have browsers and forms.

Report

It is used to call an application report

Example

Job that opens the designed report named "MYREPORT"

REPORTGEN [LIST=MYREPORT]

Batch Job

It is used to open batch jobs: Dialog objects or SBSL scripts.

Example

Job that opens the SoftOne script named "myscript"

FORMIMPORT,SCRIPTNAME:MYSCRIPT

Web Page

It is used for opening a Webpage

Other files

It is used for opening files of the computer

File list

It is used for opening a folder of the computer

System Tools

It is used for opening Softone internal tools

Example

Job that opens the SoftOne tool "SELECTTOP"

ACMD:ACSELECTTOP

Qlik View

It is used for opening Qlik View files

Dll Form

It is used for calling forms from dll files

Example

Job that opens the form "MyMainForm" from the file "Mydll.dll" (file is in the application folder)

.Mydll.dll;MyMainForm

C.3 Menu Job Parameters – Menu Commands

The following table shows the available menu commands:

Menu Commands	
Command	Operation
AUTOEXEC= 1	Runs the browser using the default filters.
AUTOEXEC=2	Opens an object in insert mode (new entry).
AUTOLOCATE=HeaderID	Locates an object using the given id. Example: SALDOC[AUTOLOCATE=620]
BROWSERONLY= 1	Opens an object displaying only its browser.
CUSTOM=0	Locks the toolbar of the object. Use this command when you want to prevent the user from selecting from the available browsers and forms.
EXTRALOCATE=LineID	Locates a datagrid record using the given id. Combined with AUTOLOCATE. Example: SALDOC[AUTOLOCATE=620,EXTRALOCATE=5] (FINDOC=620, MTRLINES=5)
FORCEFILTERS = filename1:fieldvalue1? filename2:fieldvalue2	Runs the browser with the filters specified. Example: CUSTOMER[LIST=MyList,AUTOEXEC=1,FORCEFILTERS=CODE=000-00*? DISTRICT=Athens]
FORCEVALUES = filename1:fieldvalue1? filename2:fieldvalue2	Opens the form of the object in a new entry with the values specified Example: CUSTOMER[AUTOEXEC=2,FORCEVALUES=TRDCATEGORY=3000?TRDBUSINESS=1]
FORM=FormName	Opens the form named "FormName".
LIST=ListName	Opens the browser named "ListName".
MAXIMIZE= 1	Opens the window in Full Screen mode.
NOGRIDTOOLBAR =1	Hides the toolbar of all the grids inside a form.
NOTOOLBAR=1	Hides the main toolbar of the given object.
NOSIDEBAR=1	Hides the sidebar (browsers, forms, tools) of the given object.
PHOTO=MyPhoto	Opens an object in insert mode using a specific template (MyPhoto).
RELJOBS=0	Hides the Related jobs of the given object.
SELECTION=Name	Opens the object in Browse mode using the given pre-selected list (of records)
STYLE=MODAL	Opens an object in Modal mode
STYLE=MODELESS	Opens an object in Modeless mode (new Tab)
STYLE=NEW	Opens an object in insert mode using Modal window (Save/Cancel buttons) Example: SALDOC[STYLE=NEW,FORCEVALUES=SERIES=7062?COMMENTS=TEST]
SELECTION=Name	Runs the browser using the selected predefined list

Menu Commands	
Command	Operation
EDITOPTIONS=NOINSERT	Prohibits adding a new entry and hides the toolbar button "New" Example: SALDOC[EDITOPTIONS=NOINSERT]
EDITOPTIONS=NOINSERT	Prohibits adding a new entry and hides the toolbar button "New" Example: SALDOC[EDITOPTIONS=NOINSERT]
EDITOPTIONS=NODELETE	Prohibits entry delete and hides the toolbar button "Delete" Example: SALDOC[EDITOPTIONS=NOINSERT?NODELETE]
EDITOPTIONS=READONLY	Disables editing the data of an entry Example: CUSTOMER[EDITOPTIONS=READONLY,AUTOLOCATE=55]
EDITOPTIONS=NOBROWSER	Hides the browser of an object and the toolbar button "Browser" Example: SALDOC[EDITOPTIONS=NOBROWSER?NOINSERT?NODELETE]
EDITOPTIONS=ONEROW	Displays an EditList object in "one record" mode
EDITOPTIONS=NOPRINTFORM	Disables Form printing and hides the toolbar button "Print Form" Example: SALDOC[EDITOPTIONS=NOPRINTFORM]
EDITOPTIONS=NOPRINTBROWSER	Disables Browser printing Example: SALDOC[EDITOPTIONS=NOPRINTBROWSER]
EDITOPTIONS=NOPRINTLABEL	Disables label printing (from browser right click menu) Example: SALDOC[EDITOPTIONS=NOPRINTFORM?NOPRINTLABEL]

C.4 Examples of Menu Jobs

Examples of the use of extra commands when running menu jobs are provided below.

Job parameters

General Data Properties

Job type: Object (Edit master)

Command/File: CUSTOMER[LIST=Customers-Transactions per Item,FORM=Customers]

Job title: Customers

Available jobs

OK Cancel

Figure C4.1 – Specific Browser and Form

Job parameters

General Data Properties

Job type: Object (Edit master)

Command/File: SALDOC[LIST=MySalesList ,FORM=MySalesForm,Custom=0]

Job title: Sales Documents

Available jobs

OK Cancel

Figure C4.2 – Lock Vertical Toolbar Browsers and Forms

Job parameters

General Data Properties

Job type: Object (Edit master)

Command/File: SALDOC[LIST=MySalesList ,FORM=MySalesForm,Autoexec=1]

Job title: Sales Documents

Available jobs

OK Cancel

Figure C4.3 – Run the Browser using default filters

Job parameters

General Data Properties

Job type: Object (Edit master)

Command/File: SALDOC[LIST=MySalesList ,FORM=MySalesForm,Photo=MyPhoto]

Job title: Sales Documents

Available jobs

OK Cancel

Figure C4.4 – Open the form in new record using a specific template (photo)

8. SCHEDULER & MESSAGES

- A. [Remote Server](#)
- B. [Windows Scheduler](#)
- C. [SoftOne Scheduler](#)
- D. [Messages – Reminder](#)

A. Remote Server

Remote Server is a SoftOne server that can execute application jobs, such as scheduled emails, SMS, batch and reports. Examples of jobs you can run through remote server are the following:

- Send emails through EDA (Alerts)
- Send SMS through EDA (Alerts)
- Run a report and export the result in any supported format
- Run a report in any available printer and send a confirmation message.
- Run a report, route the result to a cell phone or an e-mail and sent to the originator a confirmation message.
- Run a batch job.

Note: Remote Server uses one SoftOne license for running and executing the above jobs.

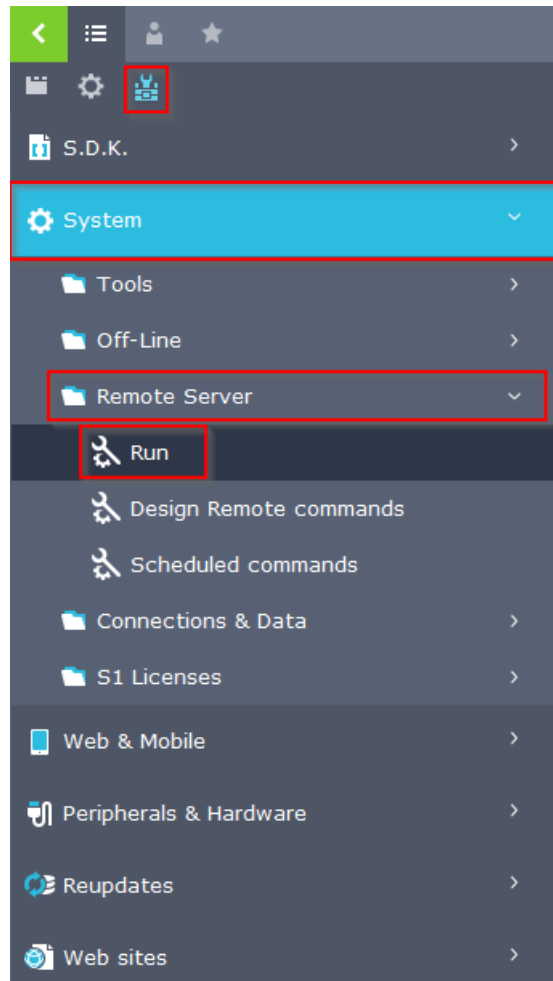


Figure A1

A.1 Activation

Open Remote Server using the settings menu: Tools → System → Remote Server → Run and follow the steps below to activate it.

- Use button **"Add device"** and select **"Database"** to activate the use of Remote Server for the current Database.
- Enable the option **"Auto activation"** and then press the button **"Activation"** (Figure A1.2).
- Select any other device (GSM Phone, E-mail) you need to activate, fill in the parameter fields and then select the button **"Save config"** to save the configuration settings as in Figure A1.3.

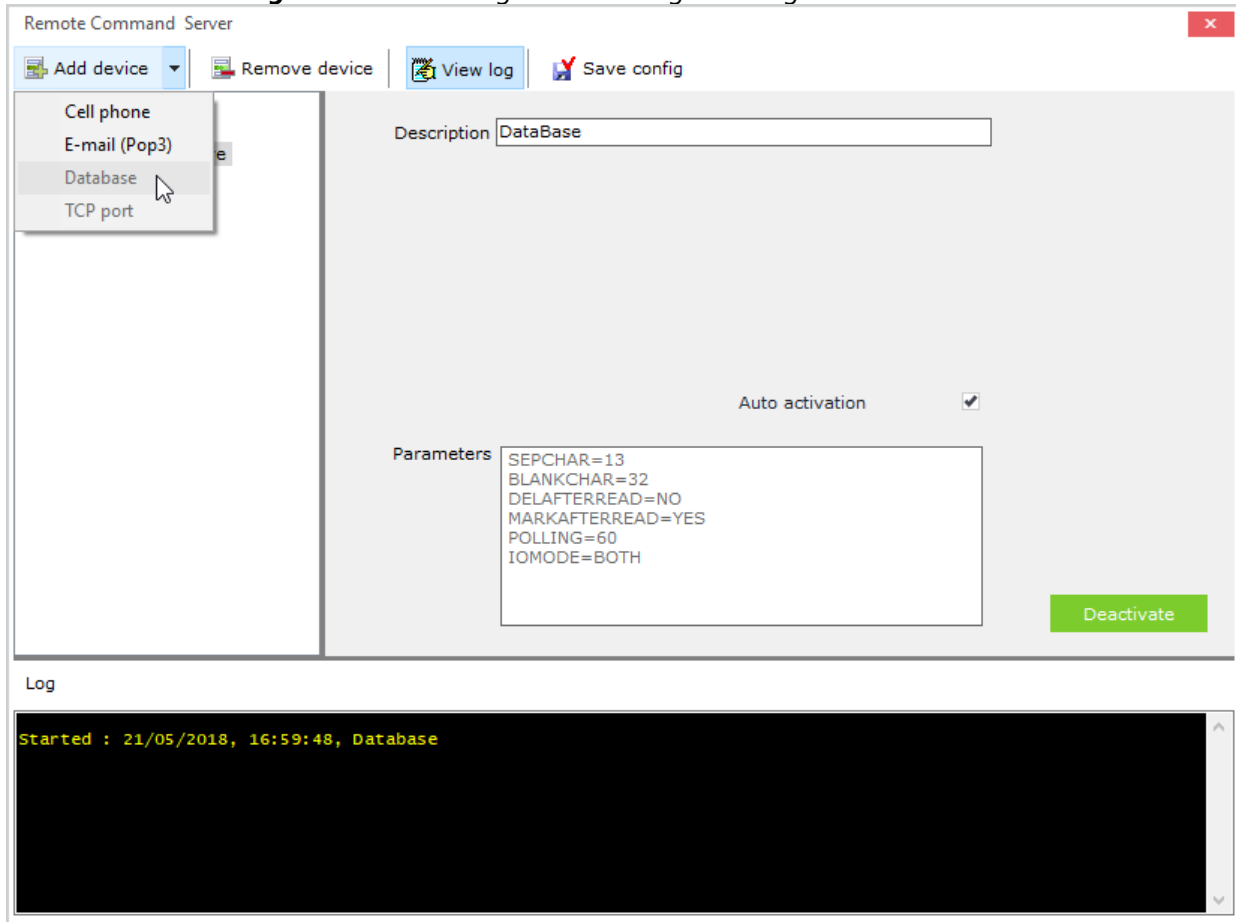


Figure A1.2

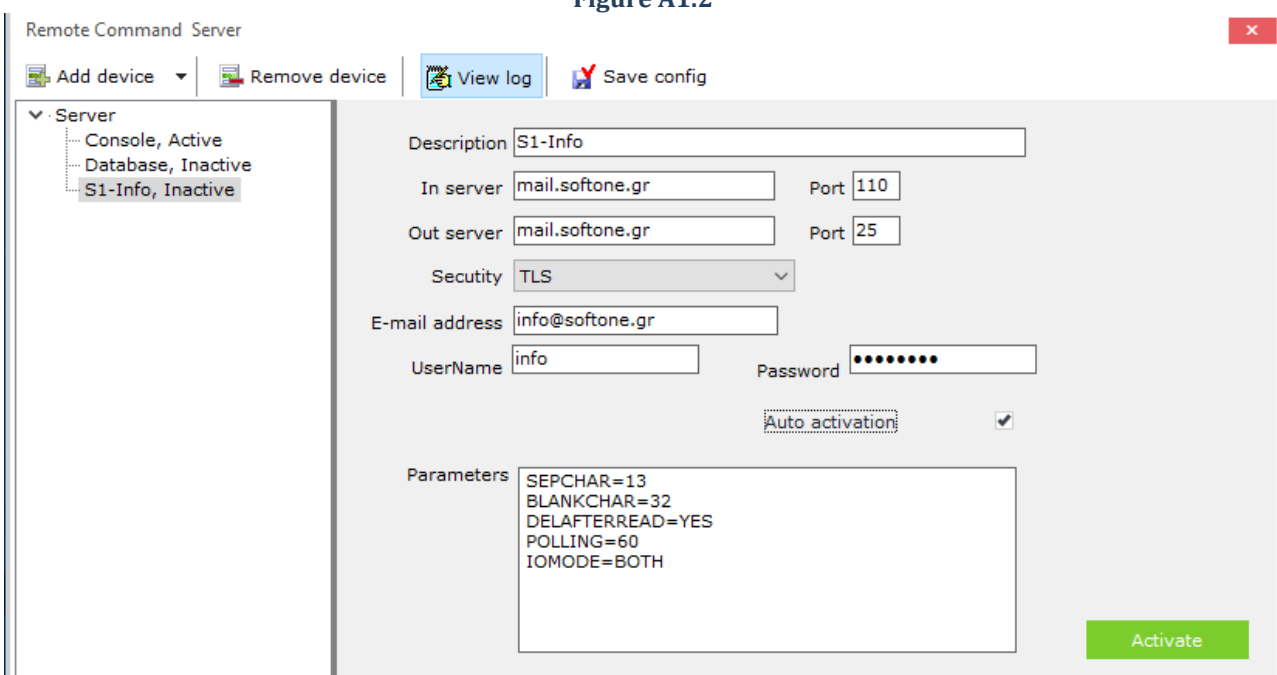


Figure A1.3

A.2 Remote Server Commands

All commands that can be used for running specific commands through the Remote Server menu are displayed in the table below.

Remote Server Commands		
Command	Parameters	Operation
TYPE	REPORT, BATCH, DIALOG, DESIGN, ANSWER, REPORTLIST, BATCHLIST, PRINTERLIST, COMPANYLIST, BRANCHLIST	Job type
OBJECT		Object selection (i.e. CUSTOMER)
JOBNAME		Name of Browser or designed Report
PHOTO		Template name
AUTOEXECUTE	0,1 (e.g. AUTOEXECUTE=1)	Auto run
OUTPUT	928, 437, EXCEL, WORD, METAFILE, PDF File, PrinterName	Sends Job Results to a specific File or printer
FILENAME		Name of the File for saving the results
SENDTO	MAIL, GSM	Send Results to Email or Cell phone
MAILADDR		Email Address
GSMNUM		Cell phone Number

The screenshot shows the 'Design remote commands' application. The main window displays a command definition for '18/01/2018- 500'. The command is defined as follows:

```

1 TYPE=ANSWER
2 SENDTO=MAIL:xxx@softone.gr
3 ANSWERSTR='Your Quote Is Ready- Order Code: OFFR000009'
  
```

A dialog box titled 'Διαθέσιμες εντολές' (Available commands) is open, listing the following parameters and their values:

- TYPE = REPORT, BATCH, DIALOG, DESIGN, ANSWER, REPORTLIST, BATCHLIST, PRINTERLIST, COMPANYLIST, BRANCHLIST
- OBJECT = <ObjectName ie:CUSTOMER>
- JOBNAME = <Custom list or Custom report name ie:NewList1>
- PHOTO = <Photo name (Report or batch dialog) ie:NewPhoto1>
- USERNAME = <Username ie:Admin>
- PASSWORD = <Password ie:SoftOne>
- COMPANY = <Company code ie:5>
- BRANCH = <Branch code ie:1>
- ANSWERSTR = <string>
- GSMNUM = <xxxxxxxxxx ie:6931607960>
- MAILADDR = <xxxx@xxxx.xx ie user@softone.gr>
- SENDTO = GSM[:gsmnum], MAIL[:mailaddr]
- OUTPUT = ASCII, 928, 437, EXCEL, WORD, METAFILE, <printerName>
- FILENAME = <Filename>
- REPLY = NOREPLY, STATUS, SHORT, LONG
- ACK = NO, YES
- STOPONERROR = FALSE, TRUE

The dialog box has a green button labeled 'Εξοδος' (Exit) at the bottom right.

A.3 Send SMS - Email

First thing you have to do in order to send messages using a cell phone from within the application is to setup and recognize the cell phone in Windows (using drivers).

Then select GSM Phone in the left pane and inside the property "**GSM port**" enter the Windows Com port where the cell phone is connected.

Inside the textbox "**Number**" enter the number of the connected phone, which will be used to send SMS. Finally, enable the parameter "**Auto activation**" and press "**Activation**" (Figure A3.1).

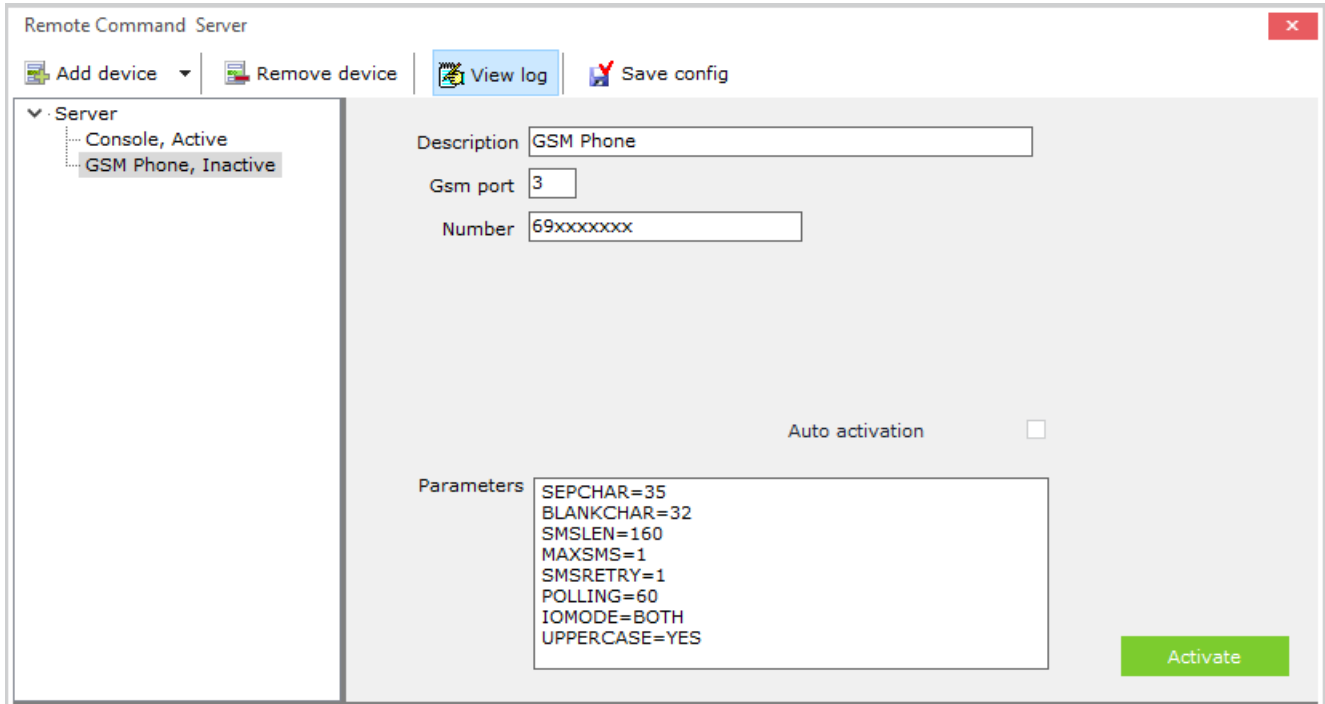


Figure A3.1

You can display the "in-progress" messages and jobs or the ones that are processed, using the menu "Remote Server → Scheduled commands" (Figure A3.2).

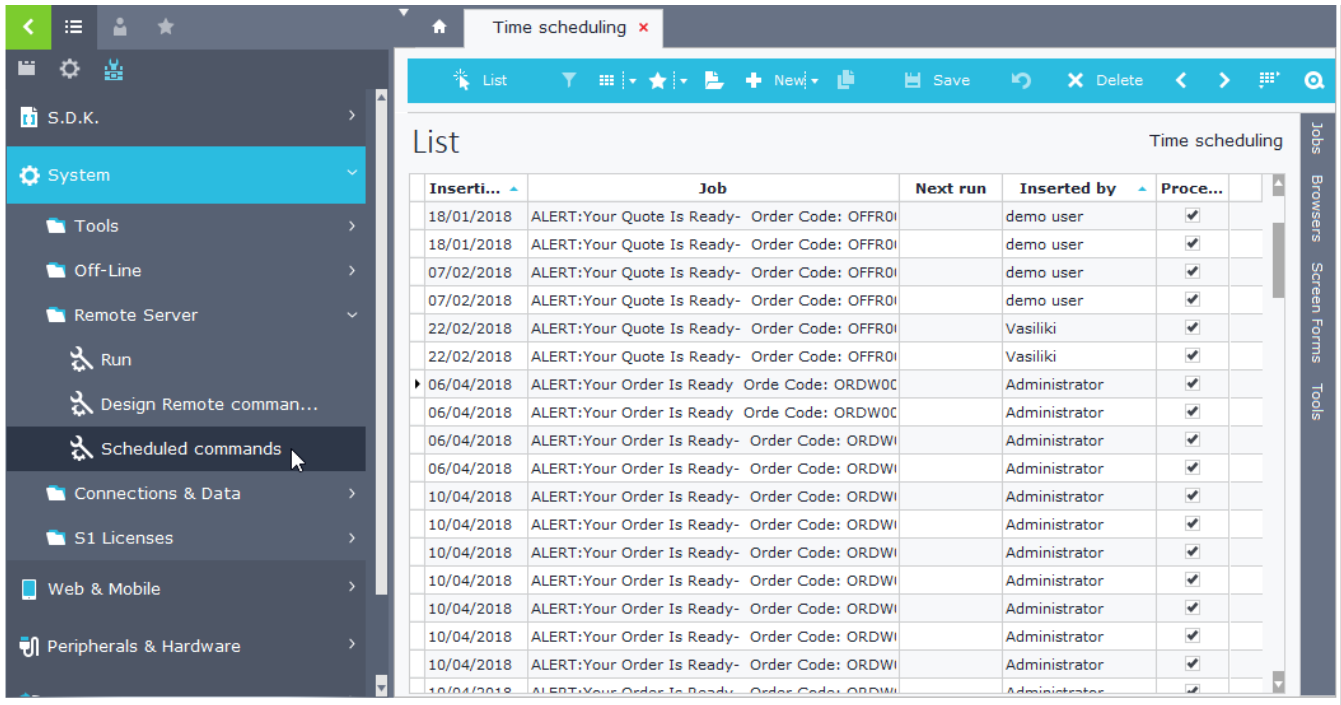


Figure A3.2

You also have the option to schedule messages or batch jobs (Figure A3.3).

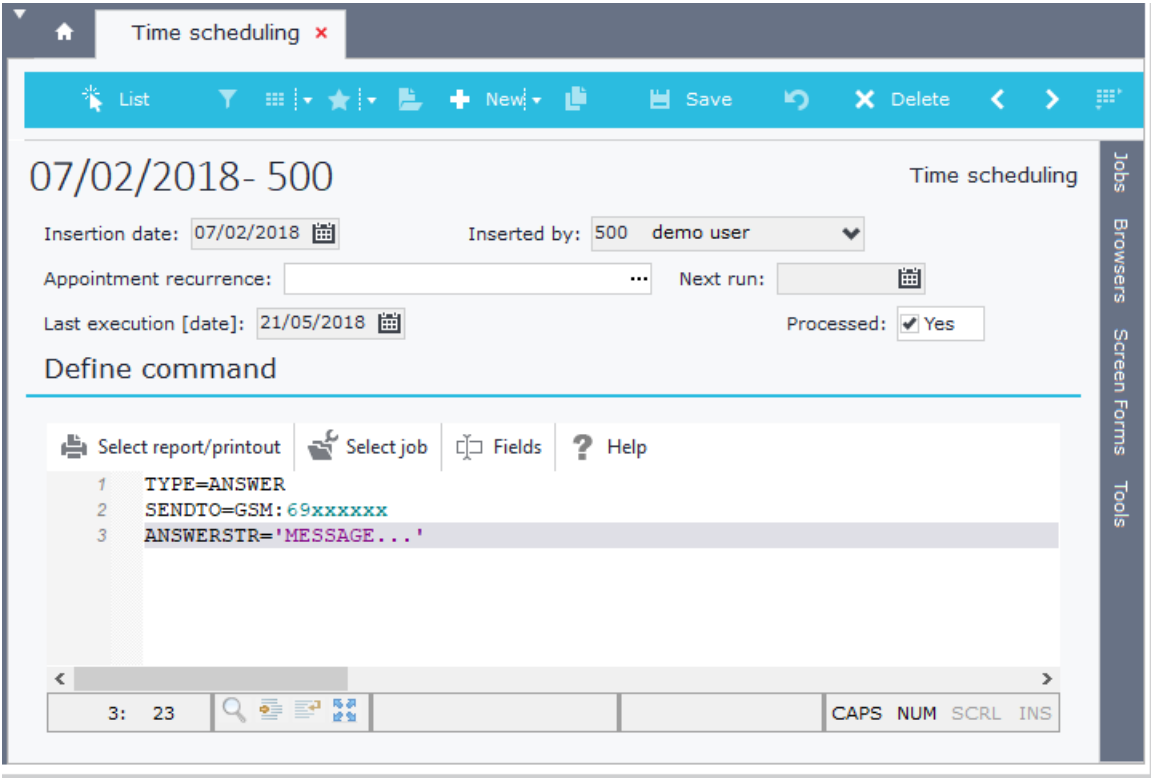


Figure A3.3

Remote Server can also be used to run any of the jobs you have created inside EDA (Alerts) and have as their type to send messages to email or user cell phones. The process is very simple, requiring only the activation of the devices (email or SMS) through the Remote Server, and the creation of EDA (Alerts) using email or cell phone types.

Example**Send multiple SMS messages to customers using fields from TRDR table.**

Select one or more customers from the customers' browser. Then, from the right-click pop up menu click on the option **"Send SMS"** (Figure A3.4).

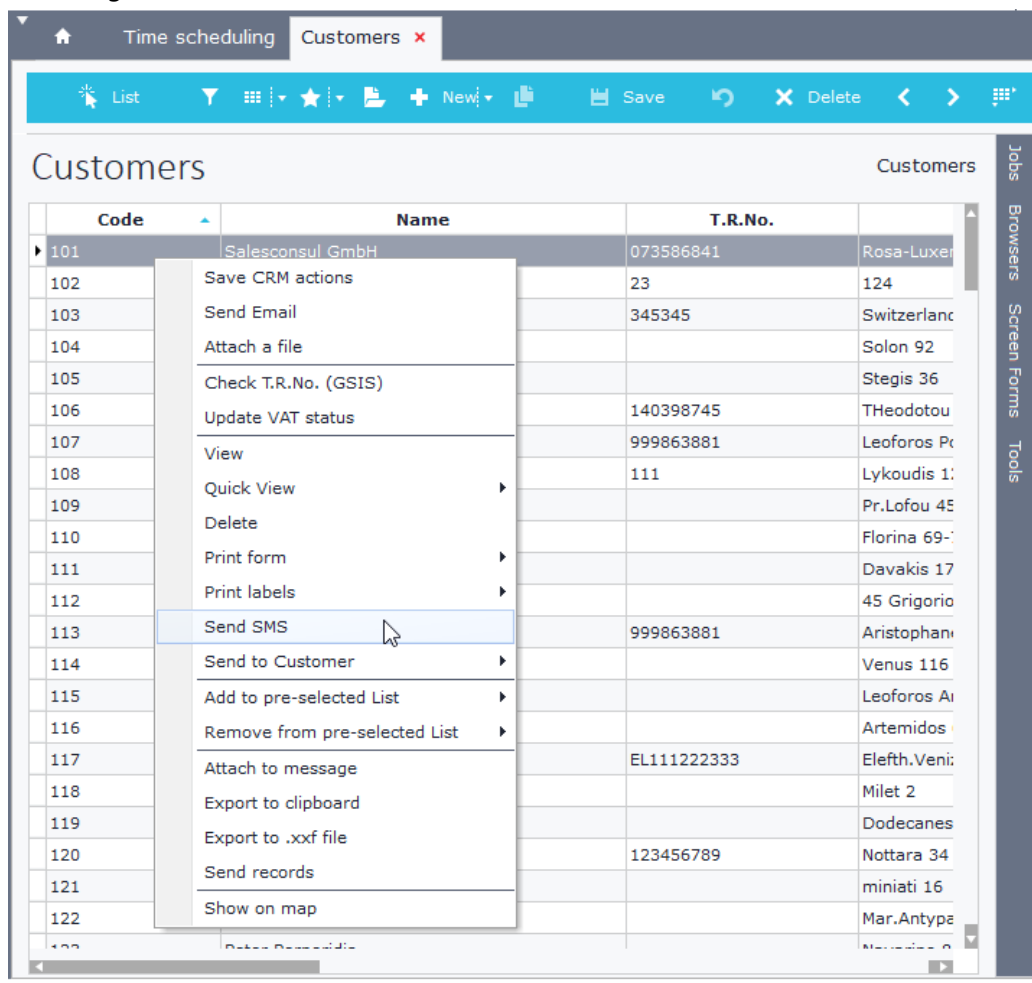


Figure A3.4

From the window that is displayed, select the button **"Fields"** and drag n drop the field Phone01 from customers in the field Recipient. Of course, you can enter the number directly if you wish to send a message to a specific phone number. In the textbox "Message" enter the message to send. However, you can use and drag in it any of the available fields. Finally, click the button "Send" to send the message.

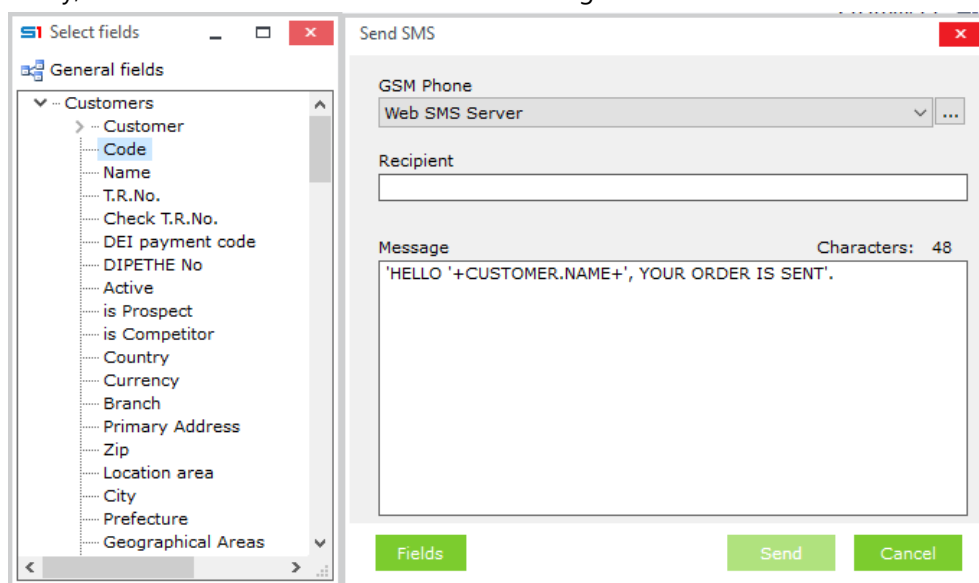


Figure A3.5

B. Windows Scheduler

B.1. Overview

Jobs running from Remote Server or SoftOne Scheduler can also run through Windows Scheduler, following the steps below, that are analyzed in the next sections:

- Create a text file which includes the SoftOne job
- Add AutoLogin section inside the XCO connection file with
- Create a Windows Scheduler task that runs SoftOne application using the SoftOne job.

All actions executed using the above mechanism, as well as the execution errors, are kept in a log file named **XECUTOR.LOG** which is created in the log folder of the program profile directory: C:\ProgramData\SoftOne\Log
Windows Scheduler execution requires that the option "Enable Scheduler/Messages" in SoftOne System Settings is not activated (Figure B1).

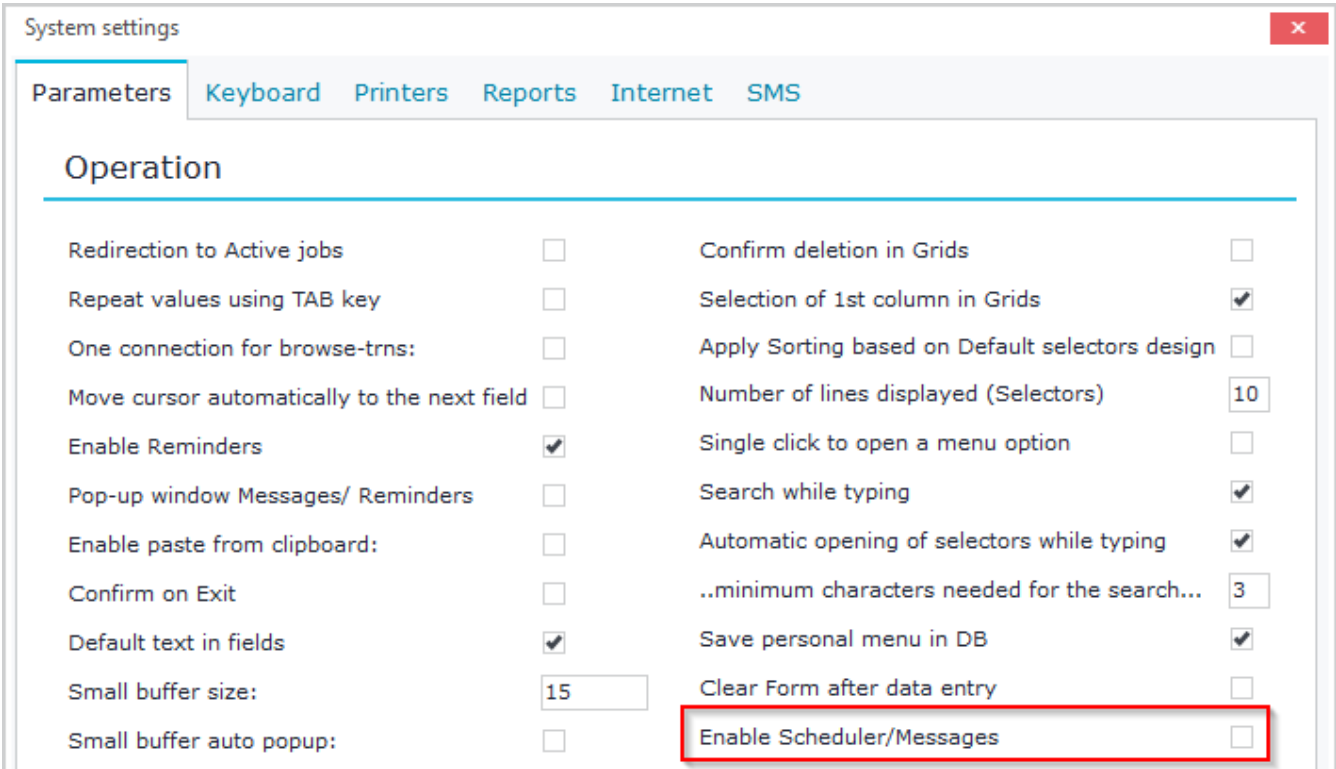


Figure B1

B.2 Job File

The SoftOne job that will be used to run through Windows scheduler must be added inside a text file. Create a .txt file and then enter the SoftOne job commands or copy them from a SoftOne Scheduler job. Add also the parameter XCOFILENAME=xxx.xco which describes the name of the XCO file as in Figure B2 and save it using any name (e.g. Soft1JobAutoRun.txt). The path of this file will be used in Windows scheduler, inside the application shortcut after the switch **/execute**. **Example:** C:\...\explorer.exe /execute:C:\...\Soft1JobAutoRun.txt

JOBNAME : Name of the job (can be any name)

TYPE=BATCH : Job type

OBJECT : Object or script to execute

XCOFILENAME : Name of the xco connection file

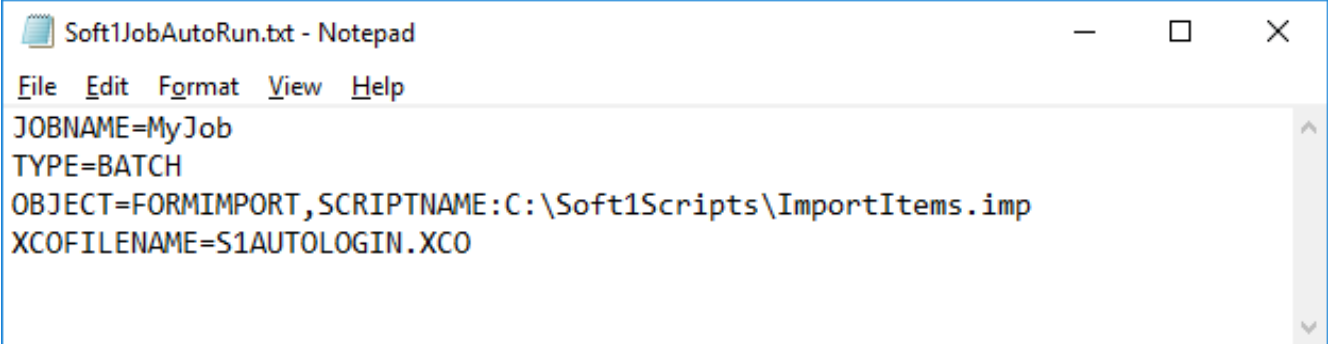


Figure B2

For **Cloud Installations** you need to create a .xco file inside the application directory as in the following example.

Attention: Inside the xco file, always use the **plain text** password of cloud installation and not the encrypted one.

S1AUTOLOGIN.XCO

```
[SAAS]
SERVER=SAAS.AZURE.ONCLOUD.GR
USERNAME=xxx (serial number of the installation)
PASSWORD=xxx (plain text - not encrypted)
DONTSHOWAGAIN=1

[LOGIN]
USERNAME=xxx
PASSWORD=xxx
COMPANY=xxx
BRANCH=xxx
GO=1
```

The Params.cfg file is not needed, because its sections are described inside the xco file.

Note that when you use autologin for scheduling SoftOne jobs it is preferred to use the switch **/noversion** in the application shortcut, so as to avoid job interruptions when new versions are released. In this case, the SoftOne folder that is used for scheduling jobs can be manually synced.

B.3 XCO Connection File

The connection file XCO must necessarily be located within the application folder.

Inside this file you need to add the section [LOGIN] (Figure B16), in order for the Windows Scheduler Task to auto Login to a specific database using a specific user, company and branch and then execute the selected job. You can use any text editor to open the XCO files.

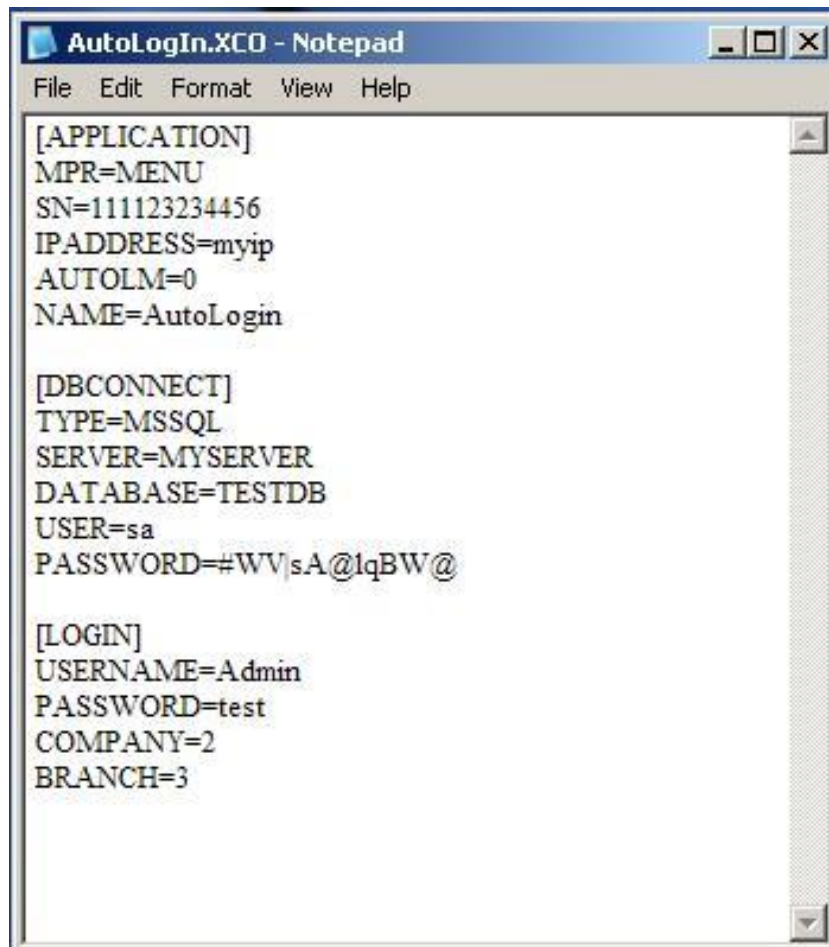


Figure B16

B.3 Windows Scheduler Task

Create a task in Windows Schedule using the following steps:

B.3.1 Windows XP

Open Windows Scheduler: Control panel → Scheduled task

Insert a new Task in Windows Scheduler (Add Scheduled Task) (Figure B3)

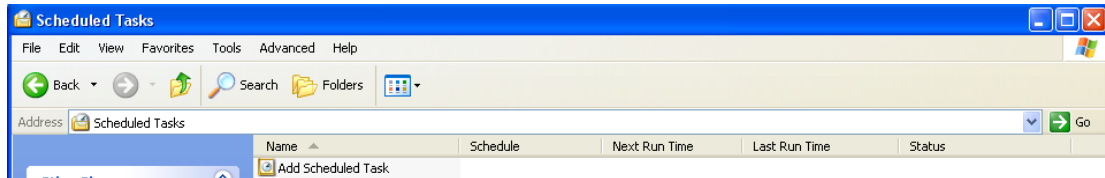


Figure B3

Use the button "Browse" to locate the application folder and then the file "Xplorer.exe" (Figure B4)

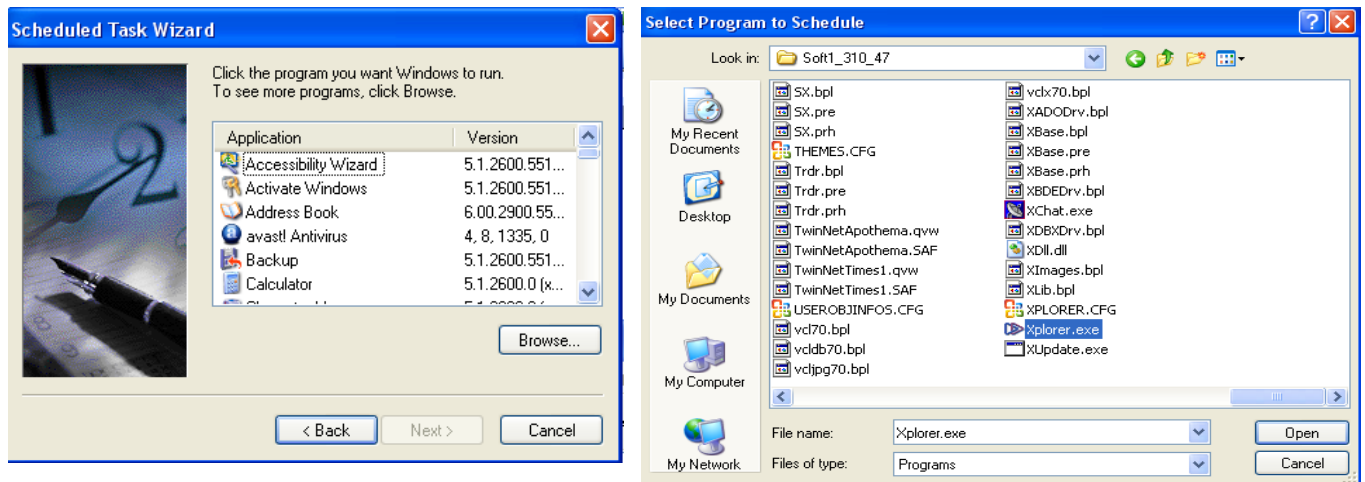


Figure B4

Select a name for the task and the execution intervals (Figure B5).

In the next window, enter the user credentials (Figure B6).

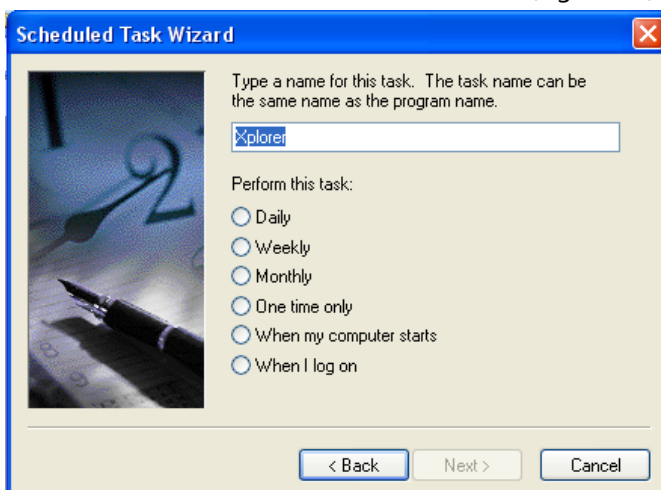


Figure B5



Figure B6

In the last window select the option “Open advanced properties” (Figure B7)

Click “Finish” to display the window shown in Figure B8.

Inside the textbox “Run”, after the path of Xplorer.exe file, enter the switch parameter
/EXECUTE:C:\TEST\Commands.TXT



Figure B7

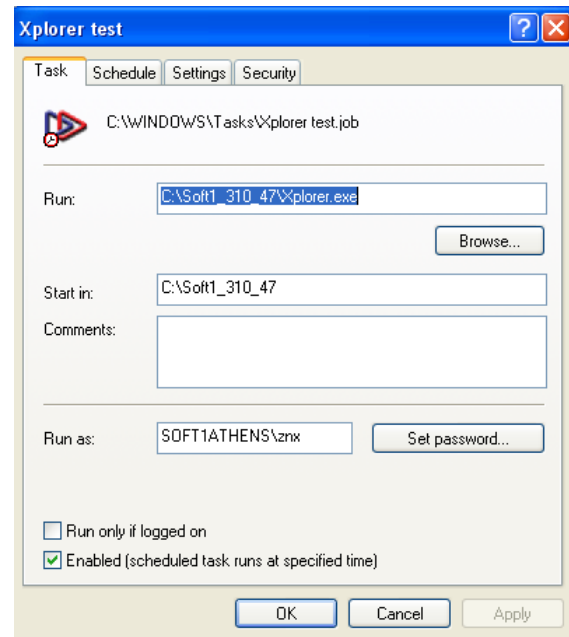


Figure B8

B.3.2 Windows 7

Open Windows Scheduler: Control panel → Administrative Tools → Task Scheduler (Figure B9)

Create a new task, either through right-click, or through panel Actions → Create Basic Task. (Figure B10)

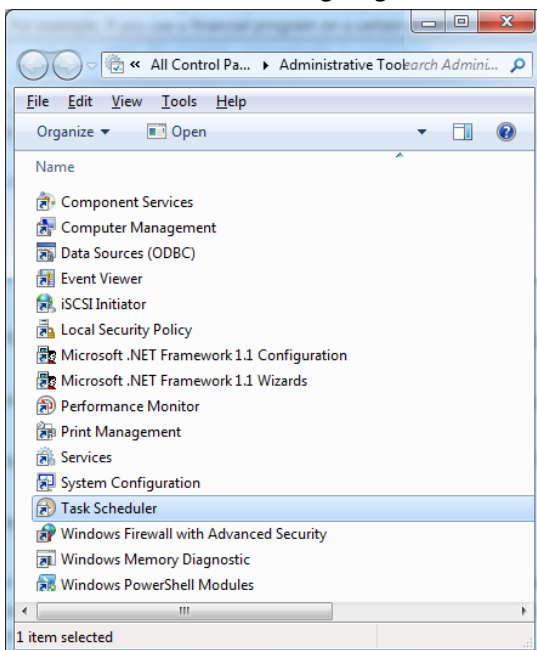


Figure B9

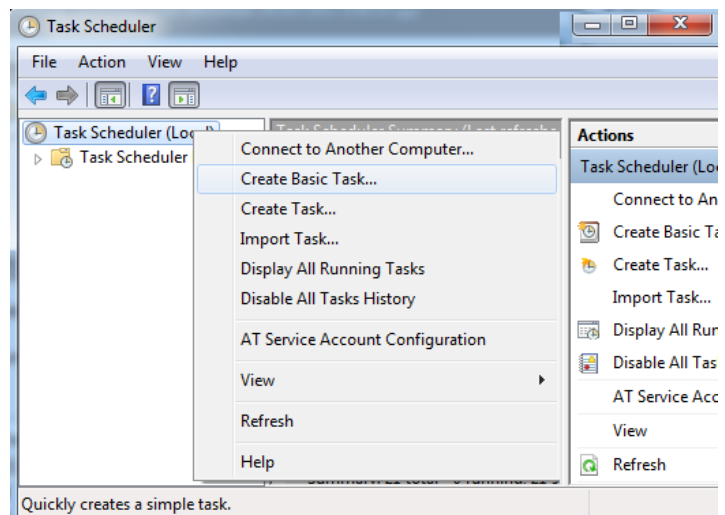


Figure B10

Enter a name and description for the task (Figure B11).

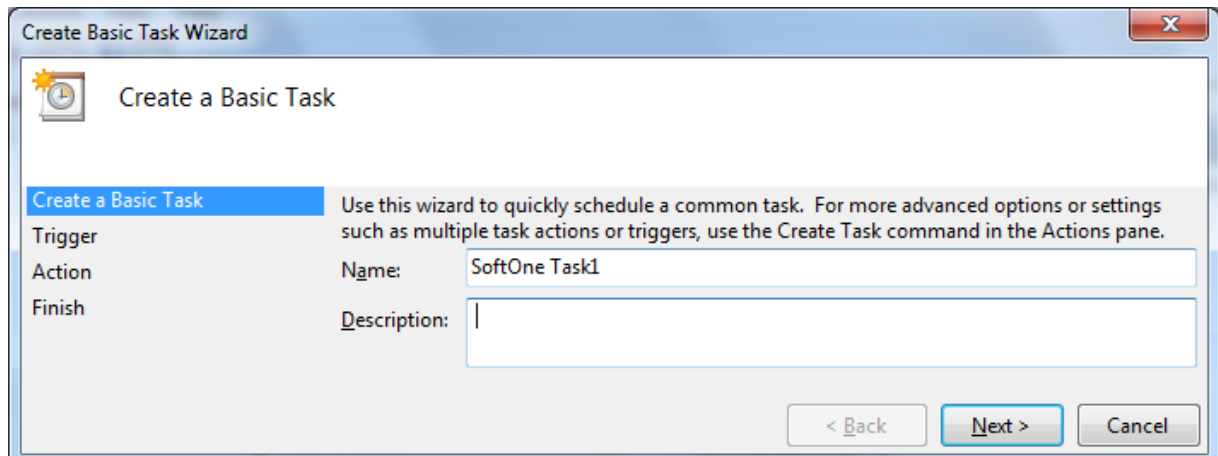


Figure B11

In the next window select the execution intervals (Figure B12).

In the next window, select "Start a program" (Figure B13)

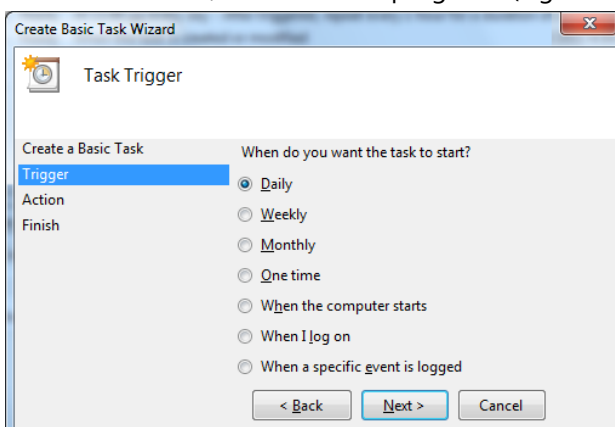


Figure B12

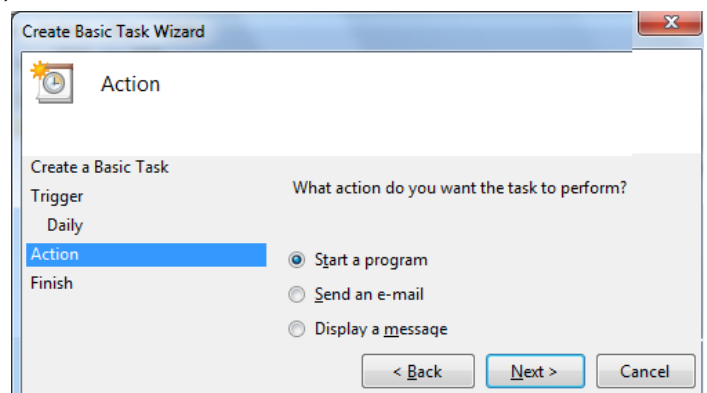


Figure B13

Use the button "Browse" to locate the application folder and then the file "Xplorer.exe" (Figure B14)

In the same window inside the textbox "Add arguments (optional)" enter the switch parameter /EXECUTE:C:\TEST\ENTOLES.TXT that will be executed after launching SoftOne.

The last window displays a summary of the actions that will run when executing this Task (Figure B15).

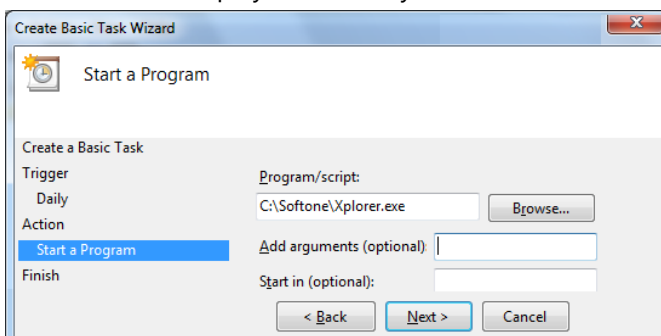


Figure B14

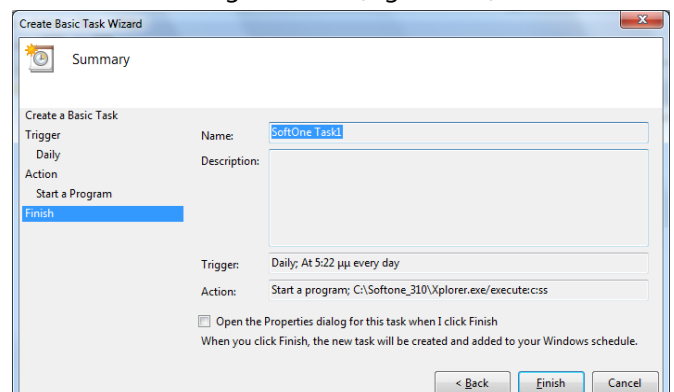
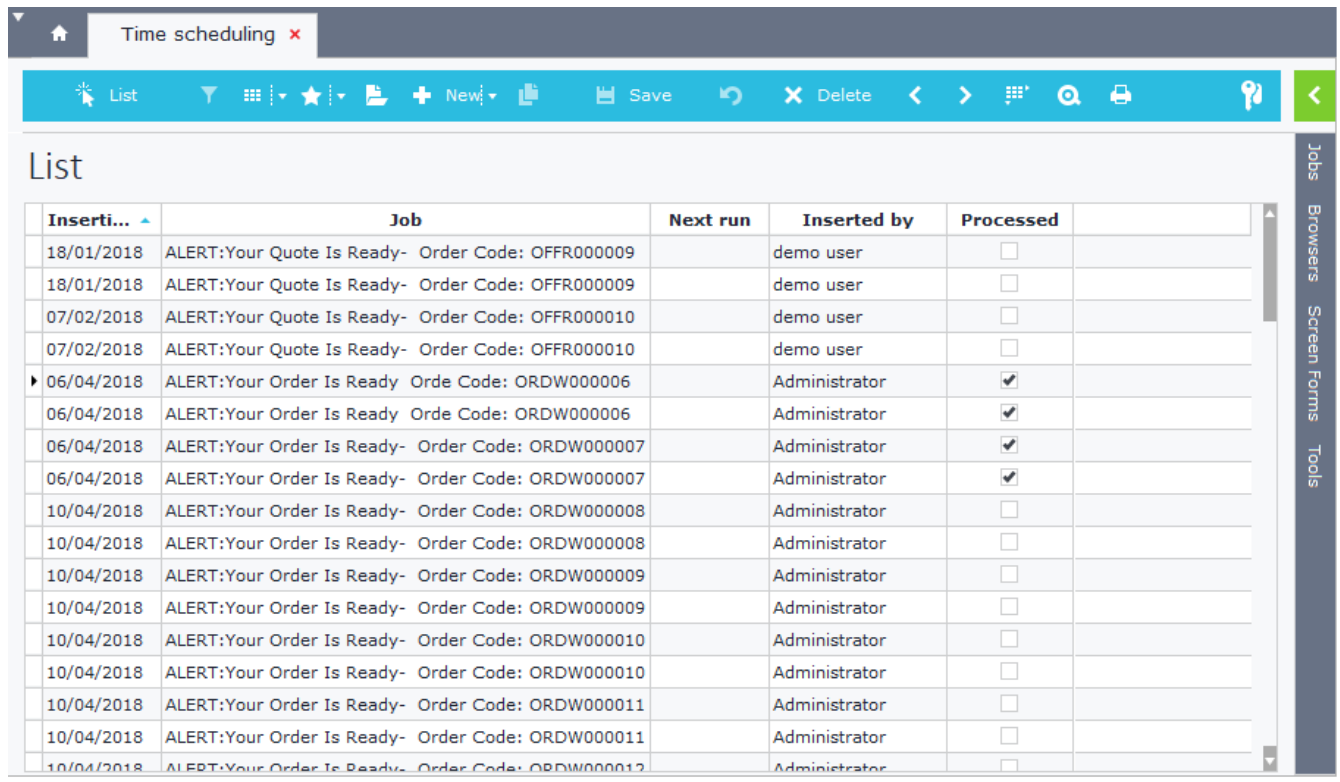


Figure B15

C. SoftOne Scheduler

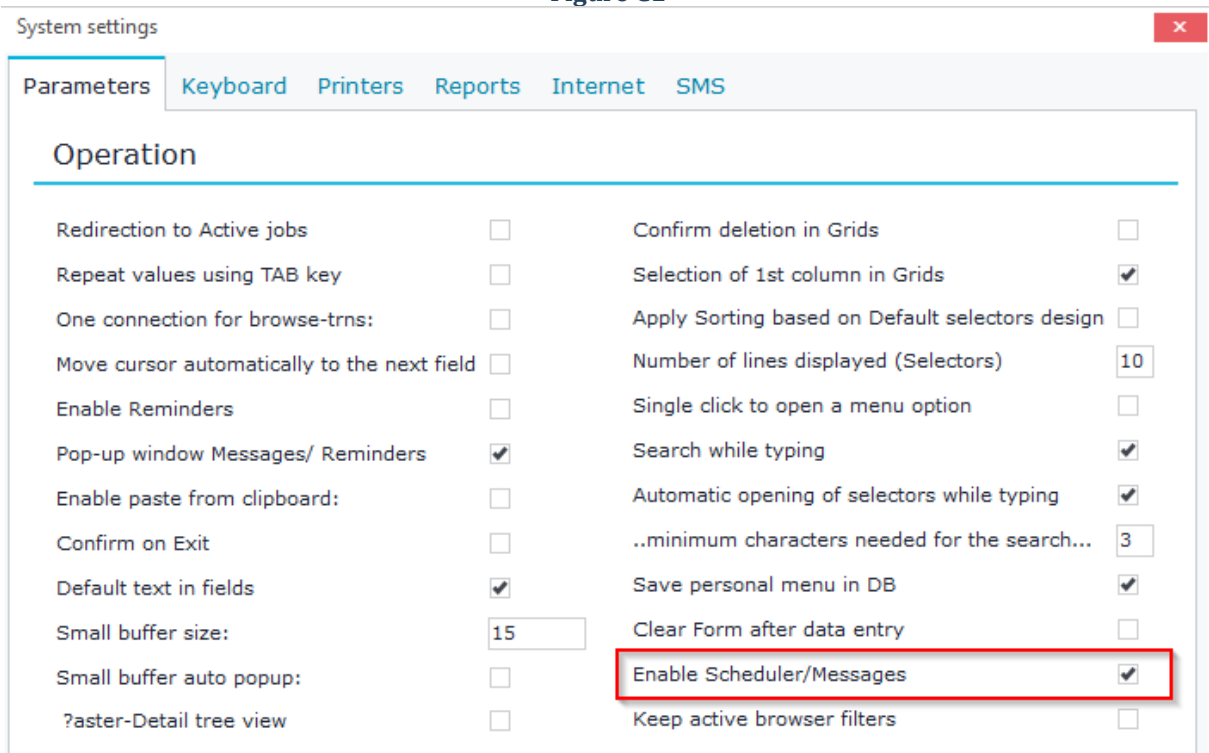
SoftOne Scheduler is a tool for scheduling SoftOne jobs (Browsers, Reports, etc.) which can run at a specific time and day from a specific user and repeat within specific time interval. It is located Tools-System-Scheduled Jobs menu (Figure C1). When you open the Scheduler tool then a browser is displayed with all the scheduled tasks. Scheduler messages are activated from the "System Settings" option "Enable Scheduler / Messages" (Figure C2).



The screenshot shows the 'Time scheduling' window with a toolbar containing icons for List, Filter, Sort, Star, New, Save, Delete, and navigation arrows. Below the toolbar is a table titled 'List' with the following columns: Inserti..., Job, Next run, Inserted by, and Processed. The table contains 18 rows of scheduled tasks, mostly with dates from 2018 and order codes. The 'Processed' column has checkboxes, some of which are checked.

Inserti...	Job	Next run	Inserted by	Processed
18/01/2018	ALERT:Your Quote Is Ready- Order Code: OFFR000009		demo user	<input type="checkbox"/>
18/01/2018	ALERT:Your Quote Is Ready- Order Code: OFFR000009		demo user	<input type="checkbox"/>
07/02/2018	ALERT:Your Quote Is Ready- Order Code: OFFR000010		demo user	<input type="checkbox"/>
07/02/2018	ALERT:Your Quote Is Ready- Order Code: OFFR000010		demo user	<input type="checkbox"/>
06/04/2018	ALERT:Your Order Is Ready Orde Code: ORDW000006		Administrator	<input checked="" type="checkbox"/>
06/04/2018	ALERT:Your Order Is Ready Orde Code: ORDW000006		Administrator	<input checked="" type="checkbox"/>
06/04/2018	ALERT:Your Order Is Ready- Order Code: ORDW000007		Administrator	<input checked="" type="checkbox"/>
06/04/2018	ALERT:Your Order Is Ready- Order Code: ORDW000007		Administrator	<input checked="" type="checkbox"/>
10/04/2018	ALERT:Your Order Is Ready- Order Code: ORDW000008		Administrator	<input type="checkbox"/>
10/04/2018	ALERT:Your Order Is Ready- Order Code: ORDW000008		Administrator	<input type="checkbox"/>
10/04/2018	ALERT:Your Order Is Ready- Order Code: ORDW000009		Administrator	<input type="checkbox"/>
10/04/2018	ALERT:Your Order Is Ready- Order Code: ORDW000009		Administrator	<input type="checkbox"/>
10/04/2018	ALERT:Your Order Is Ready- Order Code: ORDW000010		Administrator	<input type="checkbox"/>
10/04/2018	ALERT:Your Order Is Ready- Order Code: ORDW000010		Administrator	<input type="checkbox"/>
10/04/2018	ALERT:Your Order Is Ready- Order Code: ORDW000011		Administrator	<input type="checkbox"/>
10/04/2018	ALERT:Your Order Is Ready- Order Code: ORDW000011		Administrator	<input type="checkbox"/>
10/04/2018	ALERT:Your Order Is Ready- Order Code: ORDW000012		Administrator	<input type="checkbox"/>

Figure C1



The screenshot shows the 'System settings' window with the 'Parameters' tab selected. Under the 'Operation' section, there are two columns of settings. The 'Enable Scheduler/Messages' option is highlighted with a red box and has a checked checkbox.

Operation	
Redirection to Active jobs	<input type="checkbox"/>
Repeat values using TAB key	<input type="checkbox"/>
One connection for browse-trns:	<input type="checkbox"/>
Move cursor automatically to the next field	<input type="checkbox"/>
Enable Reminders	<input type="checkbox"/>
Pop-up window Messages/ Reminders	<input checked="" type="checkbox"/>
Enable paste from clipboard:	<input type="checkbox"/>
Confirm on Exit	<input type="checkbox"/>
Default text in fields	<input checked="" type="checkbox"/>
Small buffer size:	15
Small buffer auto popup:	<input type="checkbox"/>
?aster-Detail tree view	<input type="checkbox"/>
Confirm deletion in Grids	<input type="checkbox"/>
Selection of 1st column in Grids	<input checked="" type="checkbox"/>
Apply Sorting based on Default selectors design	<input type="checkbox"/>
Number of lines displayed (Selectors)	10
Single click to open a menu option	<input type="checkbox"/>
Search while typing	<input checked="" type="checkbox"/>
Automatic opening of selectors while typing	<input checked="" type="checkbox"/>
..minimum characters needed for the search...	3
Save personal menu in DB	<input checked="" type="checkbox"/>
Clear Form after data entry	<input type="checkbox"/>
Enable Scheduler/Messages	<input checked="" type="checkbox"/>
Keep active browser filters	<input type="checkbox"/>

Figure C2

C.1 CreateTask

While in Scheduler browser, click on the button "New" to create a new scheduler task (Figure C1.1).

Available options are the following:

Insert date: Inserted automatically using the insert date of the task (Login Date).

Inserted by: Inserted automatically using the insert user of the task (Login User).

Executed by: Sets the user that will run the task.

Recurrency: Recurrency of the task (Figure C1.2).

Restart(Minutes): Time for restarting the job.

Auto run: "Yes" → Displays the task to the user and runs it automatically, "No" → Displays the task and waits for user reaction.

Date of Last Run: Displays the last execution date.

Next Run: Displays the next execution date.

Processed: "Yes" → If the task has been executed, "No" → If the task is not yet executed, or if the task is scheduled to run at regular intervals.

Select report/printout: Select from the available browsers, design reports and open reports (Figure C1.3).

Select Job: Select from the available jobs of the application (Figure C1.4).

Figure C1.1 shows an example of scheduling the batch job "Batch modification of sales prices".

Note: When inserting a new task some properties are auto filled from SoftOne. You may delete any property you do not need. For example, if you wish to run an object but you don't want to use a certain template (photo) then delete the line PHOTO=.

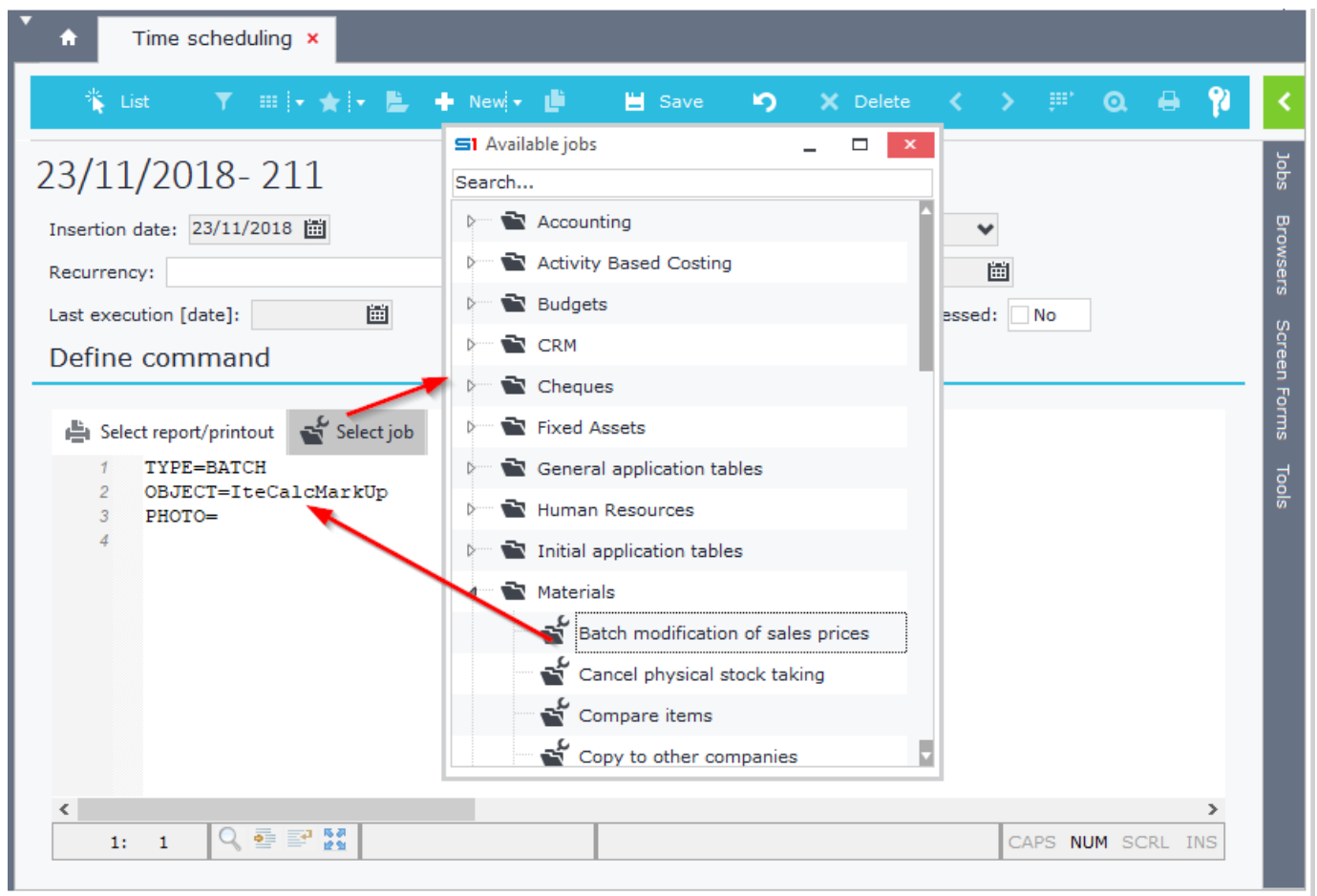


Figure C1.1

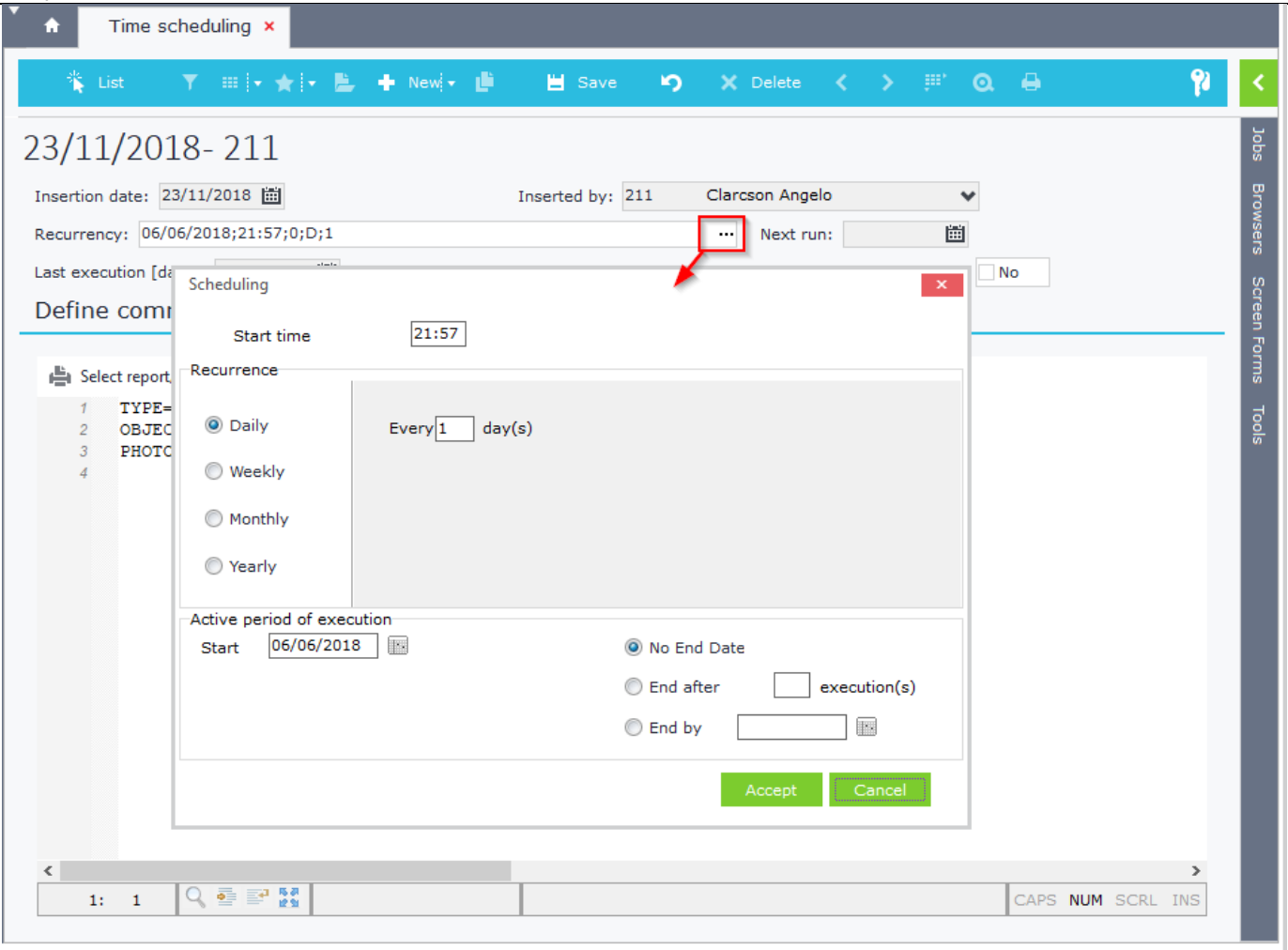


Figure C1.2

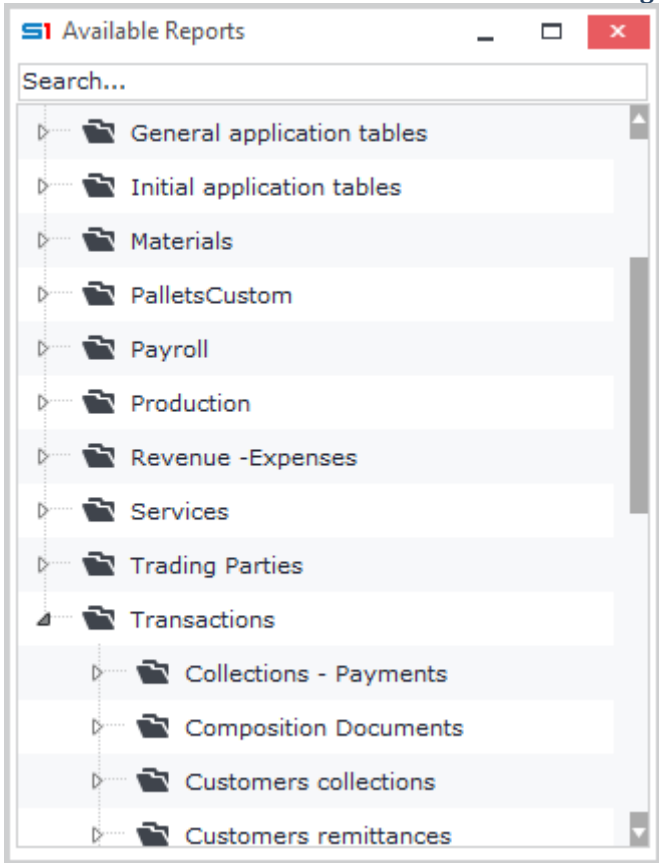


Figure C1.3

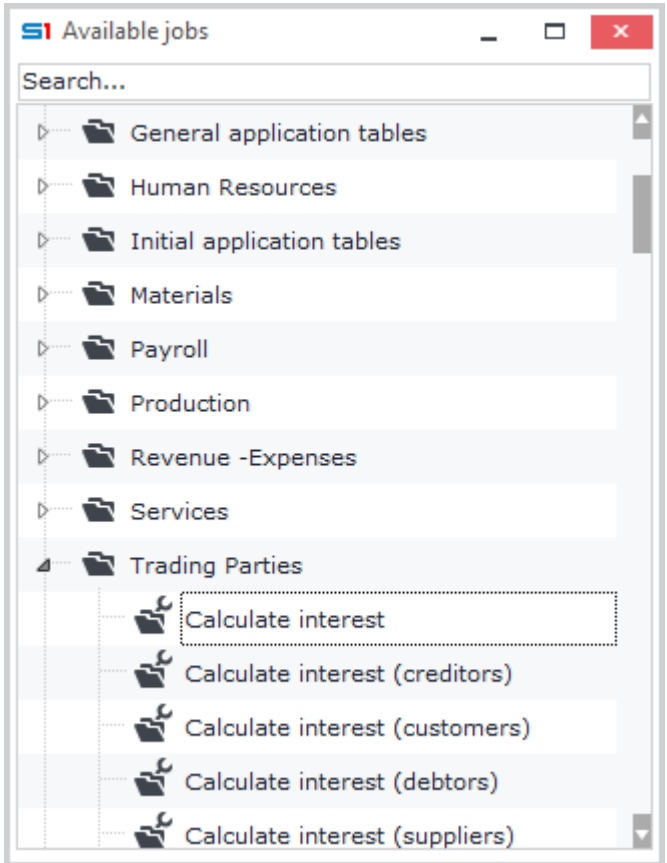


Figure C1.4

C.2 Scheduler Commands

You can use extra commands when creating a task in Scheduler. All the available commands are displayed in the table below, which are similar to the Remote Server commands.

Scheduler Commands		
Command	Parameters	Operation
TYPE	REPORT, BATCH, DIALOG, DESIGN, ANSWER, REPORTLIST, BATCHLIST, PRINTERLIST, COMPANYLIST, BRANCHLIST, BAM	Job type (Figure C2.1)
OBJECT		Object selection (i.e. CUSTOMER)
JOBNAME		Name of Browser or designed Report
PHOTO		Template name
AUTOEXECUTE	0,1 (e.g. AUTOEXECUTE=1)	Auto run
OUTPUT	928, 437, EXCEL, WORD, METAFILE, PDF File, PrinterName	Sends Job Results to a specific File or printer
FILENAME		Name of the File for saving the results
SENDTO	MAIL, GSM	Send Results to Email or Cell phone
MAILADDR		Email Address
GSMNUM		Cell phone Number

Time scheduling

18/01/2018- 500

Insertion date: 18/01/2018 Inserted by: 500 demo user

Recurrency: ... Next run: ...

Last execution [date]: Processed: ☐ No

Define command

Select report/printout Select job Fields ? Help

```

1 TYPE=ANSWER
2 SENDTO=GSM:123456789
3 ANSWERSTR='Your Quote Is Ready- Order Code: OFFR000009'

```

2: 21 CAPS NUM SCRL INS

Figure C2.1

D. Messages

SoftOne messages are used for text communication between application users and they are activated (for every pc) through the “System settings” option “Enable Scheduler/Messages” (Figure D1).

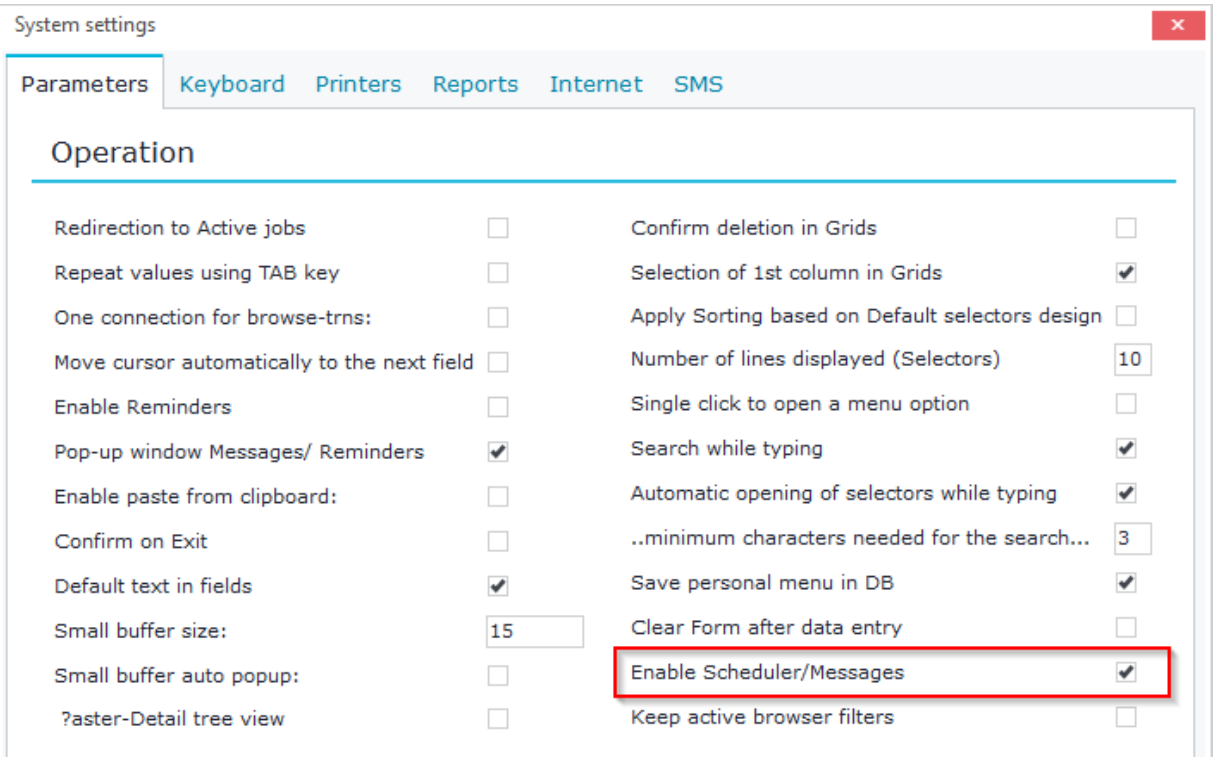



Figure D1

New messages are added from the “Messages” icon  that sits in the application toolbar (Figure D2). This opens a window that shows all pending messages. Use the button “New” to add a new message. Then select the “Users” or “User Groups” that will receive the message and enter the text message in the memo box.

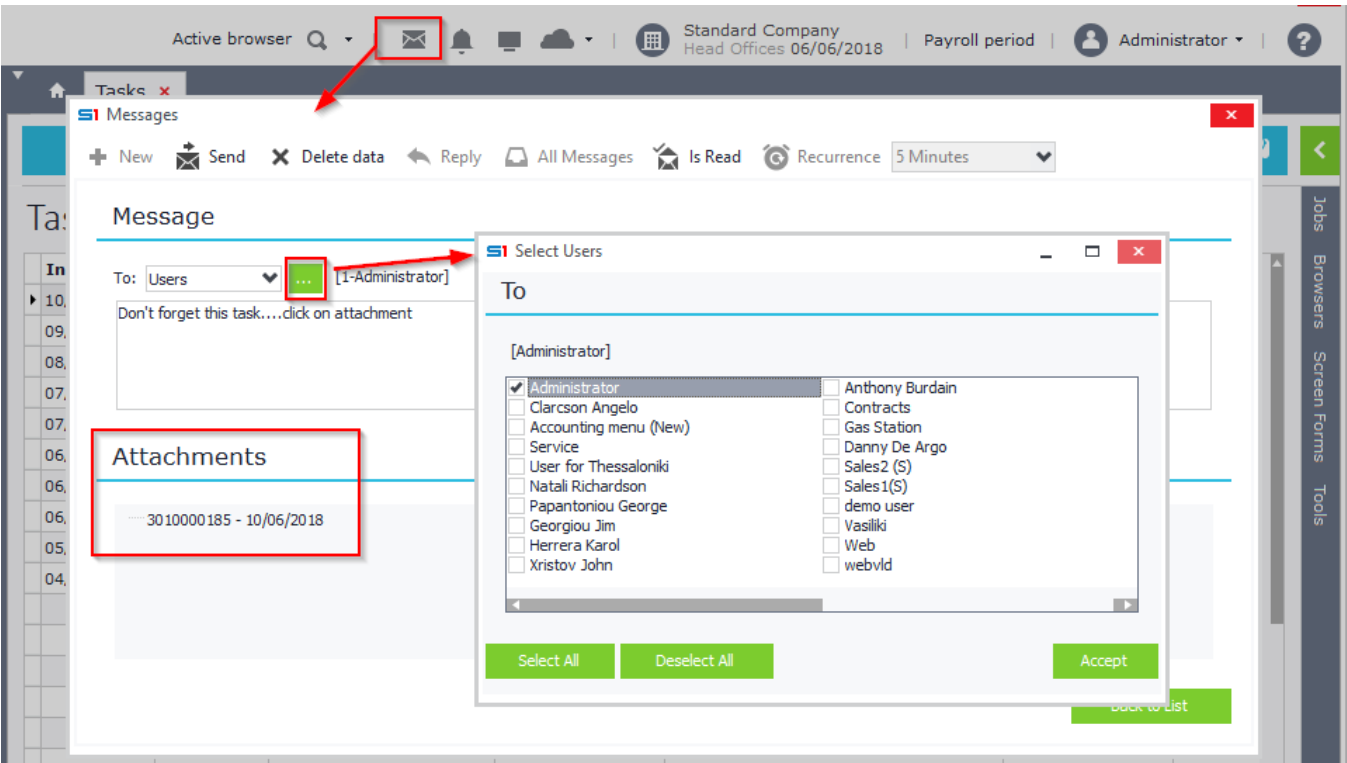


Figure D2

There is also an option of adding links to entity records that will display the records when clicked from recipients. This is very useful when you need to refer to a record, because it will be easily accessed from users that receive the message. The only thing you need to do before sending the message is right click on the preferred record (from the object browser) and select "Attach to message" (Figure D3). This option is available in all the objects of the application. Then, when you create a new message, the record will be auto inserted as attachment (Figure D2).

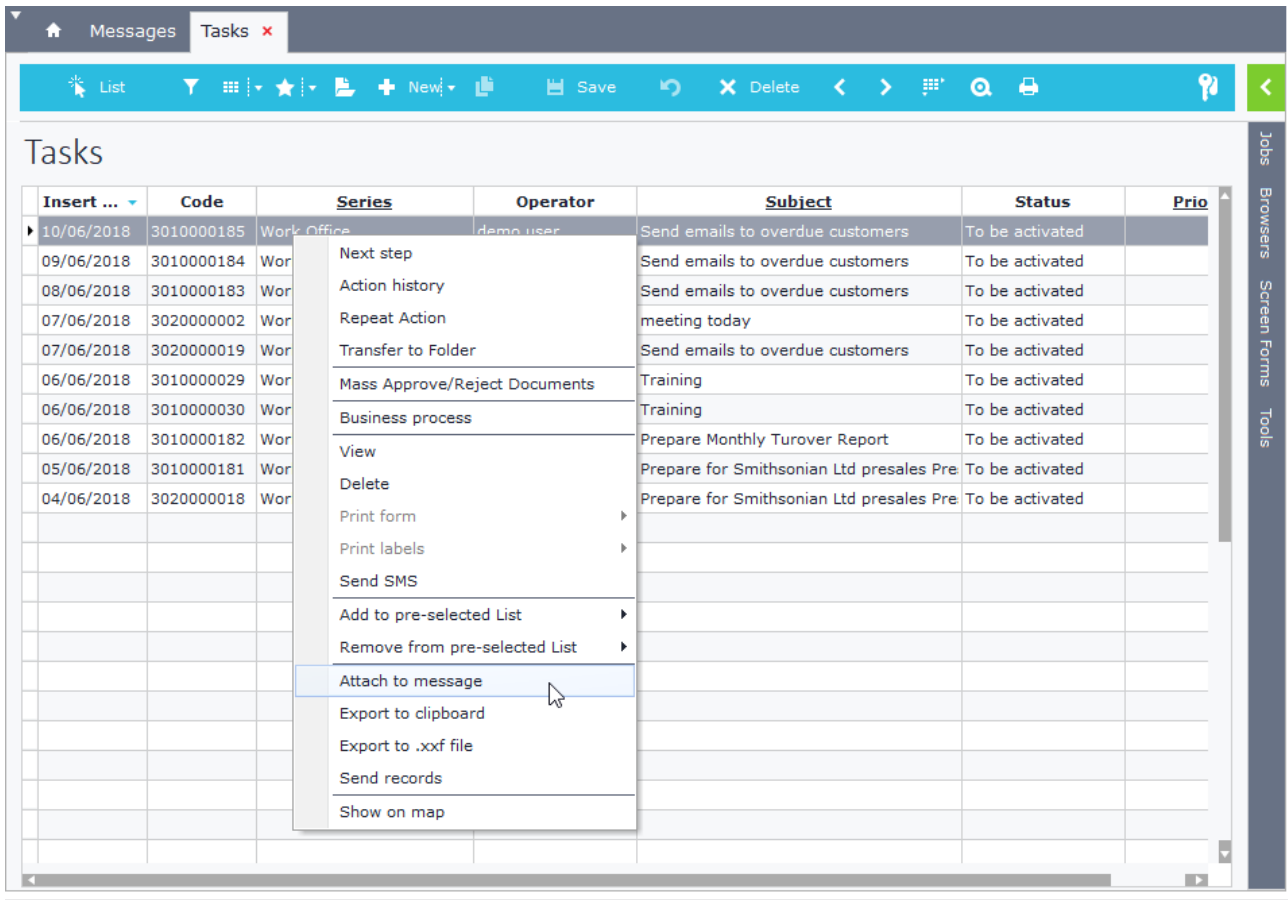


Figure D3

There is also another option of adding a message, and that is through the menu job "Company processes – Business processes – Messages". This opens the window of Figure D4 where you have also the option of recurrency (Time scheduling).

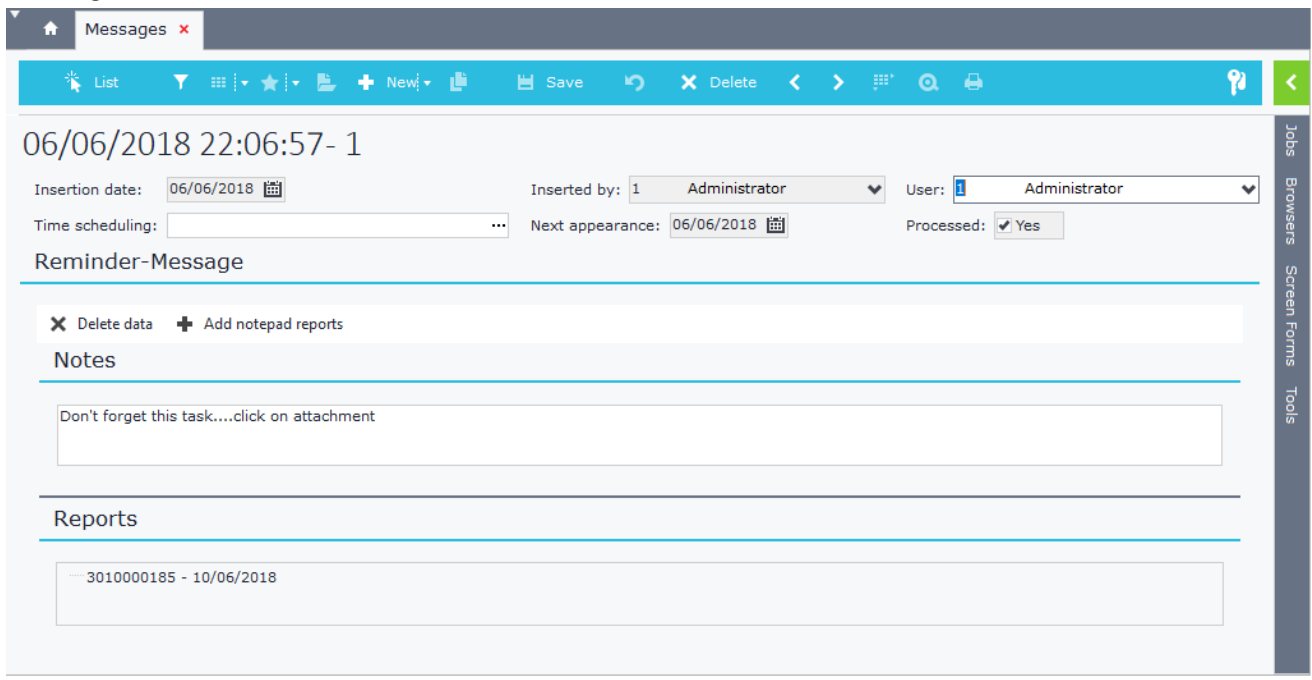



Figure D4

Pending messages are displayed through the application “Messages” icon  (Figure D5). On double click of a record, the message is displayed on a different window, along with its attachments, that allow to be redirected to linked records (if any) (Figure D6).

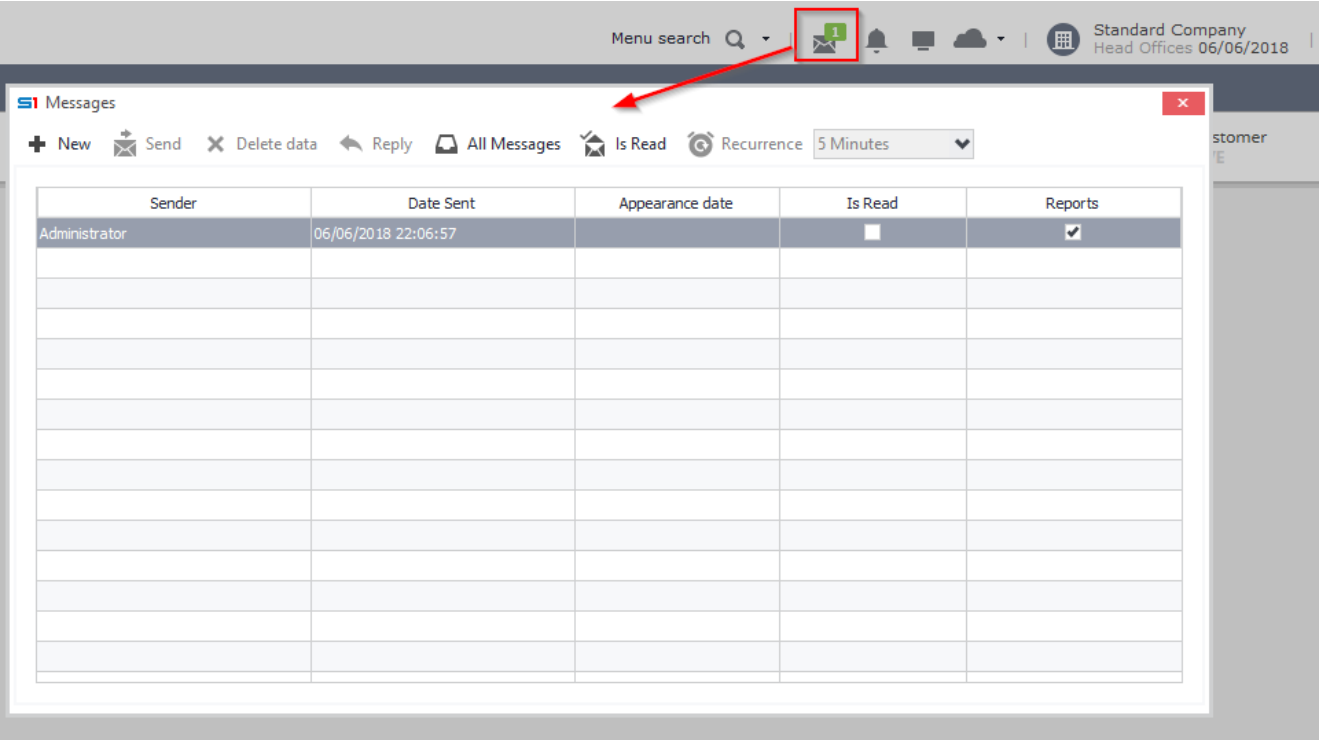


Figure D5

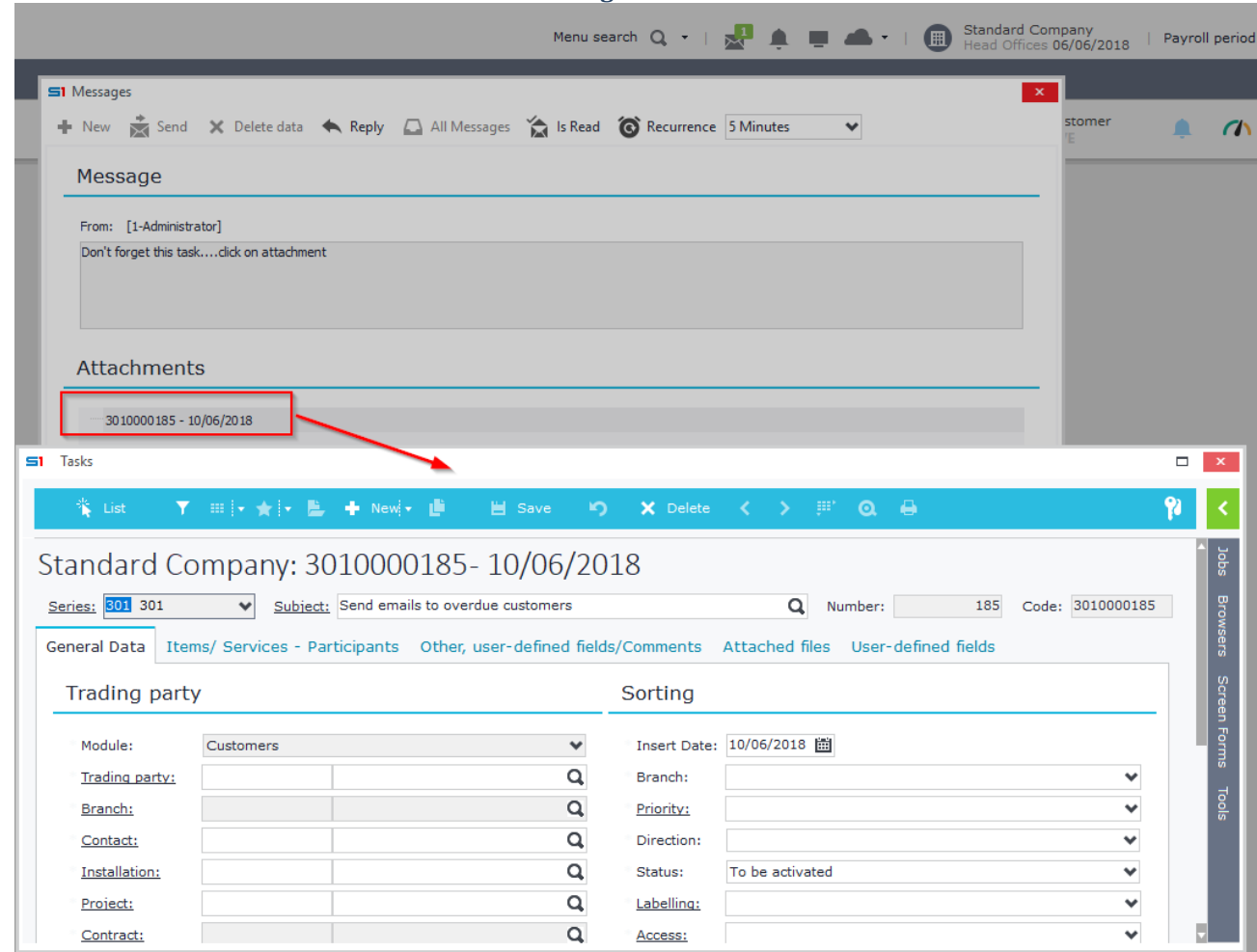


Figure D6

9. FORM SCRIPTS

A. [General Features](#)

B. [Object Methods](#)

C. [Object Functions](#)

D. [Dataset Methods](#)

E. [Dataset Functions](#)

F. [Field Events](#)

G. [Dataset Events](#)

H. [Object Events](#)

I. [Sub Form Events](#)

J. [Report Objects Events](#)

K. [Advanced JavaScript](#)

[Case Studies](#)

A. General Features

Form Script Engine is a very powerful tool that allows you to override SoftOne Objects functionality, execute custom jobs through commands or events, post data to different objects, run external applications, calculate data through different database objects, run SoftOne SBSL Scripts, make web service calls and many others that are discussed in the following sections.

Form Scripts affect only the form that they are written, where [Advanced JavaScript](#) affects all the forms of an object. The code of a script inside an object form is entered in Design mode, inside the "Script" tab (Figure A1).

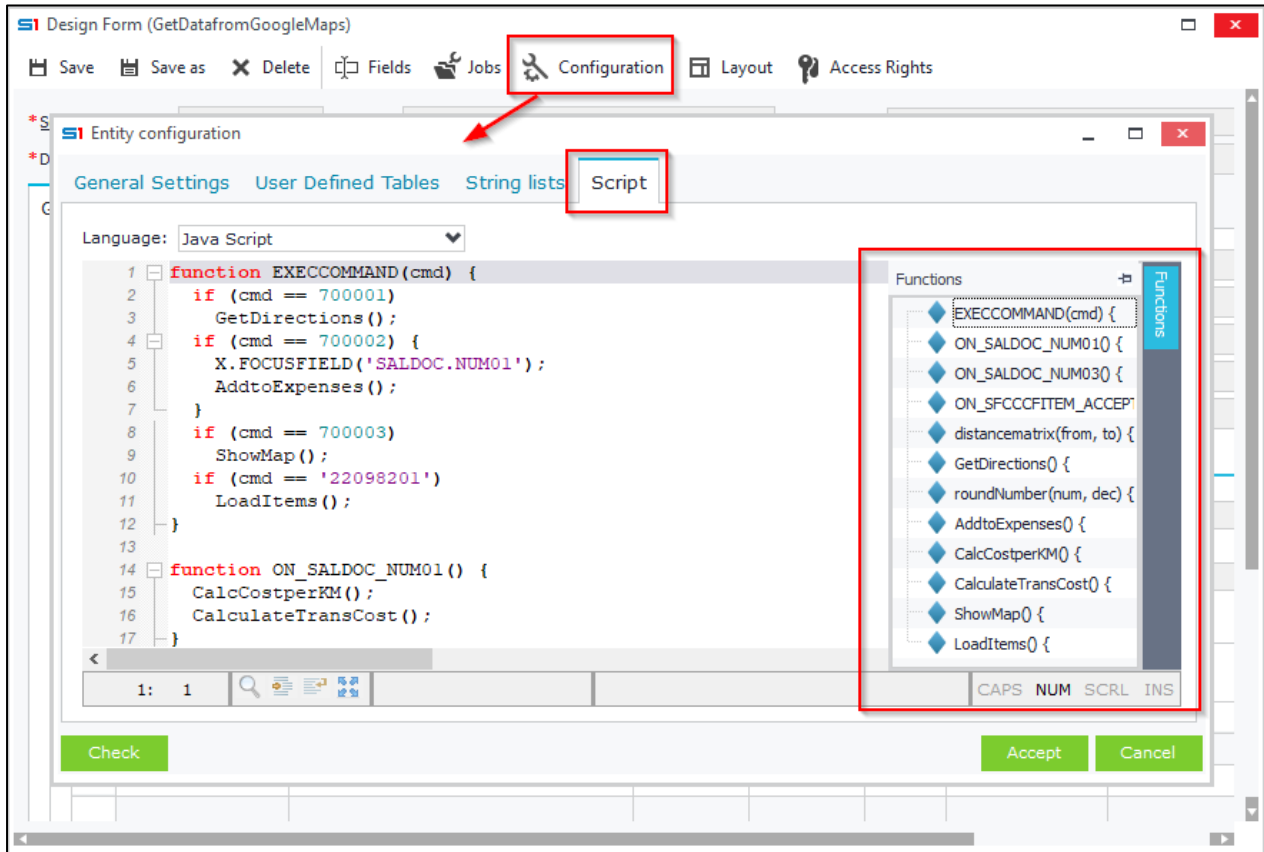


Figure A1 – Script Editor – Functions

A.1 Script Editor

All the script functions are displayed and can be easily accessed from the right-hand side of the script window. Use **F12** to display the definition of the selected function in code. The button "**Check**" performs a **syntax** check of the JavaScript code and displays the results in a message (Figure A1.1).

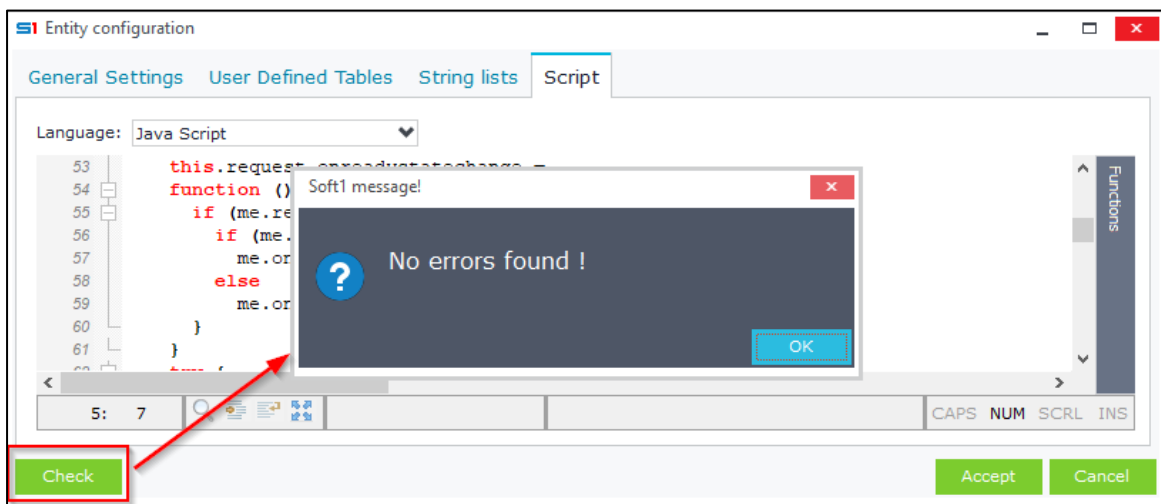


Figure A1.1 – Script Editor – Check Syntax

A list of SoftOne / JavaScript functions & methods is displayed when **CTRL+Spacebar** is pressed, depending on the symbol that the cursor points inside the script code:

- After **X.** → Displays all the available object methods & functions (Figure A1.2).
- After **TableName.** → Displays all the fields of the table and all the SoftOne functions & methods that can be used (Figure A1.3). It also displays fields from FindTable variables (after CreateObj is used).
- On empty space → Displays all the available SoftOne Events and JavaScript functions (Figure A1.4).
- After JavaScript function → Displays JavaScript methods & properties (Figure A1.5).

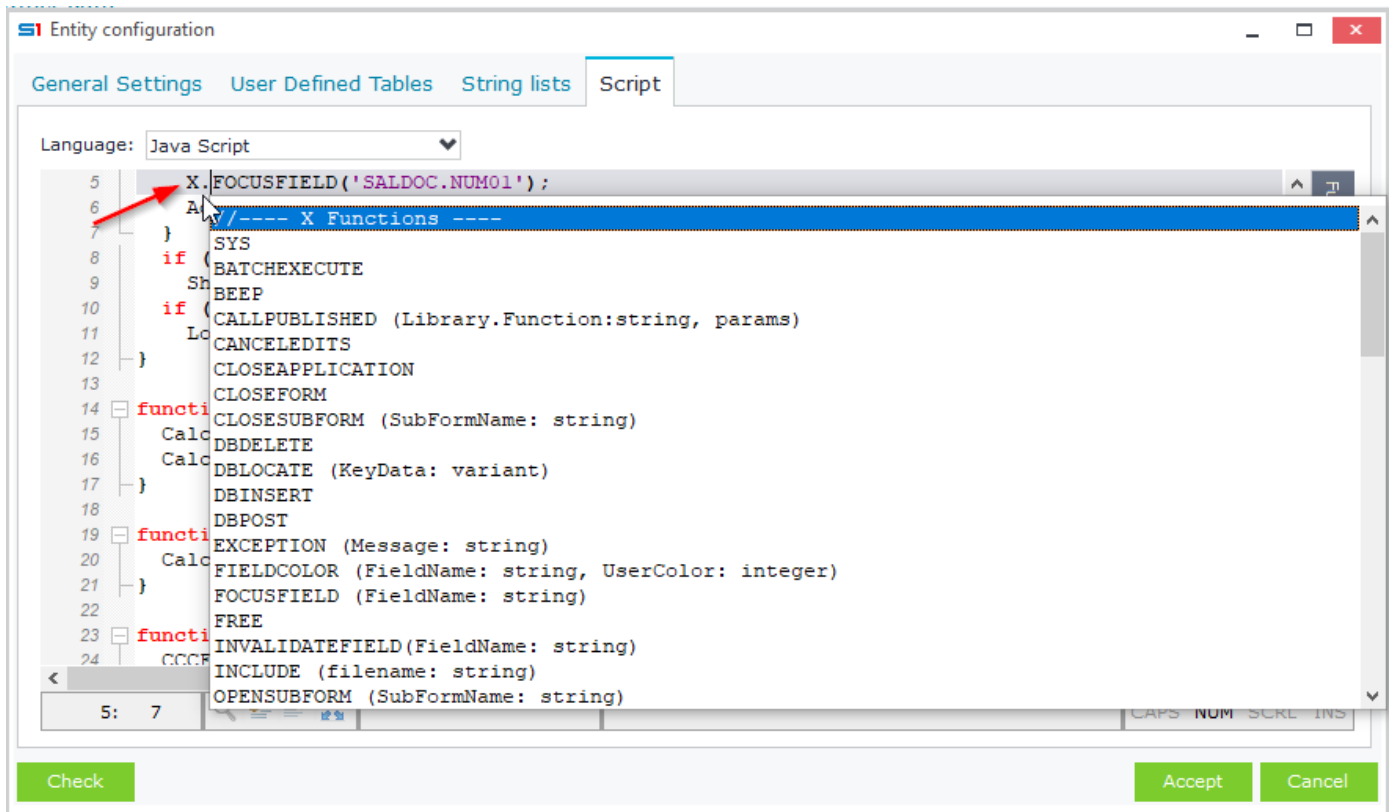


Figure A1.2 – Script Editor – CTRL+Spacebar on X.

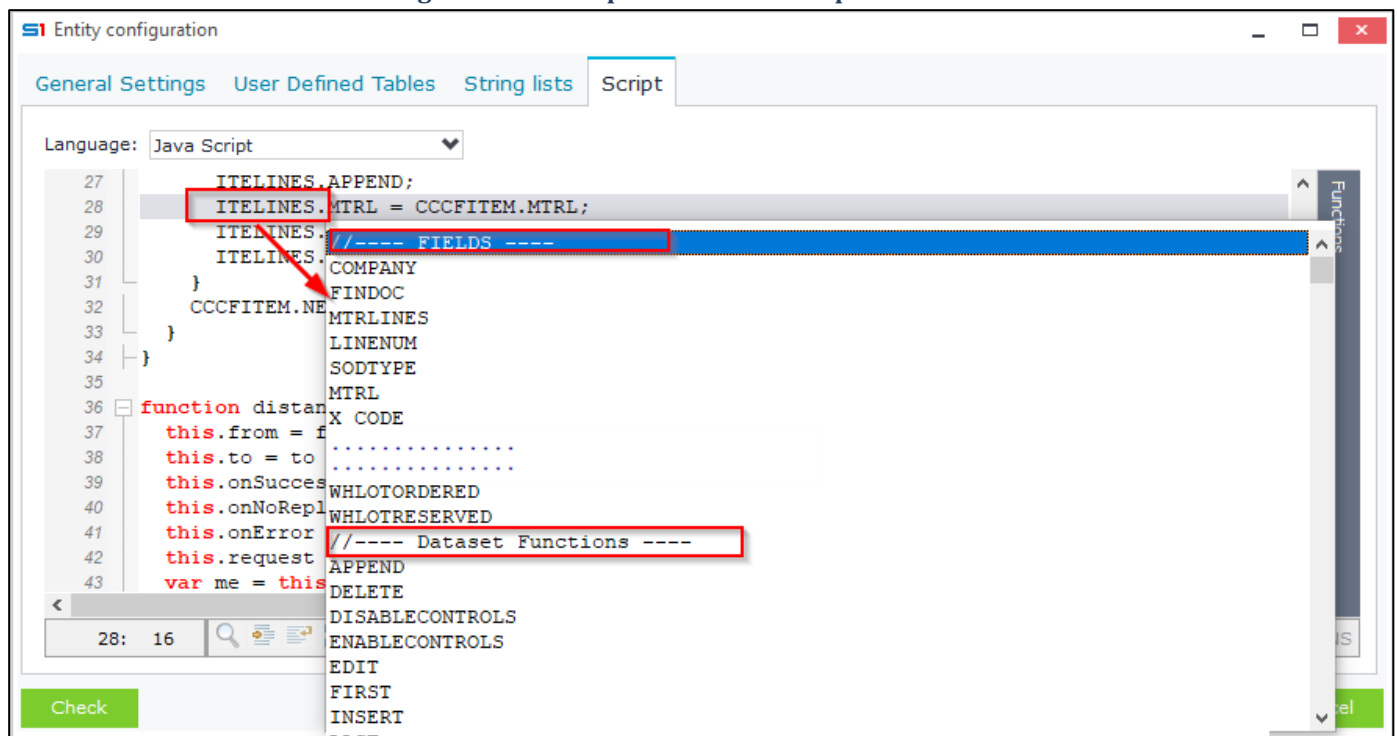


Figure A1.3 – Script Editor – CTRL+Spacebar on TableName.

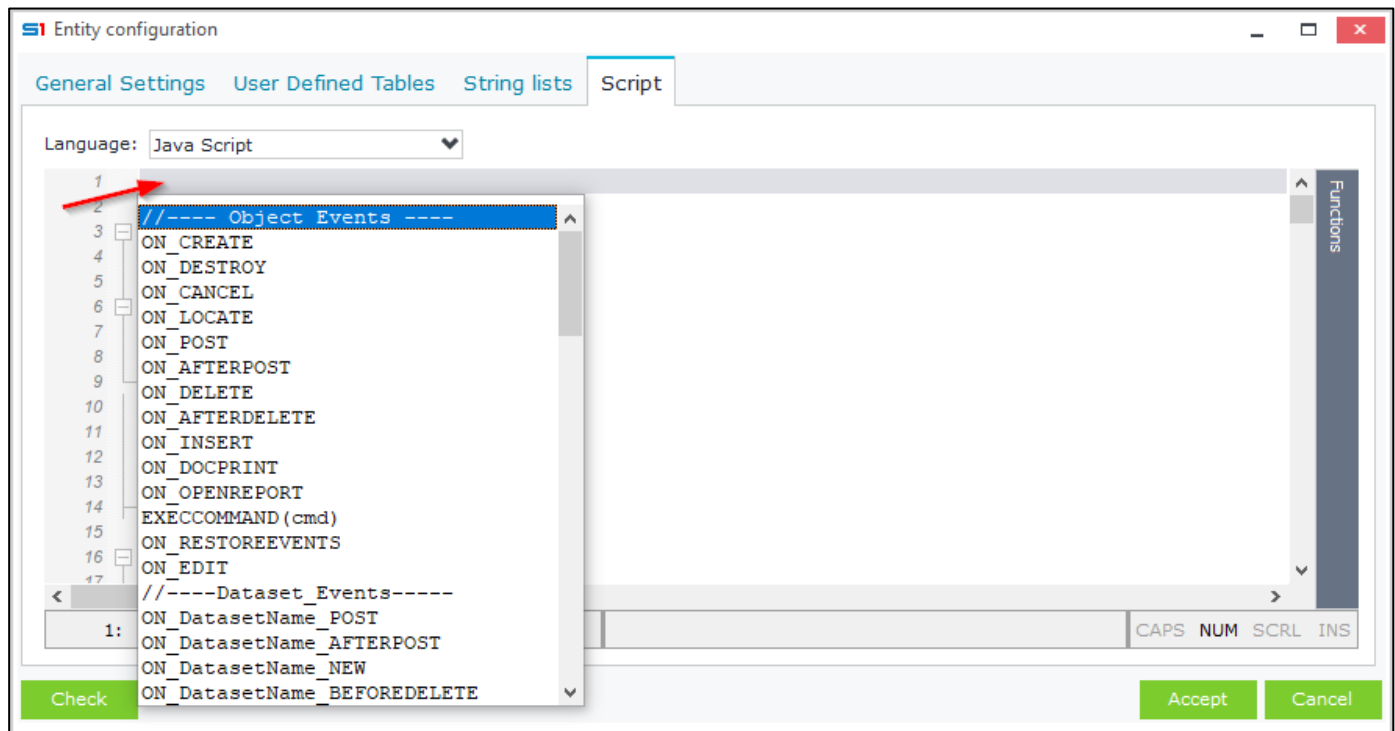


Figure A1.4 – Script Editor – CTRL+Spacebar on empty space.

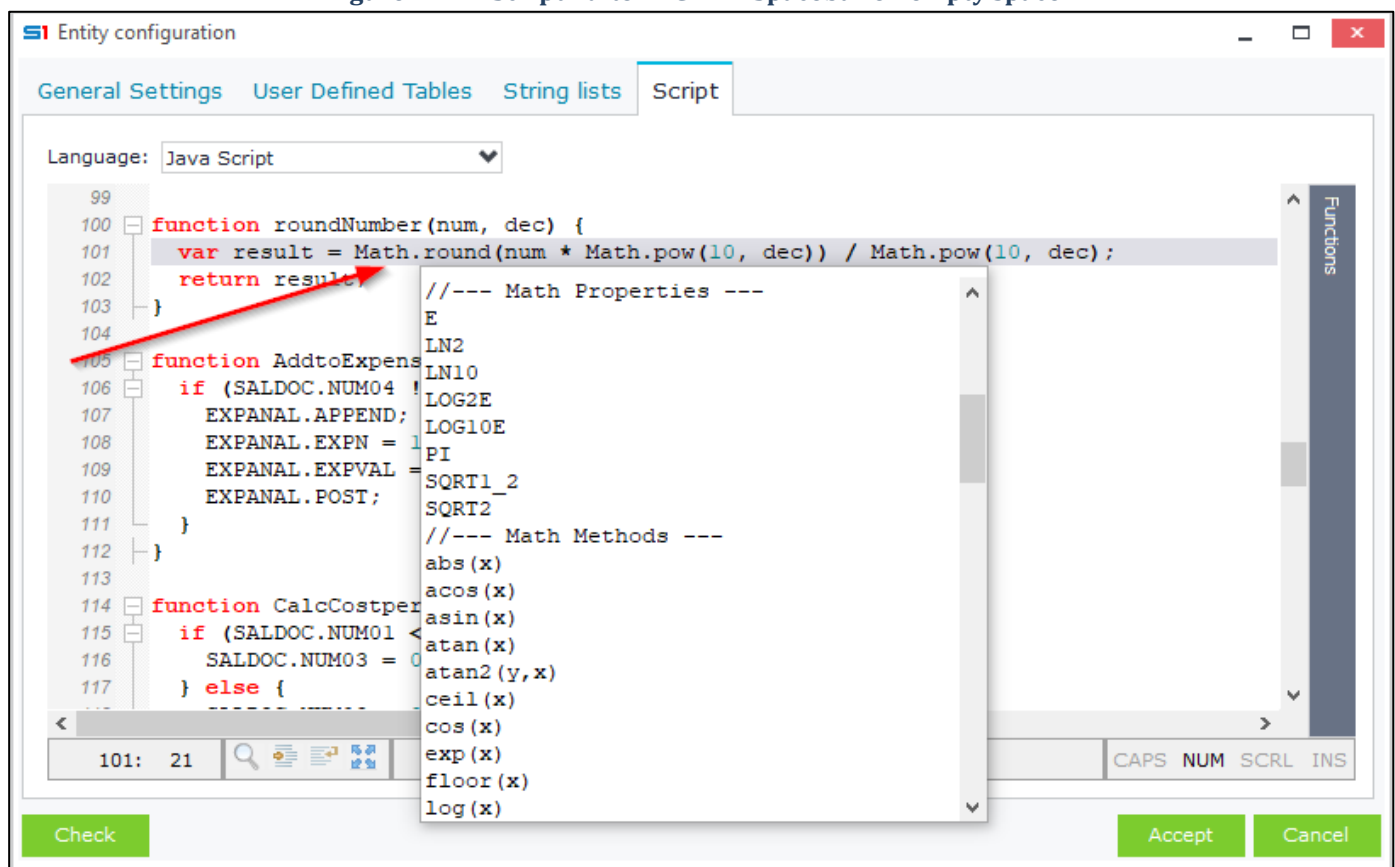


Figure A1.5 – Script Editor – CTRL+Spacebar on Javascript function.

A.2 Debug JavaScript

SoftOne JavaScript code can be debugged using **Just-In-Time Debugger** through Microsoft Script Editor or Visual Studio. Enable /Disable Just-In-Time debugger using the following instructions:

- Open Registry Editor (regedit).
- Add a new REG_DWORD value inside the registry path (Figure A2.1):
"HKEY_CURRENT_USER\Software\Microsoft\Windows Script\Settings"
- Name the DWORD as "JITDebug". Value data **1** enables the feature, while **0** disables it (Figure A2.2).
- Once the above are completed you can add the word "**debugger;**" at any point of a form script (or Advanced JavaScript). This word functions as **breakpoint** and prompts to launch an installed editor for debugging.

Note: When this feature is enabled all the SoftOne Exception messages will be prompted for debug.

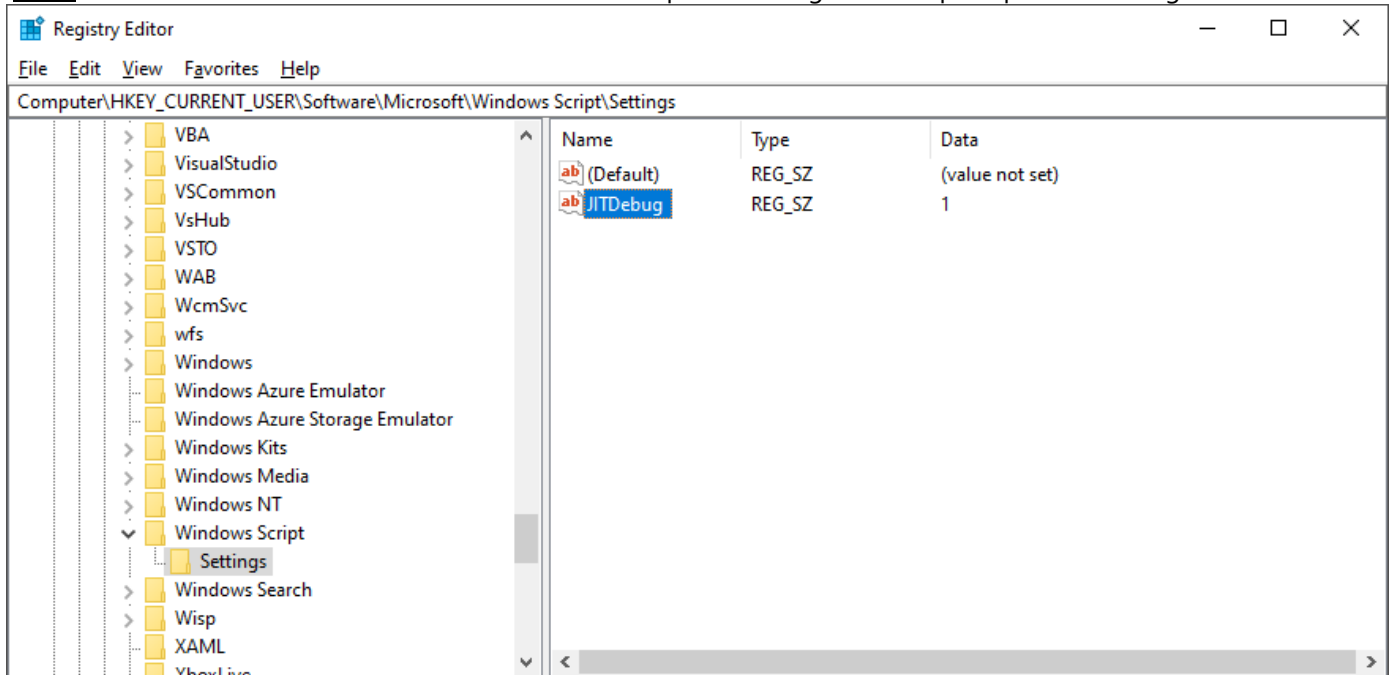


Figure A2.1 – Registry Editor – Debug Javascript

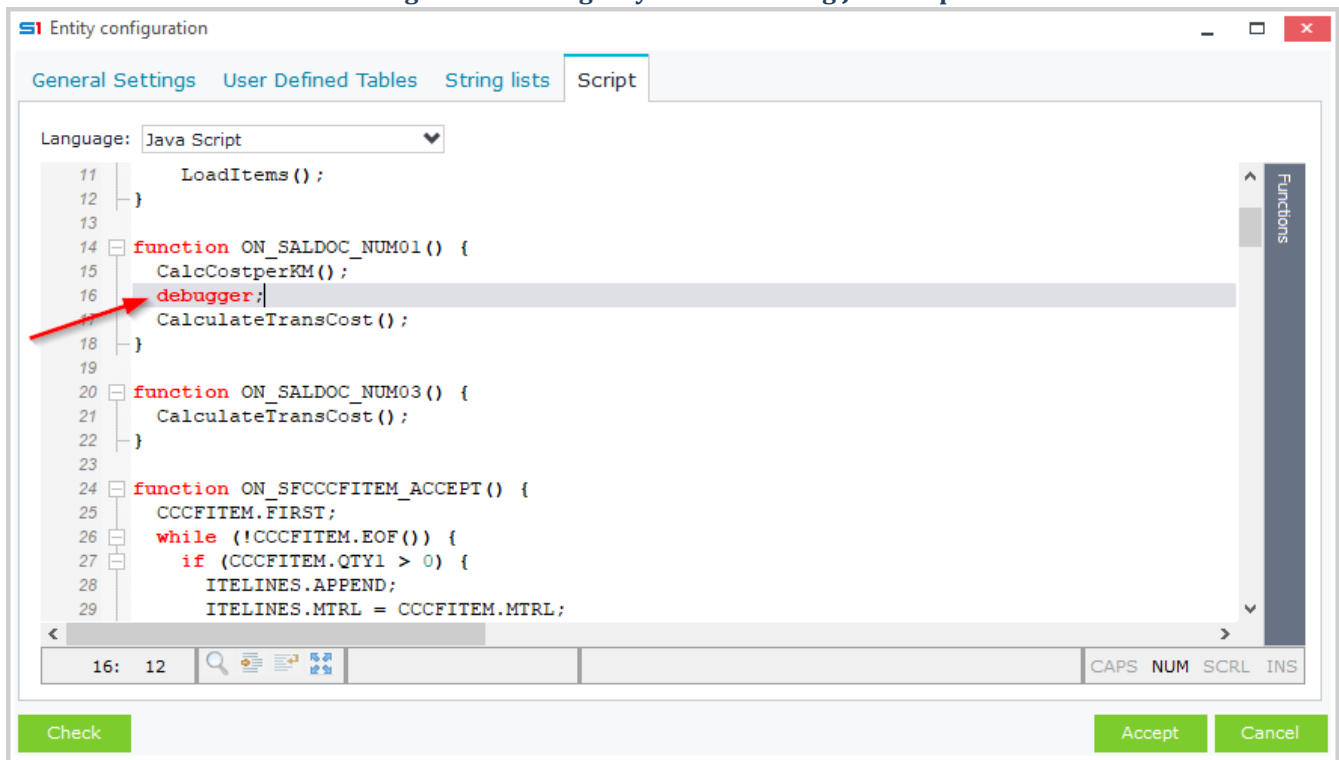


Figure A2.2 – Debug Javascript using debugger;

A.3 Global Variables

SoftOne script engine allows the usage of global variables, that can be accessed from any object or custom script. Global variables hold their value and are accessible during application execution.

The syntax of global variables is:

XGlobal.VariableName

Example 1

Declare a global variable named vCustomer that stores the id of the located customer record (on Locate event of Customers object).

```
function ON_LOCATE() {  
    XGlobal.vCustomer = CUSTOMER.TRDR;  
}
```

This variable can be used in any other object (e.g. after the selection of a customer in Sales Documents). It is preferred to check if the variable is null.

```
function ON_SALDOC_TRDR() {  
    if ( (XGlobal.vCustomer != null) && (SALDOC.TRDR > 0)  
        && (SALDOC.TRDR == XGlobal.vCustomer) ) {  
        //Add Code Here  
    }  
}
```

Global variables can be redefined in any script using the same way.

Example 2

Create a button inside Installations module, that copies the data of the located record (source) in a new record (target) that is displayed on the screen. Add also, the data of the source field INST.INST inside a custom field (INST.CCCRELINST) of the target record.

When users locate a record of Installations, a global variable (vCurrentInst) stores the data of the located INST id. Then, when the button (20200401) is clicked we perform a copy of the selected record and finally, we assign the field INST.CCCRELINST with the value of the global variable vCurrentInst that was previously stored.

```
function ON_LOCATE() {  
    XGlobal.vCurrentInst = INST.INST;  
}  
  
function EXECCOMMAND(cmd) {  
    if (cmd == 20200401) {  
        X.EXEC('BUTTON:NEW');  
        X.EXEC('BUTTON:COPY');  
        INST.CCCRELINST = XGlobal.vCurrentInst;  
    }  
}
```

A.3 S1P & X.SYS

The [built-in functions](#) of the internal SoftOne library can be easily used at any point of a script through the command **S1P.functionName**.

```
var vEasterDate = S1P.GetPasxaDate(S1P.LoginDate);  
var vConvSeries = S1P.GetConvertSeries(1351, 7001)//returns 7021;7721;7041
```

[X.SYS parameters](#) are populated on login and are available inside scripts using **X.SYS.ParamName**.

```
var vLoginUser = X.SYS.USER;
```

String variables that use quotes can also access X.SYS params with a preceding colon, as in the following example.

```
var msg = "Login Year is :X.SYS.FISCPRD"
```

Example

Form script inside Customers form that calculates data using S1P and displays a message, when users click on button "Financial Data". There is also another button that checks for valid Tr.No and displays the results.

```
function EXECCOMMAND(cmd) {  
    if (cmd == 202021) {  
        var vTurnover = S1P.CusNetTurnover(CUSTOMER.TRDR,  
S1P.FirstPeriodDate(S1P.LoginYear,1),S1P.EndOfYear(S1P.LoginDate));  
        var vBalance = S1P.CusBalance(CUSTOMER.TRDR,S1P.EndOfYear(S1P.LoginDate));  
        var vAvgBalDays = S1P.Trunc(S1P.CusAvgBalDays(CUSTOMER.TRDR,  
S1P.EndOfYear(S1P.LoginDate)));  
        var vPendCheques = S1P.CusChequeBalance(CUSTOMER.TRDR,1);  
        var vOpenOrders = S1P.CusOpenOrder(CUSTOMER.TRDR);  
        var strMessage = "";  
        strMessage = "Customer Financial Data\n"  
            + "\nTurnOver: " + S1P.ARound(vTurnover,0.01)  
            + "\nBalance: " + S1P.Round(vBalance,2) + " (" +  
S1P.Spell(S1P.Round(vBalance,2),1) + ") "  
            + "\nAverage Balance Days: " + vAvgBalDays  
            + "\nPending Cheques: " + vPendCheques  
            + "\nOpen Orders: " + vOpenOrders  
        X.WARNING(strMessage);  
    }  
    else if (cmd == 202022) {  
        var vValidTrNo = S1P.CusCheckAFM(CUSTOMER.AFM);  
        X.WARNING("Tr.No is " + (vValidTrNo == 1 ? "" : "not")+ " valid";  
    }  
}
```

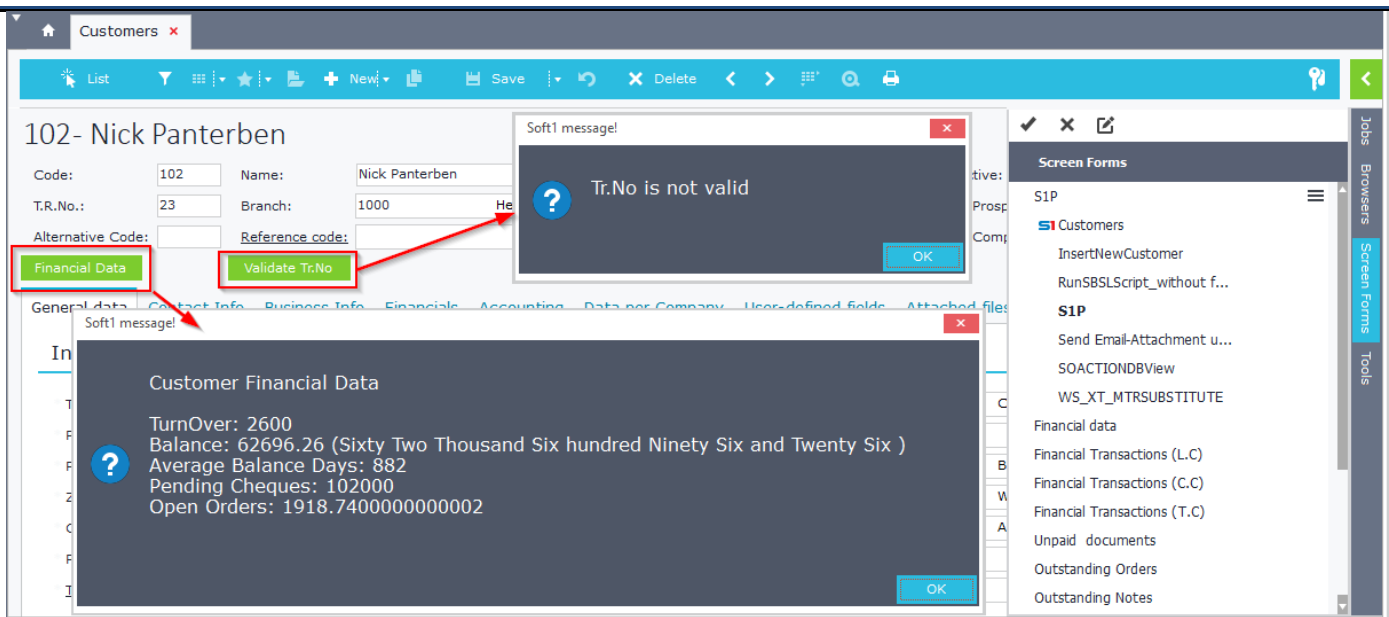


Figure A3.1 – S1P Calculations

B. Object Methods

This module demonstrates all the object methods you can use in form scripts through X.FunctionName.

BATCHEXECUTE

Runs the Batch Object

Examples

[See Case Study 4](#) - [Case Study 10](#)

BEEP

Plays a simple beep sound

Example

```
X.BEEP;
```

CALLPUBLISHED (Library.Function:string, params)

Runs the SoftOne SBSL CallPublished function. Functions can be found in section [Libraries](#) of Chapter "[SoftOne Batch Script Language](#)" (SBSL). Params are comma separated.

Example 1

Send SMS using the WEB SMS Account defined in Settings and the function XSendWebSMS.

```
function SendSMS () {  
    X.CALLPUBLISHED('SysRequest.XSendWebSMS', '0123456789', 'SMS Message');  
}
```

Example 2

Case Study 12 – Auto apply ABC model on post of a record (Supplier Other Transactions).

```
//sCommand = S1Tablename+';'+dim1+';'+dim2+';'+dim3+';'+dim4+';'+dim5+'  
;'+dim6+';'+dim7+';'+dim8+';'+dim9+';'+dim10;  
X.CALLPUBLISHED('ProgLibIntf.ModuleCommand', X.MODULE, 10120, sCommand)
```

CANCELEDITS

Cancels all changes made to data of an Object (Cancel entry).

Example

Prohibition - cancelation of changes for a specific user when saving an entry.

```
function ON_POST () {  
    if (X.SYS.USER == 901) //Checks if Login UserCode = 901  
        X.CANCELEDITS;  
}
```

CLOSEAPPLICATION

Closes the application.

Example

```
X.CLOSEAPPLICATION();
```

CLOSEFORM

Closes the active object form.

Example

```
X.CLOSEFORM();
```

CLOSESUBFORM (SubFormName: string)

Closes sub form name "SubformName".

Example

```
X.CLOSESUBFORM('MYSUBFORM');
```

COPY – PASTE

Copies all the data of a record (same as the toolbar button "Copy from last").

Example

```
function CopyCustomer(vTRDR) {
    XGlobal.vFromTrdr = vTRDR;

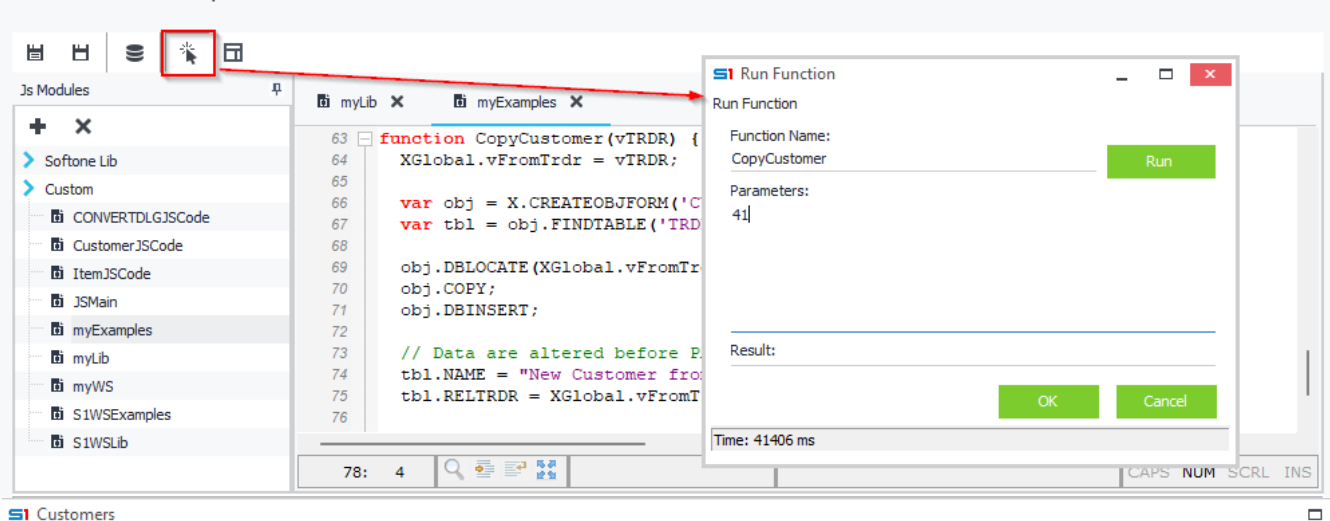
    var obj = X.CREATEOBJFORM('CUSTOMER');
    var tbl = obj.FINDTABLE('TRDR');

    obj.DBLOCATE(XGlobal.vFromTrdr);
    obj.COPY;
    obj.DBINSERT;

    // Data are altered before PASTE
    tbl.NAME = "New Customer from Copy";
    tbl.RELTRDR = XGlobal.vFromTrdr;

    obj.PASTE;
    obj.SHOWOBJFORM; //Display new record on screen
}
```

Advanced Javascript Editor



Customers

Save Cancel

New Entry

Customers

* Code		Name	New Customer from Copy	Active	<input checked="" type="checkbox"/> Yes	
TRN	0735	Branch	1000 Head Offices	is Prospect	<input type="checkbox"/> No	
Alternative Code		Reference code	101 Salesconsul GmbH	is Competitor	<input type="checkbox"/> No	

Add Photo

General data Contact Info Business Info Financials Accounting Data per Company User-defined fields Attached files/No

INVOICING DATA

Title ..

GROUPING/ CLASSIFICATION

Business unit 104 Clothing & Footwear

CREATEAPPLPOPUP(ModuleHandle, 0)

SBSL (SysRequest) function that can be used in form scripts to create button sub menus (string lists) and display them using the function **SHOWAPPLPOPUP**.

Syntax: X.EXEC('CODE:SysRequest.CreateApplPopup', X, 0);

Example

```
var mysubmenu = X.EXEC('CODE:ModuleIntf.FindXStrings', X.MODULE, 'MyButtonMenu');
var mypopup = X.EXEC('CODE:SysRequest.CreateApplPopup', X, 0);
X.EXEC('CODE:SysRequest.ShowApplPopup', mypopup, mysubmenu);
```

[See → Case Study 12](#)

DBDELETE

Deletes the located record of an Object, executing all SoftOne object methods and jobs (similar to Toolbar button "Delete"). If the object does not allow record deletion, an error is raised.

Example 1

Locate and delete a record of Purchase documents.

```
function DeleteRecord(vID) {
    var myObj = X.CreateObj('PURDOC');
    myObj.DBLocate(vID);
    myObj.DBDelete;
}
```

Example 2

[See → Case Study 2](#)

DBLOCATE (KeyData: variant)

Locates the record containing "KeyData" value in the primary key.

Example 1

Locate a record in Sales documents and display a message.

```
function LocateRecord(vID) {
    var myObj = X.CreateObj('SALDOC');
    myObj.DBLocate(vID);
    X.WARNING('Located record: ' + vID);
}
```

Example 2

[See → Case Study 2](#)

DBINSERT

Modifies the Object status to insert mode in order to accept new data (similar to Toolbar button "New").

Example 1

Function that adds a new customer from any SoftOne object and displays the new id in a warning message.

```
function AddNewCustomer() {
    var myObj = X.CREATEOBJ('CUSTOMER');
    var CustTbl;
    try {
        CustTbl = myObj.FINDTABLE('TRDR');
        myObj.DBINSERT;
        CustTbl.CODE = "000*";
        CustTbl.NAME = "TEST_CUSTOMER";
        var newid = myObj.DBPOST;
        X.WARNING("New Customer's ID is: "+newid);
    }
    catch (e) {
        if (myObj != null)
            X.WARNING("General Error: " + e.message + "\nObject Error: "+myObj.GETLASTERROR);
        else
            X.WARNING("General Error: " + e.message);
    }
}
```

Example 2

[See → Case Study 4](#)

DBPOST

Posts the data of an Object in the database, applying all SoftOne methods (similar to Toolbar button "Save").

Example

Function that locates a PRJC (Projects object) record, updates specific fields and posts the data.

```
function UpdatePRJC(vPRJCID, vnewTRDR, vnewBOOL01) {

    var myObj = X.CREATEOBJ('PRJC;MYCUSTOMFORM');
    var tblPRJC = myObj.FINDTABLE('PRJC');
    var tblPRJEXTRA = myObj.FINDTABLE('PRJEXTRA');

    try {
        myObj.DBLOCATE(vPRJCID);

        tblPRJC.TRDR = vnewTRDR;
        tblPRJEXTRA.BOOL01 = vnewBOOL01;

        myObj.DBPOST();
    }
    catch (e) {
        if (myObj != null)
            X.WARNING("General Error: " + e.message + "\nObject Error: "+myObj.GETLASTERROR);
        else
            X.WARNING("General Error" + e.message);
    }
}
```

EXCEPTION (Message: string)

Displays an exception message. Cancels all jobs and stops running code.

Example

Prevent User Groups 101, 102 from saving a record.

```
function ON_POST() {
    if ( (X.SYS.GROUPS == 101) || (X.SYS.GROUPS == 102) )
        X.EXCEPTION('No changes allowed. Save is cancelled!!!');
}
```

FIELDCOLOR (FieldName: string; UserColor: integer)

Sets the back color of a control. **UserColor = (1 * Red) + (256 * Green) + (65536 * Blue)**

Example

Set color in field Item.Mtrcategory according to field value.

```
var Colors = { WHITE: [16777215], BLUE: [16711680], RED: [255], GREEN: [32768],
    LIGHTYELLOW: [14745599], YELLOW: [52428], BROWN: [19609], GRAY: [12632256]
    , DARKBLUE: [6697728] , LIGHTBLUE: [16769490] , DARKRED: [534458], LIGHTRED:
    [6711008], LIGHTAZURE: [16776118], LIGHTAZURE2: [16449439]
    , LIGHTBLUE2: [16769459], LIGHTGREEN: [14680009], LIGHTYELLOW2: [10876927] };

function ON_LOCATE() {
    SetColors();
}

function ON_ITEM_MTRCATEGORY() {
    SetColors();
}

function SetColors() {
    var vColor;
    if (ITEM.MTRCATEGORY == 101) vColor = Colors.RED;
    else if (ITEM.MTRCATEGORY == 102) vColor = Colors.YELLOW;
    else if (ITEM.MTRCATEGORY == 103) vColor = Colors.GREEN;
    else if (ITEM.MTRCATEGORY == 104) vColor = Colors.BLUE;
    else vColor = Colors.WHITE;

    X.FIELDCOLOR('ITEM.MTRCATEGORY', vColor);
}
```

FOCUSFIELD (FieldName: string)

Sets focus in "FieldName" field.

Example

In sales documents, when user adds a new item line change the focus depending on the selected series.

```
function ON_ITELINES_NEW() {
    if ((SALDOC.SERIES == 7003) || (SALDOC.SERIES == 7023))
        X.FOCUSFIELD('ITELINES.SRCHCODE');
    else
        X.FOCUSFIELD('ITELINES.X_CODE');
}
```

FREE

Destroys the Object and deallocates its memory

INVALIDATEFIELD (FieldName: string)

Repaints the data in a combo box. Usually used after the method "Refresh" when data are altered using SQL update statement.

Example

In Items object, when user selects a specific value (100) in field MTRMARK then set inactive a value of the table MTRMANFCTR and refresh the items shown in the appropriate combo box of the form.

```
function ON_ITEM_MTRMARK() {
    If (ITEM.MTRMARK == 100) {
        X.RUNSQL('UPDATE MTRMANFCTR SET ISACTIVE=0 where COMPANY=1 AND
MTRMANFCTR=10', null);
        X.MTRMANFCTR.REFRESH;
        X.INVALIDATEFIELD('ITEM.MTRMANFCTR');
    }
}
```

INCLUDE (filename: string)

Adds reference to code file (functions etc.). Use full path, when the file is saved in a specific folder or the name of the file, when it is saved in SoftOne Custom SDK Code files.

Example 1

Include code from file saved in a specific directory.

```
X.INCLUDE('C:\\temp\\myfunctions.txt');
```

Example 2

Include code from file saved in SoftOne Custom SDK files.

```
X.INCLUDE('myfunctions.txt');
```

OPENSUBFORM (SubFormName: string)

Opens the Sub form named "SubFormName"

Example

Open sub form Vat Analysis by clicking on a specific button (cmd = 201805011) in sales documents

```
function EXECCOMMAND(cmd) {
    if (cmd == 201805011)
        X.OPENSUBFORM('SFVAT');
}
```

QUICKVIEW (ObjectName: string, ListName: string, Keydata: string)

Opens the Quick View window and displays data for the record "Keydata" of the object "ObjectName", using the browser "ListName".

Example

Open the Quick View named "BROWSER1" of the module Customers, by clicking on a specific button (cmd = 201805021) in sales documents

```
function EXECCOMMAND(cmd) {
    if (cmd == 201805021)
        X.QUICKVIEW('CUSTOMER', 'BROWSER1', 'SALDOC.TRDR');
}
```

PRINTDOCURL (ObjectName, FormName, RecordID, TemplateCode)

Returns a URL (valid for 10 minutes) that prints the template (TemplateCode) of the object (ObjectName) for the record (RecordID) in PDF format and makes it available for download.

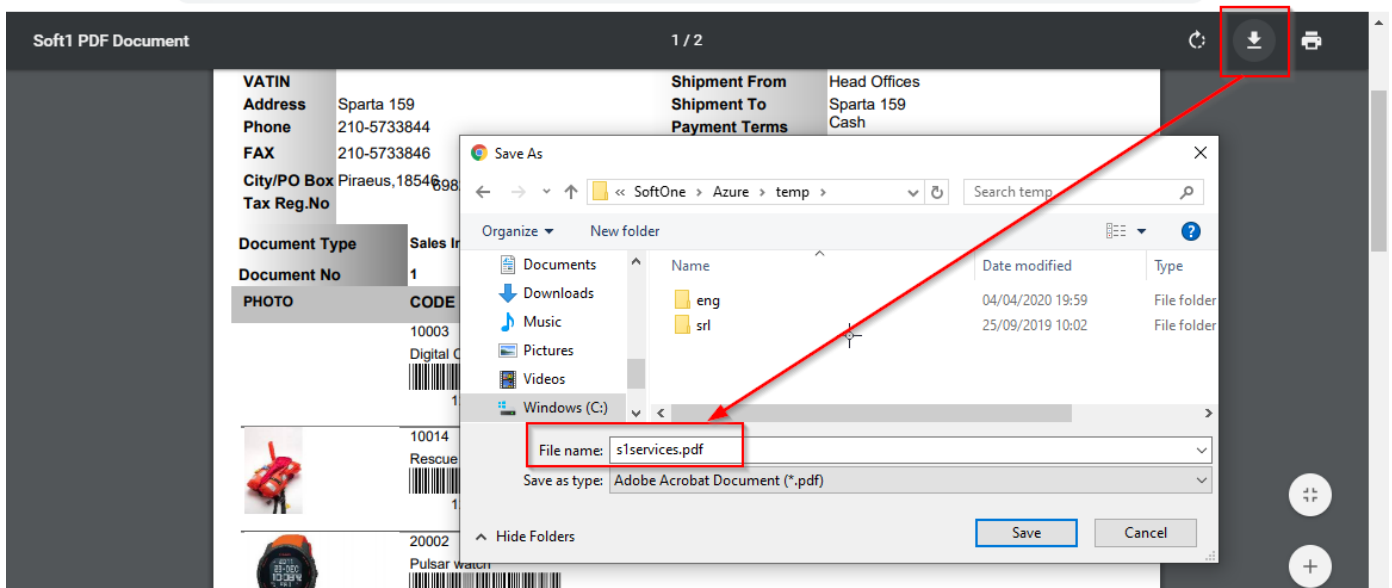
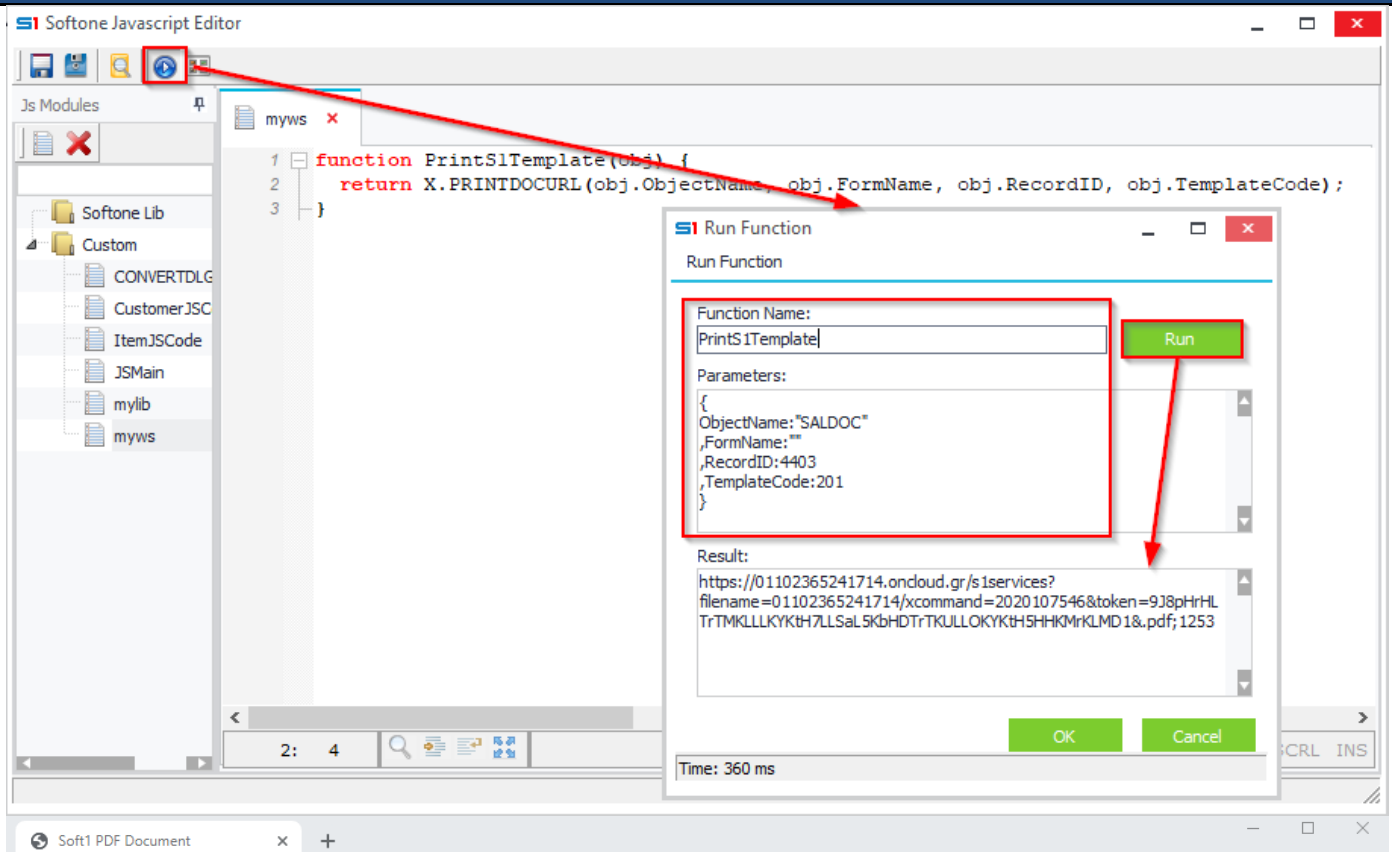
This function is usually used in Web Services, through custom web service requests in Advanced JavaScript.

Note that the above URL works only if the module “Soft1 Open Enterprise Engine” exists on the current installation.

Example

Custom web service that prints a template form in PDF and returns the URL.

```
function PrintS1Template(obj) {  
    return X.PRINTDOCURL(obj.ObjectName, obj.FormName, obj.RecordID,  
obj.TemplateCode);  
}
```



PRINTFORM (TemplateCode: integer, PrinterName: string, FileName: string)

Prints the template that uses the code number "TemplateCode" of the current object and sends it to the printer named "PrinterName". If the print media is "file type" you also need to define the "FileName".

FormCode: Code Number of the printout form

PrinterName: Printer name / File type

Available File Types: Ascii928 / Ascii437 / Word (use 928) / Excel / PDF file / XPS file / Image / HTML / E-mail

FileName: File path when "PrinterName" is file.

Example 1

Locate and print a record in Sales document using a specific printout form and printer.

```
function PrintSalesRecord() {  
    ObjSaldoc = X.CreateObj('SALDOC');  
    ObjSaldoc.DBLocate(12345);  
    ObjSaldoc.PRINTFORM(100,'MyPrinterName',''); //FormCode=100,Printer=MyPrinterName  
}
```

Example 2

[See →Case Study 1](#)

REFRESH

Refreshes the data of a combo box control

Example

```
X.MTRMANFCTR.REFRESH;
```

RUNSQL (ASQL: string; AParams: Variant)

Runs an action query by using the "ASQL" statement.

ASQL: The SQL statement to execute.

AParams: Array list containing ASQL Parameters

Example

Update custom table field with a specific value

```
X.RUNSQL('UPDATE CCCMYTABLE SET CCCMYFIELD=1 WHERE COMPANY=:1', X.SYS.COMPANY);
```

SETDECIMALS (FieldName: string, Decimals: integer)

Sets the number of decimal digits in the field "FieldName"

SETDOCPRINT (PrintNum, Mode, TempID: integer, PrinterName, Caption: string)

Prints a specific number of copies of the TempID form. Must be used under the event ON_DOCPRINT.

PrintNum: Number of copies

Mode: Method (1 = Automatically, 2 = With question)

TempID: Form Code

Printer Name: Name of the printer

Caption: Window title

SETFIELDEDITOR (FieldName: string; Editor: String)

Changes the editor of any field that **already has a predefined editor** inside its "Editor" property.

Example 1

Change the editor of the field ITELINES.MTRL (selector list) in item lines of sales documents

```
X.SETFIELDEDITOR('ITELINES.MTRL','ITEM(W[A.MYCCCFIELD=1 AND EXISTS (...)])');
```

Example 2

Change the editor of the field MTRDOC.WHOUSE (combo box) in sales documents based on the selected series

```
function ON_SALDOC_SERIES() {  
    if ((SALDOC.SERIES == 7001) || (SALDOC.SERIES == 7002))  
        X.SETFIELDEDITOR('MTRDOC.WHOUSE', 'WHOUSE(F[WHOUSE=1001])');  
    else  
        X.SETFIELDEDITOR('MTRDOC.WHOUSE', 'CCCVIEWWHOUSE');  
}
```

SETFIELDVALUE (FieldName: string, Value: Variant)

Assigns the new value to "FieldName" field.

Example

```
X.SetFieldValue('SALDOC.COMMENTS1','MyComments');
```

SETPARAM(PrmName, PrmValue);

Sets a new value in the parameter "PrmName" of the object.

Example 1

```
X.SETPARAM('MyParameterName', X.SYS.FISCPRD);
```

Example 2

```
X.SETPARAM('WARNINGS','OFF');  
X.SETPARAM('NOMESSAGES',1);
```

Example 3

Follow-up an e-mail. Create a new object of SOEMAIL and set its parameters in such a way that it uses "Reply" "Reply to all" or "Forward".

```
var obj = X.CREATEOBJ('SOEMAIL');  
obj.SETPARAM('EMAIL',vID); //vID = ID of previous email record  
obj.SETPARAM('FOLLOW',1); //1=Reply, 2=Reply to all, 3=Forward
```

Example 4

```
var obj = X.CREATEOBJ('PRDORDDOC');  
obj.SETPARAM('IMPORT',1);
```

SETPROPERTY ('EDITOPTIONS', 'READONLY=True/False')

Enables / Disables record editing (read-only fields) of the current object.

Example 1

```
X.SETPROPERTY('EDITOPTIONS', 'READONLY=True');
```

Example 2

[See → ObjectEvents - ON LOCATE \(Example 2\)](#)

SETPROPERTY ('MERCHANGELOG', 'True or False')

Merges and saves any data changes, overwriting existing values. Usually used after populating a virtual table in a view, in order to "disable" the Save button.

Example 1

```
X.SETPROPERTY ('MERGECHANGELOG', 1);
```

Example 2

```
X.SETPROPERTY ('MERGECHANGELOG', 'False');
```

SETPROPERTY ('FIELD', 'FieldName', 'CAPTION', 'NewCaption')

Changes the caption of the field *FieldName*.

Example

```
X.SETPROPERTY ('FIELD', 'SALDOC.TRDR', 'CAPTION', 'MYNEWCAPTION');
```

SETPROPERTY ('FIELD', 'FieldName', 'VISIBLE', 'True/False or 1/0')

Changes the visible property of Fields in grids.

Example

```
X.SETPROPERTY ('FIELD', 'ITELINES.QTY1', 'VISIBLE', 'False');
```

SETPROPERTY ('FIELD', 'FieldName', READONLY, 'True/False or 1/0')

Changes the read-only property of Fields.

Example 1

```
X.SETPROPERTY ('FIELD', 'INST.NAME', 'READONLY', 1);
```

Example 2

```
function ON_INSERT() {
    CheckLockFields();
}
function ON_CANCEL() {
    CheckLockFields();
}
function ON_LOCATE() {
    CheckLockFields();
}
function CheckLockFields() {
    if ((X.SYS.USER == 1) || (X.SYS.USER == 2) )
        && (X.SYS.ISADMIN != 1) ) {
        X.SETPROPERTY('FIELD', 'INST.CODE', 'READONLY', 0);
        X.SETPROPERTY('FIELD', 'INST.NAME', 'READONLY', 0);
    }
    else {
        X.SETPROPERTY('FIELD', 'INST.CODE', 'READONLY', 1);
        X.SETPROPERTY('FIELD', 'INST.NAME', 'READONLY', 1);
    }
}
```

SETPROPERTY ('PANEL', 'PanelName', 'CAPTION', 'NewCaption')

Changes the caption of the panel *PanelName*.

Example

```
X.SETPROPERTY ('PANEL', 'MyPanel', 'CAPTION', 'NewPanelName');
```

SETPROPERTY ('PANEL', 'PanelName', 'VISIBLE', 'True/False or 1/0')

Changes the visible property of Panels, Tabs and Datagrids. Parameter "PANEL" is also used for Datagrids and Tabs.

Example 1

```
X.SETPROPERTY ('PANEL', 'PANEL14', 'VISIBLE', 'TRUE');
```

Example 2

```
X.SETPROPERTY ('PANEL', 'MYPAGE', 'VISIBLE', 'TRUE');
```

Example 3

```
X.SETPROPERTY ('PANEL', 'MYGRID', 'VISIBLE', 1);
```

TOFILE (FileName, AMessage: string)

Saves the "AMessage" text in the "FileName" file. If the file exists, then it is replaced.

WARNING (Message: string)

Displays a warning message on the user's screen.

Example

After posting a Sales Documents (new or edit), display a warning message with the ID of the record.

The id of newly created records is returned through the function X.NEWID. So we create a custom function GETID() that returns the ID for both new or already posted records.

```
function GetID() {  
    return SALDOC.FINDOC > 0 ? SALDOC.FINDOC : X.NEWID;  
}  
  
function ON_AFTERPOST() {  
    X.WARNING('Posting completed, ID = ' + GetID());  
}
```

C. Object Functions

This module demonstrates all the object functions you can use in form scripts.

ASK (ACaption, AMessage: string): integer

Displays a dialog message that displays three button (Yes, No, Cancel) and returns the index of the button: **6=Yes, 7=No, 2=Cancel**

ACaption: Title of the window, **AMessage:** Message displayed inside the window

Example

Confirm saving an entry.

```
function ON_POST() {
    var ans;
    ans = X.ASK("Confirm save", "Continue ?"); // 6=Yes, 7=No, 2=Cancel
    if ((ans == 7) || (ans == 2))
        X.EXCEPTION("Save cancelled by user");
    else
        X.WARNING("Yes pressed");
}
```

ASKYESNO (ACaption, AMessage: string): boolean

Displays a dialog message that displays the button Yes and No and returns 1 for Yes and 0 for No.

ACaption: Title of the window, **AMessage:** Message displayed inside the window

Example

```
function ValidateEntries() {
    if ( X.ASKYESNO("My Title","My Question") == 1)
        X.WARNING("User clicked Yes");
    else
        X.WARNING("User clicked No");
}
```

CASE (IfCase, ThenCase, ElseCase: variant): Variant;

Returns formula "ThenCase" when formula "IfCase" is true. Otherwise, it returns formula "ElseCase"

CHECKACCESS(JobName): Boolean

Checks if Login User has access rights to JobName

Example

```
function CheckUserAccess() {
    if (X.CHECKACCESS("CUSTOMER")) {
        //Add Code here
    }
    if (X.CHECKACCESS("FifoFinPayTerms")) {
        //Add Code here
    }
}
```

CONNECTIONSTATUS: string;

Returns the SoftOne connection status.

0=Standalone, 1=Client, 2=Server, 3=SleepClient, 4=OffLine, 5=SaaS, 6=ESupport

CREATEOBJ (ObjectName: string): OBJECT (IDispatch)

Creates the "ObjectName" object and returns its IDispatch interface. In order to create an object using a specific form use the syntax: CreateObj('ObjectName;**Myform**').

Example 1

Create and save a sales document, while in object Customers, using current located customer record id.

```
function CreateSales() {
    var myObj = X.CreateObj('SALDOC;MYSALESVIEW');
    try {
        myObj.DBINSERT;
        var tblFINDOC = myObj.FindTable('FINDOC');
        var tblITELINES = myObj.FindTable('ITELINES');

        tblFINDOC.Edit;
        tblFINDOC.SERIES = 7062;
        tblFINDOC.TRDR = CUSTOMER.TRDR;

        tblITELINES.Append;
        tblITELINES.MTRL = 123456;
        tblITELINES.QTY1 = 100;
        tblITELINES.PRICE = 50;
        tblITELINES.Post;

        var id = myObj.DBPOST;
        if (id > 0)
            X.WARNING('New id is:' + id);
    }
    catch (e) {
        if (myObj != null)
            X.WARNING("General Error: " + e.message + "\nObject Error: " + myObj.GETLASTERROR);
        else
            X.WARNING("General Error: " + e.message);
    }
    finally {
    }
}
```

Example 2

```
function PrintSelectedDocuments() {
    var vSelRecs = X.GETPARAM('SELRECS');
    var fso = new ActiveXObject('Scripting.FileSystemObject');
    var vFolder = "C:\\Temp";
    var ObjSaldoc = X.CreateObj('SALDOC');
    vQueryBrowser = "SELECT DISTINCT FINDOC, TRDR, (SELECT CODE FROM TRDR WHERE TRDR=FINDOC.TRDR) as TRDRCODE FROM FINDOC WHERE " + vSelRecs + " ORDER BY TRDR";
    ds = X.GETSQLDATASET(vQueryBrowser, null);
    ds.FIRST;
    while (!ds.EOF) {
        vCurTrdr = ds.TRDR;
        vCurTrdrCode = ds.TRDRCODE;
        if (!fso.FolderExists(vFolder + "\\ " + vCurTrdrCode)) {
            fso.CreateFolder(vFolder + "\\ " + vCurTrdrCode);
        }
        ObjSaldoc.DBLocate(ds.FINDOC);
        ObjSaldoc.PRINTFORM(1001, 'PDF file', vFolder + '\\ ' + vCurTrdrCode + '\\ ' + ds.FINDOC + '.PDF');
        ds.NEXT;
    }
}
```

Example 3

[See →Case Study 1](#)

CREATEOBJFORM (ObjectName: string): OBJECT (IDispatch)

Creates an instance of the SoftOne object defined in "ObjectName" and returns its IDispatch interface. Object's form or list can be specified in the same way as in EXEC commands, for example:

```
X.CREATEOBJFORM('SALDOC[FORM=MyFormName, LIST=MyListName]');
```

The target object's record can be displayed on user's screen (before posting the data) using the function ShowObjForm instead of DBPost.

Example

Locate a record of Items entity. Add a new purchase document using the located item record id(MTRL). Display the purchase document on screen, wait for the user to save it and then display the new id in a message box.

```
function CreatePurchase() {
    var myObj = X.CreateObjForm('PURDOC[FORM=MyFormName]');
    try {
        myObj.DBINSERT;
        var tblFINDOC = myObj.FindTable('FINDOC');
        var tblITELINES = myObj.FindTable('ITELINES');

        tblFINDOC.Edit;
        tblFINDOC.SERIES = 2001;
        tblFINDOC.TRDR = 123456;

        tblITELINES.Append;
        tblITELINES.MTRL = ITEM.MTRL;
        tblITELINES.QTY1 = 100;
        tblITELINES.PRICE = 50;
        tblITELINES.Post;

        var id = myObj.SHOWOBJFORM();
        if (id > 0)
            X.WARNING('New id is: ' + id);
    }
    catch (e) {
        if (myObj != null)
            X.WARNING("General Error: " + e.message + "\nObject Error: " + myObj.GETLASTERROR);
        else
            X.WARNING("General Error: " + e.message);
    }
    finally {
    }
}
```

DOUBLE (StrNum: string): real

Converts StrNum to a real number and returns the result.

DIR (Name: string): string

Returns the "Name" directory. Returned names are:

Soft1Data: Soft1 Profile Directory

UserData: User Profile Directory

EXE: SoftOne Dir

HTML: Soft1 Web Server Dir

SDK: Soft1 SDK files Dir

EVAL (Formula: string): Variant

Calculates the "Formula" and returns the result. Used primarily for executing internal SoftOne functions.

Example 1

```
DateforSQLQuery = X.EVAL('SQLDATE(SALDOC.TRNDATE)')
```

Example 2

```
EasterDate = X.EVAL('DateToStr(GetPasxaDate('+X.SYS.FISCPRD+'))');
```

Example 3

```
function FillFinalDate() {
    if ((GUARANTY.ISNULL('MTRL') == 0) && (GUARANTY.ISNULL('FROMDATE') == 0)) {
        var vGUARTIME = X.SQL("SELECT ISNULL(GUARTIME,0) FROM MTRL WHERE MTRL=:1",
GUARANTY.MTRL);
        GUARANTY.FINALDATE = X.EVAL("FormatDateTime('dd/mm/yyyy'
, INCMONTH(GUARANTY.FROMDATE, " + vGUARTIME + "))");
    }
}
```

Example 4

[See → Case Study 3](#)

EXEC (Command: string): variant

Executes the SoftOne "Command" based on the following parameters:

A. Button: xxx

The corresponding button of the toolbar is executed. Syntax is: **X.EXEC("button:XXX");**

Toolbar Buttons are: Browse, View, New, Copy, Save, Cancel, Delete, Prior, Next, Print.

Example 1

Save entry

```
function SaveEntry() {
    X.EXEC('Button:Save');
}
```

Example 2

Cancel entry

```
function CancelEntry() {
    X.EXEC('Button:Cancel');
}
```

B. XCMD: xxx

- If it is followed by an integer then it runs an internal command of the application or any other custom command (e.g. custom form button). Syntax is: **X.EXEC('xcmd:XXX');**
All the available internal commands can be found in ANNEX Tables
-

- If it is followed by a menu command (e.g. ObjectName, TableName or SoftOneScript) then it executes the corresponding job.
Notice that if you need to open an object that closes automatically when user clicks on Save, then you have to use the parameter ,N as in example 3.

Example 1

Display open item window inside a sales document form

```
X.EXEC ('XCMD:1031')
```

Example 2

Open customer

```
X.EXEC ('XCMD:CUSTOMER')
```

Example 3

Open new sales document from object Customers assigning the current value to field "SALDOC.TRDR"

```
X.EXEC ('XCMD:SALDOC,N["SET FINDOC.TRDR="+CUSTOMER.TRDR+""]')
```

Example 4

Run SoftOne SBSL script "myscript" using the parameter myparam=1.

```
X.EXEC ('XCMD:FormImport,ScriptName:myscript,myparam:1')
```

C. ACMD: xxx

Runs the xxx internal tool. All the available system tools can be found in ANNEX table "[ACMD System Tools](#)"

Example 1

```
X.EXEC ('ACMD:acZReportSignB')
```

D. Script: xxx

Runs the script function that follows it.

Example 1

```
X.EXEC ('Script:Myfunction')
```

E. Code: xxx

Runs the SoftOne callpublished function that follows it. Functions can be found in the section [Libraries](#) of Chapter "SoftOne Batch Script Language" (SBSL).

Example 1

```
X.EXEC ('CODE:ModuleIntf.GetDataset', X.MODULE, 'CCCMYTABLE');
```

Example 2

Send SMS using the Web SMS Account defined in Settings.

```
X.EXEC ('CODE:SysRequest.XSendWebSMS', '0123456789', 'SMS Message');
```

Example 3

Send email using doSendMail3.

```
X.EXEC('CODE:SysRequest.doSendMail3', strTO, strCC, strBCC, strSubject, strBodyPlain, strBodyHTML, strAttachment, strFromName, vEmailAccount);
```

Example 4

Add menu items dynamically to the Browser Context Menu of any object.

```
function ON_CREATE() {  
    var vBrowserMenu = X.EXEC('CODE:ModuleIntf.FindXStrings', X.MODULE, 'BRMENU');  
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201910011=Customer Financial');  
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201910012=1;Custom ReUpdate');  
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201910021=1;SubJob1?MySubMenu');  
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201910022=1;SubJob2?MySubMenu');  
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201910023=3;Displayed on Lines  
Analysis Browser');  
    X.EXEC('CODE:SysRequest.RefreshPopupMenu', X.MODULE, 'BRMENU', 1);  
    // See section G. Object Events (Example 2)  
}
```

Example 5

Run (include) function from SoftOne Designer. The function named 'myFunction' is inside the script 'myDesignerScript' of SoftOne Designer.

```
var myDesignerScript = X.EXEC('CODE:ModuleIntf.GetTextValue', 'myScriptName');  
X.EXEC('CODE:SysRequest.ExecuteXScript', X.Module, 1, myDesignerScript,  
'myFunction');    // function inside SoftOne Designer script
```

F. Reljobs: xxx

Executes the related job.

FINDTABLE (TableName: string): variant

Returns the Dataset (IDispatch interface) of the table "**TableName**".

Example 1

```
ObjSaldoc = X.CreateObj('SALDOC');  
tblFindoc = ObjSaldoc.FindTable('FINDOC');
```

Example 2

[See → Case Study 4](#)

FILTERSUM (FieldName: string, Filter: string): real

Summarizes the data of the "**FieldName**" field of the given datagrid for entries that meet the "**Filter**" expression. Use filter "1=1" when there is no need to apply any filters.

Example 1

Summarize the data of the field ITELINE.QTY1 (only for lines that MTRUNIT=1) each time users post a line.

```
function ON_ITELINE_AFTERPOST() {  
    var TotQty1 = X.FILTERSUM('ITELINE.QTY1', 'MTRUNIT=1');  
    X.WARNING(TotQty1);  
}
```

Example 2

Check if an item already exists (double entry) in ITELINE datagrid, display a message and delete the line.

Summarize all the MTRLs inside the datagrid, that have the same value as the one inserted, and check if this number is greater than MTRL (id).

```
function ON_ITELINE_AFTERPOST() {  
    var vMTRLTimes = X.FILTERSUM('ITELINE.MTRL', 'MTRL='+ITELINE.MTRL);  
    if (vMTRLTimes > ITELINE.MTRL) {  
        X.WARNING(ITELINE.X_NAME + ' exists more than one time!');  
        ITELINE.DELETE;  
    }  
}
```

FORM: string

Returns the form name of the current object

Example

```
function ON_FORMLOAD() {  
    CheckUserAccess();  
}  
  
function ON_LOCATE() {  
    CheckUserAccess();  
}  
  
function CheckUserAccess() {  
    if ((X.FORM == 'SOACTIONDBView') && ((X.SYS.USER == 10) || (X.SYS.USER == 20))) {  
        X.EXCEPTION("\nYou do not have permissions to access the form:\n" + X.FORM);  
    }  
}
```

FORMATFLOAT (number: float, decimalsformat: string): string

decimalsformat: Number of decimals or character that retrieves company decimals (% C Q V P)

Returns the float "Value" in alphanumeric format

Example

```
var fnum = X.FORMATFLOAT(1234, 'V');
```

FORMATDATE (Dateformat: string, Date: TDateTime): string

Returns the "Date" in alphanumeric format

Example 1

Return field TRNDATE of sales documents in sqldate format.

```
strDate = X.FORMATDATE('yyyymmdd', SALDOC.TRNDATE) //e.g. 15 June 2018 returns 20180615
```

For use in sql queries:

```
strDate = String.fromCharCode(39) + strDate + String.fromCharCode(39);
```

Example 2

Return UPDDATE field of customers in specific datetime formats

```
strDate1 = X.FORMATDATE('yyyy-mm-dd HH:MM:SS', CUSTOMER.UPDDATE)
strDate2 = X.FORMATDATE('yyyy-mm-dd 00:00:00', CUSTOMER.UPDDATE)
```

FORMATTEXT (Text: string, parameters: string): string

Returns a formatted text using the parameters defined

Example

```
function ON_POST() {
    var vCompany = X.SYS.COMPANY;
    var vUser = X.SYS.USER;
    var vFiscprd = X.SYS.FISCPRD;
    var str = X.FormatText('Company: {0}\nUser: {1}\nFiscal Year: {2}', vCompany,
vUser, vFiscprd);
    X.WARNING(str);
}
```

FROMFILE (FileName: string): string

Returns the data of the file "FileName"

Example

```
X.WARNING(X.FROMFILE(X.DIR("EXE")+"\\MyText.txt"));
```

ExecuteNetFunction(Dll FileName, Class (with Namespace), FunctionName, Params);

SBSL Function that runs a .NET DLL Function using the defined Params.

[See → SBSL - ExecuteNetFunction](#)

GETYEARPERIOD (Adate: TDateTime): Variant

Returns the year and the period of date "Adate"

GETLASTERROR: string

Returns the last error message

Example

```
function InsertNewCustomer() {
    var myObj = X.CREATEOBJ("CUSTOMER");
    try {
        var tblCustomer = myObj.FINDTABLE("TRDR");
        myObj.DBINSERT;
        tblCustomer.CODE = "*";
        tblCustomer.NAME = "TEST_CUSTOMER";
        newid = myObj.DBPOST;
        X.WARNING("New Customer ID is:"+newid);
    }
    catch (e) {
        if (myObj != null)
            X.WARNING("General Error: " +e.message+"\nObject Error: "+myObj.GETLASTERROR);
        else
            X.WARNING("General Error: " +e.message);
    }
    finally {
    }
}
```

GETPARAM(PrmName)

Returns the value of PrmName of the object

Example 1

```
X.GETPARAM('MyParameterName');
```

Example2

```
function ON_CREATE() {
    X.GETPARAM('SOSOURCE');
}
```

GETPROPERTY('CUSTOMFIELDS, **TABLENAME**): string

Returns the custom fields of the defined table. Returned data are separated by semicolons.

Example 1

Button inside Sales Documents that displays the custom fields of tables FINDOC and MTRLINES

```
function EXECCOMMAND(cmd) {  
    if (cmd == 202007133) { //Get Findoc-Mtrlines Custom Fields  
        var vCustomFieldsFindoc = X.GETPROPERTY('CUSTOMFIELDS,FINDOC');  
        var vCustomFieldsMtrlines = X.GETPROPERTY('CUSTOMFIELDS,MTRLINES');  
        X.WARNING("Custom Fields\n\nFINDOC: " + vCustomFieldsFindoc + "\nMTRLINES: " +  
vCustomFieldsMtrlines);  
    }  
}
```

The screenshot shows the SAP Sales Documents interface for 'Standard Company: SISN000141- 18/12/2018'. The 'Custom Fields' button is highlighted with a red box. A message box titled 'Message!' is displayed, showing the custom fields for 'FINDOC' and 'MTRLINES'. The message box contains the text: 'FINDOC: CCCDATE1;CCCINT1;CCCSTR1' and 'MTRLINES: CCCMYTRDR'. The 'OK' button is visible in the message box.

Series: 7062SISN Type: 7062 Sales Invoice - D Document: SISN000141
Date: 18/12/2018 Branch: 1000 Head Offices Warehouse: 1000 Central

Custom Fields Master Table Name

General data Delivery - Transfer International transactions Other data

Customer: Message!
Cust. branch:
Salesperson:
Currency:
Comment:

Items Ser

Item	Material	Description	Quantity	Price	Total
1	20003	GPS	20	513.50	
2	20017	Altimeter aviation	80	697.84	
3	20024	Door 12-280 * 30	100.00	5,000.00	

GETPROPERTY('MASTERTABLENAME'): string

Returns the name of the master table for the current object.

Example 1

Button inside Sales Documents that displays the Master Table Name.

```
function EXECCOMMAND(cmd) {  
  if (cmd == 202007134) { //Get Master Table Name  
    var vMasterTableName = X.GETPROPERTY('MASTERTABLENAME');  
    X.WARNING(vMasterTableName);  
  }  
}
```

The screenshot shows the 'Sales Documents' application window. The title bar indicates 'Sales Documents' with a close button. The main header displays 'Standard Company: SISN000141- 18/12/2018'. Below the header, there are fields for 'Series: 7062SISN', 'Type: 7062 Sales Invoice - D', 'Document: SISN000141', 'Date: 2/2018', 'Branch: 1000 Head Offices', and 'Warehouse: 1000 Central'. A 'Custom Fields' section contains a button labeled 'Master Table Name', which is highlighted with a red box. A red arrow points from this button to a 'Message!' dialog box that displays 'SALDOC'. The dialog box has an 'OK' button. The background shows a list of items with columns for 'Code', 'Description', 'Quantity', and 'Price'. The list contains 8 items, with the first item being '20003' and the last item being '50014'.

	Code	Description	Quantity	Price
1	20003			
2	20017	Altimeter aviation	80	697.84
3	20024	Door 12-280 * 30	100.00	5,000.00
4	24001	Yeast 02 (Puff)	100.00000000	200.00
5	20010	Modern Kitchen	250	165.10
6	20004	Car GPS	50	799.50
7	30001	Rope climbing 10 (closed container 60	50	166.00
8	50014	Adjustable shock absorbers	100	68.64

GETPROPERTY('HASCHANGES'): boolean

Returns true when record is on edit mode (record has pending changes and Save button is activated).

Example 1

Button that checks whether the record has pending changes.

```
function EXECCOMMAND(cmd) {  
  if (cmd == 202007135) { //Check if on edit mode  
    var vPendingChanges = X.GETPROPERTY('HASCHANGES');  
    X.WARNING(vPendingChanges);  
  }  
}
```

Standard Company: ORDR000001- 07/07/2020

Series: 7023ORDR Type: 7023 Sales Order (Reta Document: ORDR000001 Number: 1

Date: 07/07/2020 Branch: 1000 Head Offices Warehouse: 1000 Central

Custom Fields Master Table Name Has Pending Changes

General data Delivery - Transfer International transactions Other data

Customer: 108 Kousk...
Cust. branch:
Salesperson: 00015 Clarcs...
Currency: 100 Exchange rate:
Comment:

Shipping: Payment: 1003 Check T.R.No.

	Search by	Code	Description	Qty 1	Price	Disc....	Value
1	10001	10001	TV LED 32 LG RTDXA32	2	1.34		
2	10002	10002	TV LCD 21	3	2.34		
3	10003	10003	Digital Camera 8 MP	5	2.65		1

Standard Company: ORDR000001- 07/07/2020

Series: 7023ORDR Type: 7023 Sales Order (Reta Document: ORDR000001 Number: 1

Date: 07/07/2020 Branch: 1000 Head Offices Warehouse: 1000 Central

Custom Fields Master Table Name Has Pending Changes

General data Delivery - Transfer International transactions Other data

Customer: 108 Kousk...
Cust. branch:
Salesperson: 00015 Clarcs...
Currency: 100 Exchange rate: 1
Comment: test comments changed

Shipping: Payment: 1003 Check T.R.No.

	Search by	Code	Description	Qty 1	Price	Disc....	Value
1	10001	10001	TV LED 32 LG RTDXA32	2	1.34		

GETPROPERTY('GRIDSELECTED:GRIDNAME|FIELDNAME'): string

Returns the **FIELDNAME** values of the **selected rows** for the **GRIDNAME** grid. Returned data are separated by \r\n.

Example 1

Add a new option inside the pop-up menu of Sales Documents item lines grid (**MAINGRID**), that returns the selected **MTRL** in a warning message. Display also the data of the first selected record in message box.

```
function ON_CREATE() {
    ChangeContextMenus(); //Change context menu items
}

function ChangeContextMenus() {
    //===== Items Grid Menu =====//
    var vGridMenu = X.EXEC('CODE:ModuleIntf.FindXStrings', X.MODULE, 'GRIDMENU');
    X.EXEC('CODE:PiLib.TStringsAdd', vGridMenu, '--');
    X.EXEC('CODE:PiLib.TStringsAdd', vGridMenu, '201707195=Relative campaigns');
    X.EXEC('CODE:PiLib.TStringsAdd', vGridMenu, '201707196=Lots availability');
    X.EXEC('CODE:PiLib.TStringsAdd', vGridMenu, '201907131=Grid Selected Items');
    X.EXEC('CODE:SysRequest.RefreshPopupMenu', X.MODULE, 'GRIDMENU', 0);
}

function EXECCOMMAND(cmd) {
    if (cmd == 201907131) { //Grid Selected Records
        var selectedMTRL = X.GETPROPERTY('GRIDSELECTED:MAINGRID|MTRL');
        X.WARNING(selectedMTRL);
        var ArraySelMtrl = selectedMTRL.split("\r\n");
        if (ArraySelMtrl.length > 0) X.WARNING(ArraySelMtrl[0]);
    }
}
```

The screenshot displays the 'Sales Documents' application window. At the top, there's a header bar with 'Sales Documents' and a close button. Below it is a toolbar with icons for List, Filter, Star, New, Save, Delete, and navigation arrows. The main area shows 'Standard Company: SISN000141- 18/12/2018'. Below this are input fields for Series (7062SISN), Type (7062 Sales Invoice - D), Document (SISN000141), Date (2/2018), Branch (1000 Head Offices), and Warehouse (1000 Central). A 'Message!' dialog box is open, displaying a list of selected MTRL values: 1642, 1656, 1649, and 1643. To the right, a context menu is visible, listing various options like 'Serial numbers', 'Coverage of pending', 'Cost data', 'Alternative items', 'Item agreement', 'Production orders', 'Activity Based Costing analysis', 'Relative campaigns', 'Lots availability', and 'Grid Selected Items'. The 'Grid Selected Items' option is highlighted with a red box and a red arrow pointing to it. Below the message box, a table with columns 'Code' and 'Description' is visible, showing items like GPS, Altimeter aviation, Door 12-280 * 30, Yeast 02 (Puff), Modern Kitchen, Car GPS, Rope climbing 10 (closed container 60, and Adjustable shock absorbers.

	Code	Description
1	20003	GPS
2	20017	Altimeter aviation
3	20024	Door 12-280 * 30
4	24001	Yeast 02 (Puff)
5	20010	Modern Kitchen
6	20004	Car GPS
7	30001	Rope climbing 10 (closed container 60
8	50014	Adjustable shock absorbers

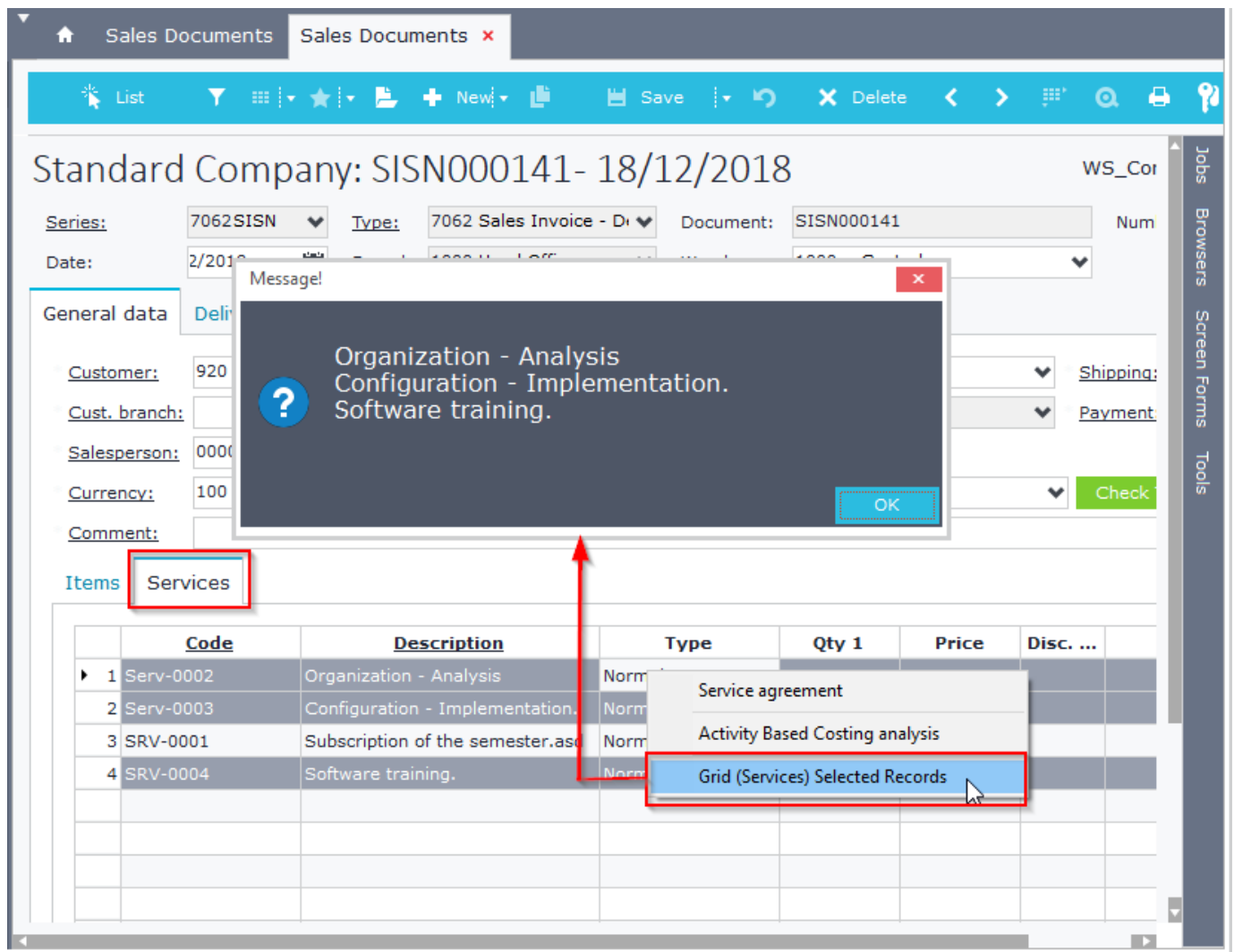
Example2

Add a new option inside the pop-up menu of Sales Documents service lines grid (**MAINGRID121**), that displays a warning message with the selected service descriptions (**X_NAME**).

```
function ON_CREATE() {
    ChangeContextMenus();
}

function ChangeContextMenus() {
    //===== Services Grid Menu =====//
    var vGridMenuSrv = X.EXEC('CODE:ModuleIntf.FindXStrings', X.MODULE, 'SRVGRIDMENU');
    X.EXEC('CODE:PiLib.TStringsAdd', vGridMenuSrv, '--');
    X.EXEC('CODE:PiLib.TStringsAdd', vGridMenuSrv, '201907132=Grid (Services) Selected
Records');
    X.EXEC('CODE:SysRequest.RefreshPopupMenu', X.MODULE, 'SRVGRIDMENU', 0);
}

function EXECCOMMAND(cmd) {
    if (cmd == 201907132) { //Grid Services Selected Records
        var selSrvName = X.GETPROPERTY('GRIDSELECTED:MAINGRID121|X_NAME');
        X.WARNING(selSrvName);
        var ArraySelSrvName = selSrvName.split("\r\n");
        if (ArraySelSrvName.length > 0)
            X.WARNING(ArraySelSrvName[0]);
    }
}
```



GETSQLDATASET (ASQL: string; AParams: Variant): TDataset

Executes the "ASQL" statement and populates a Dataset with the results set.

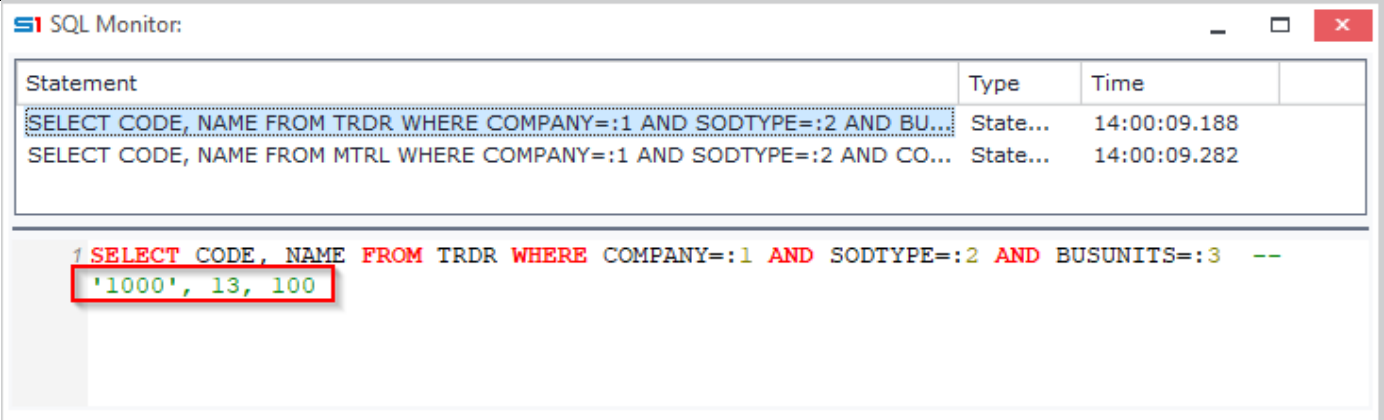
ASQL: SQL statement

AParams: Comma separated list containing query parameters

Example 1

Populate a dataset with data from Customers (Code, Name). Filter query results so that only customers with specific business unit (100) and company (Login) are displayed.

```
var qry = "SELECT CODE, NAME FROM TRDR WHERE COMPANY=:1 AND SODTYPE=:2 AND  
BUSUNITS=:3";  
var ds = X.GETSQLDATASET(qry, X.SYS.COMPANY,13,100);
```

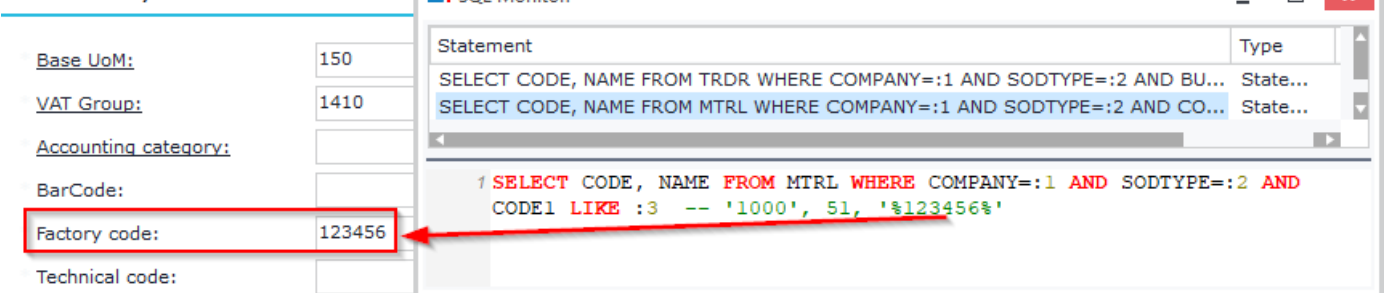


Example 2

Inside the object ITEM, create a script that populates a dataset with all the Items that contain the value of field Code2 in any position of field Code1.

```
var qry = "SELECT CODE, NAME FROM MTRL WHERE COMPANY=:1 AND SODTYPE=:2 AND  
CODE1 LIKE :3";  
var ds = X.GETSQLDATASET(qry, X.SYS.COMPANY,51,'%'+ITEM.CODE2+'%');
```

Fast entry



Example 3

Populate a dataset with the balance per year of the located item.

```
X.GETSQLDATASET("SELECT FISCPRD, ISNULL(QTY1,0) AS BAL FROM MTRDATA WHERE MTRL=:1",  
ITEM.MTRL);
```

HTTPCALL (URL: string, PostData: string, Headers: string, Method: string): Variant

Makes HTTP request. Used for interaction with web services, API and web sites in general.

URL: URL of the web request

PostData: post data of the web request

Headers: Request headers split by \r\n

Method: GET, POST, PUT, PATCH, DELETE method of the web call

Example 1

Web request using GET method.

```
function myGetWebRequest () {
    try {
        var vAns = X.HTTPCALL("https://www.mysite.com/webcall1");
        if (vAns != "") {
            //...
            return true;
        } else {
            X.WARNING(vAns);
        }
    } catch (err) {
        X.WARNING(err.message);
    }
    return false;
}
```

Example 2

Web request using POST method.

```
function myPostWebRequest (vKey) {
    var ans = {};

    var vMethod = "POST";
    var vHeaders = "Content-Type:application/json \r\nAuthorization: Bearer " + vKey;
    var vWebRequest = "https://www.mysite.com/webcall2";

    var vBody = {};
    vBody.username = "user1";
    vBody.password = "pass";
    var postdata = JSON.stringify(vBody);

    try {
        var resp = JSON.parse(X.HTTPCALL(vWebRequest, postdata, vHeaders, vMethod));
        ans.success = resp.success;
        ans.data = resp.data;
        ans.message = resp.message;
    } catch (ex) {
        ans.success = false;
        ans.message = ex.message;
    }
    return JSON.stringify(ans);
}
```

ID (TableName: string, CodeValue: string): integer

Searches the "CodeValue" inside the field Code of the main table of the object "**ObjectName**" and returns its primary key value.

Example

Display a message on screen with the ID of customer with code '001'.

```
X.WARNING(X.ID("CUSTOMER", "001"));
//The sql query executed uses login company (e.g. 1000)
//SELECT TRDR FROM TRDR A WHERE A.COMPANY=1000 AND A.SODTYPE=13 AND A.CODE='001'
```

INPUTBOX (Prompt: string, DefaultValue: string): string

Displays a dialog message that allows the user to enter text and returns the text.

Example

Display a dialog message that asks the user to enter the e-mail.

```
var inputMail = X.INPUTBOX('Please enter recipient e-mail:', '');
```

INPUTQUERY (Title: string, Prompt: string, DefaultVal:Variant, vPassword:integer) : Variant

Displays a dialog message that allows the user to enter text. Returns the text entered by the user when the user selects OK. Returns blank when Cancel or Esc is pressed.

ACaption: Title of the window,

APrompt: Internal window text (input field title),

Value: Preselected value of the input field,

vPassword: Displays the input text with asterisks (1 = Asterisk Text, 0 = Normal Text)

Examples

[See →Case Study 1](#)

[See → Case Study 4](#)

ISVALIDCONTRACT: boolean

Checks if serial number - contract is valid

LIST: string

Returns the name of the current browser

Example

```
var vListName = X.LIST;
```

LOCALE: integer

Returns the Current Locale ID in language code (LCID)

Example

```
X.WARNING(X.LOCALE); // For Greek it returns 1032
```

LOCALESTRING (value: string): string

Returns the login language of variable "value". Requires the existence of specific language files.

LOGINDATE: TDateTime

Returns the current date(same as X.SYS.LOGINDATE)

Example

```
X.WARNING("LoginDate is: " + X.LOGINDATE);  
//Returns: Login Date is: Sat Oct 5 00:00:00 UTC+0300 2013
```

MULTISQL: variant

Creates an IDispatch interface that can be used **to execute multiple SQL queries in a single transaction**.

SQL statements are added using the function "**AddSQL**" and are executed using function "**Execute**".

The returned datasets are accessed through the function "**DataSets**".

Note: Queries that use **X.SYS** commands are resolved using the function "**Resolve**" inside AddSQL.

Example

Execute queries that retrieve data from Customers, Items and Sales Statistics.

Use different dataset functions JSON, SUM, RECORDCOUNT, etc. on the returned datasets.

Check that all queries are executed in a single transaction (SQL Monitor).

```

function Example_MULTISQL() {
    var mSQL = X.MULTISQL();

    var qryCustomers = "SELECT TOP 20 A.TRDR, A.NAME, ISNULL(B.LBAL,0) AS BALANCE "
        + " FROM TRDR A "
        + " LEFT OUTER JOIN TRDFINDATA B ON B.COMPANY=:X.SYS.COMPANY AND B.TRDR=A.TRDR "
        + " WHERE A.COMPANY=:X.SYS.COMPANY AND A.SODTYPE=13 "
        + " AND A.ISACTIVE=1 AND B.LBAL>100";
    mSQL.AddSQL("dtCustomers", X.RESOLVE(qryCustomers));

    var qryItems = "SELECT TOP 20 A.MTRL, A.CODE,A.NAME,ISNULL(B.QTY1,0) AS QTY1 "
        + " FROM MTRL A "
        + " LEFT OUTER JOIN MTRDATA B ON B.COMPANY=A.COMPANY "
        + " AND A.MTRL=B.MTRL AND B.FISCPRD=:X.SYS.FISCPRD "
        + " WHERE A.COMPANY=:X.SYS.COMPANY AND A.SODTYPE=51";
    mSQL.AddSQL("dtItems", X.RESOLVE(qryItems));

    var qrySalesStats = "SELECT A.TrnDate,A.Findoc,A.Trdr, "
        + " B.CODE AS X_TCODE ,B.NAME AS X_TNAME, "
        + " C.CODE AS X_MCODE ,C.NAME AS X_MNAME, "
        + " A.MtrTrn ,ISNULL(A.SalQty1,0) AS SalQty1 ,ISNULL(A.SalVal,0) AS SalVal "
        + " FROM VMTRSTAT A "
        + " LEFT OUTER JOIN TRDR B ON A.Trdr=B.TRDR "
        + " LEFT OUTER JOIN MTRL C ON A.Mtrl=C.MTRL "
        + " WHERE A.COMPANY=:X.SYS.COMPANY "
        + " AND A.FISCPRD=:X.SYS.FISCPRD "
        + " AND A.PERIOD=:X.SYS.PERIOD "
        + " AND A.SoSource=1351 AND A.TSodType=13 "
        + " ORDER BY A.TrnDate,A.Findoc,A.MtrTrn "
    mSQL.AddSQL("dtSalesStats", X.RESOLVE(qrySalesStats));

    mSQL.Execute; //Executes all queries

    var dtCustomers = mSQL.DataSets('dtCustomers');
    var dtItems = mSQL.DataSets('dtItems');
    var dtSalesStats = mSQL.DataSets('dtSalesStats');

    var vTotCusLbal = dtCustomers.Sum("BALANCE", "NAME LIKE 'B*'");
    var vItemsJson = dtItems.JSON;

    if (dtSalesStats.RECORDCOUNT > 0) {
        dtSalesStats.FIRST;
        while (!dtSalesStats.EOF) {
            //Enter your Code Here
            dtSalesStats.NEXT;
        }
    }
}

```


The screenshot shows the SQL Monitor application window. The title bar reads "SQL Monitor: S1Developers". The main window contains a table with three columns: "Statement", "Type", and "Time". The first row of the table is highlighted in blue and contains the following text:

Statement	Type	Time
SELECT TOP 20 A.TRDR, A.NAME, ISNULL(B.LBAL,0) AS BALANCE FROM TRDR A L...	State...	11:13:34.264

Below the table, the full text of the first statement is displayed in a monospaced font, with some words highlighted in red and others in green. The statement is as follows:

```

1 SELECT TOP 20 A.TRDR, A.NAME, ISNULL(B.LBAL,0) AS BALANCE FROM TRDR A
LEFT OUTER JOIN TRDFINDATA B ON B.COMPANY=1000 AND B.TRDR=A.TRDR WHERE A.
COMPANY=1000 AND A.SODTYPE=13 AND A.ISACTIVE=1 AND B.LBAL>100
2 SELECT TOP 20 A.MTRL, A.CODE,A.NAME,ISNULL(B.QTY1,0) AS QTY1 FROM MTRL A
LEFT OUTER JOIN MTRDATA B ON B.COMPANY=A.COMPANY AND A.MTRL=B.MTRL AND B.
FISCPRD=2020 WHERE A.COMPANY=1000 AND A.SODTYPE=51
3 SELECT A.TrnDate,A.Findoc,A.Trdr,B.CODE AS X_TCODE ,B.NAME AS X_TNAME, C.
CODE AS X_MCODE ,C.NAME AS X_MNAME, A.MtrTrn ,ISNULL(A.SalQty1,0) AS
SalQty1 ,ISNULL(A.SalVal,0) AS SalVal FROM VMTRSTAT A LEFT OUTER JOIN
TRDR B ON A.Trdr=B.TRDR LEFT OUTER JOIN MTRL C ON A.Mtrl=C.MTRL WHERE A.
COMPANY=1 AND A.FISCPRD=2020 AND A.PERIOD=9 AND A.SoSource=1351 AND A.
TSodType=13 ORDER BY A.TrnDate,A.Findoc,A.MtrTrn
  
```

```
1 function Example_MULTISQL() {  
2     var mSQL = X.MULTISQL();  
3  
4  
5     var qryCustomers = "SELECT TOP 20 A.TRDR, A.NAME, ISNULL(B.LBAL,0) AS BALANCE "  
6         +" FROM TRDR A "  
7         +" LEFT OUTER JOIN TRDFINDATA B ON B.COMPANY=:X.SYS.COMPANY AND B.TRDR=A.TRDR "  
8         +" WHERE A.COMPANY=:X.SYS.COMPANY AND A.SODTYPE=13 AND A.ISACTIVE=1 AND B.LBAL>100";  
9     mSQL.AddSQL("dtCustomers", X.RESOLVE(qryCustomers));  
10  
11     var qryItems = "SELECT TOP 20 A.MTRL, A.CODE,A.NAME,ISNULL(B.QTY1,0) AS QTY1 "  
12         +" FROM MTRL A "  
13         +" LEFT OUTER JOIN MTRDATA B ON B.COMPANY=A.COMPANY AND A.MTRL=B.MTRL AND B.FISCPRD=:X.SY  
14         +" WHERE A.COMPANY=:X.SYS.COMPANY AND A.SODTYPE=51";  
15     mSQL.AddSQL("dtItems", X.RESOLVE(qryItems));  
16  
17     var qrySalesStats = "SELECT A.TrnDate,A.Findoc,A.Trdr,B.CODE AS X_TCODE ,B.NAME AS X_TNAME,"  
18         +" C.CODE AS X_MCODE ,C.NAME AS X_MNAME,"  
19         +" A.MtrTrn ,ISNULL(A.SalQty1,0) AS SalQty1 ,ISNULL(A.SalVal,0) AS SalVal "  
20         +" FROM VMTRSTAT A "  
21         +" LEFT OUTER JOIN TRDR B ON A.Trdr=B.TRDR "  
22         +" LEFT OUTER JOIN MTRL C ON A.Mtrl=C.MTRL "  
23         +" WHERE A.COMPANY=1 "  
24         +" AND A.FISCPRD=:X.SYS.FISCPRD "  
25         +" AND A.PERIOD=:X.SYS.PERIOD "  
26         +" AND A.SoSource=1351 AND A.TSodType=13 "  
27         +" ORDER BY A.TrnDate,A.Findoc,A.MtrTrn "  
28     mSQL.AddSQL("dtSalesStats", X.RESOLVE(qrySalesStats));  
29  
30     mSQL.Execute; //Executes all queries  
31  
32     var dtCustomers = mSQL.DataSets('dtCustomers');  
33     var dtItems = mSQL.DataSets('dtItems');  
34     var dtSalesStats = mSQL.DataSets('dtSalesStats');  
35  
36     var vTotCusLbal = dtCustomers.Sum("BALANCE", "NAME LIKE '*B*'");  
37     var vItemsJson = dtItems.JSON;  
38  
39     if (dtSalesStats.RECORDCOUNT > 0) { //Iterate dataset  
40         dtSalesStats.FIRST;  
41         while (!dtSalesStats.EOF) {  
42             //Enter your code here  
43             dtSalesStats.NEXT;  
44         }  
45     }  
46 }
```

NEWID: integer

Returns the ID of the entry saved. Used in after post event.

Example

Return the ID of a sales document entry (Insert or Update)

```
function ON_AFTERPOST() {
    var vFindocID;
    vFindocID = GetID();
}

function GetID() {
    return SALDOC.FINDOC > 0 ? SALDOC.FINDOC : X.NEWID;
}
```

PASSWORDVALIDATE(stringtoValidate: string, Password: string): boolean

Compares the not encrypted string "StringtoValidate" with an encrypted password field (e.g. USERS.SOPASSWORD) and returns TRUE if they match.

PLAY (SoundFileName: string): boolean

Plays the sound file "SoundFileName" and returns True.

Example

```
X.PLAY("C:\\SOUND\\alarm.mp3");
```

SHOWOBJFORM: integer

Opens an object, waits for the user to save it and then returns the id of the new record. It must always be combined with function CreateObjForm.

Example

Create purchase document from items

```
function CreatePurchaseDoc() {
    var myObj = X.CreateObjForm('PURDOC');
    try {
        myObj.DBINSERT;
        var tblFINDOC = myObj.FindTable('FINDOC');
        var tblITELINES = myObj.FindTable('ITELINES');
        tblFINDOC.Edit;
        tblFINDOC.SERIES = 2001;
        tblFINDOC.TRDR = 123456;
        tblITELINES.Append;
        tblITELINES.MTRL = ITEM.MTRL;
        tblITELINES.QTY1 = 100;
        tblITELINES.PRICE = 50;
        tblITELINES.Post;
        var id = myObj.SHOWOBJFORM();
        if (id > 0)
            X.WARNING("New id is:" + id);
    }
    catch (e) {
        if (myObj != null)
            X.WARNING("General Error: " + e.message + "\nObject Error: " + myObj.GETLASTERROR);
        else
            X.WARNING("General Error: " + e.message);
    }
    finally {
    }
}
```

SPELL (Num: Real): string

Returns the spelling of number Num

Example

```
X.WARNING(X.SPELL(1101.10));  
//Returns: ONE THOUSAND ONE HUNDRED ONE AND TEN
```

SQL (ASQL: string; AParams: Variant): string

Executes the ASQL statement and returns **one row result** (comma separated).

ASQL: SQL statement

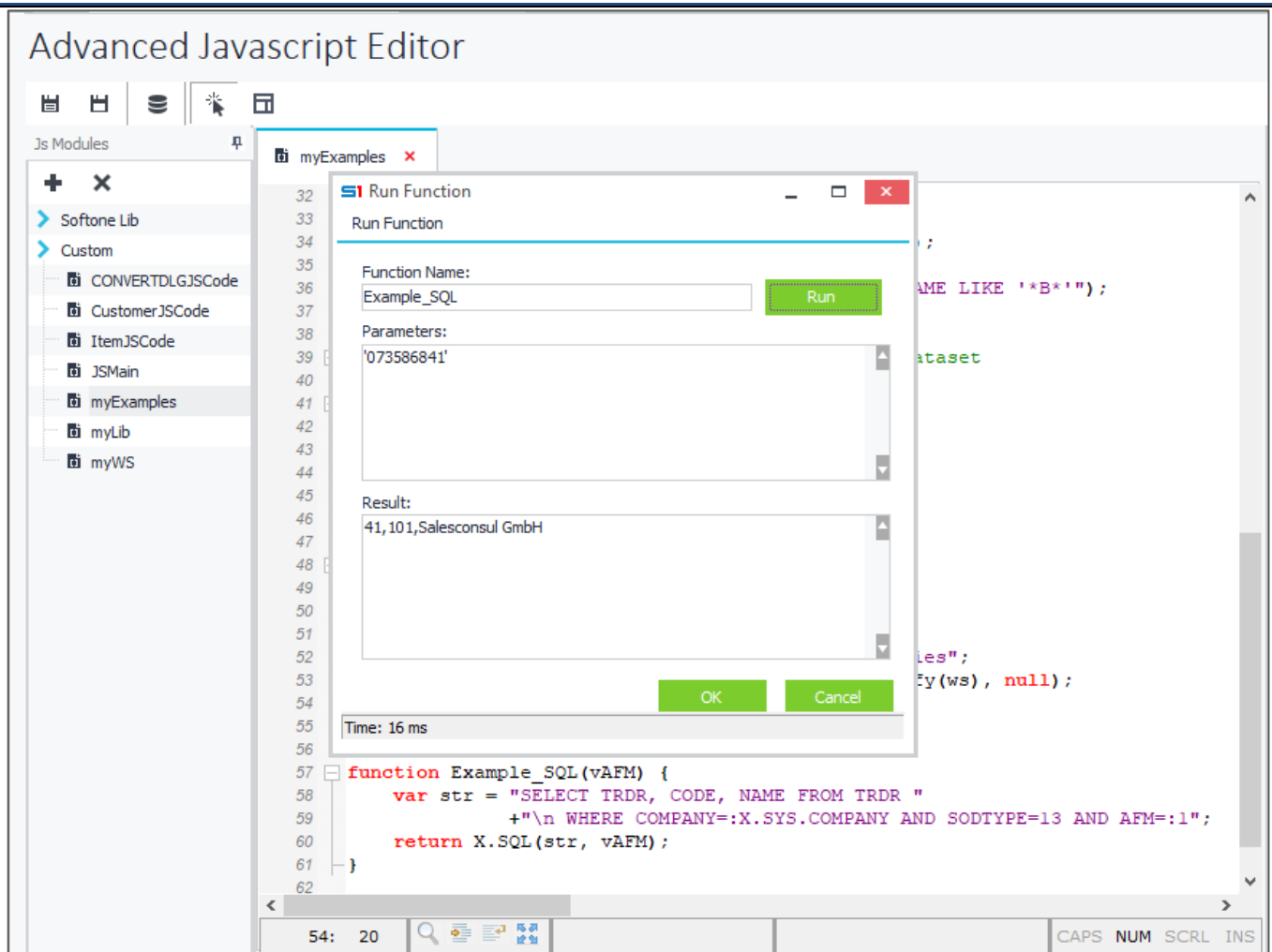
AParams: Comma separated list containing query parameters

Example 1

```
var NewCode = X.SQL("SELECT ISNULL((SELECT MAX(ISNULL(TRY_PARSE(ISNULL(SNCODE,0) AS  
INT),0)) FROM GUARANTY),0) + 1", null);  
//Returns the max number + 1 of the field Guaranty.SNCODE
```

Example 2

```
function ExampleSQL(vAFM) {  
    var str = "SELECT TRDR, CODE, NAME FROM TRDR "  
        + "\n WHERE COMPANY=:X.SYS.COMPANY AND SODTYPE=13 AND AFM=:1";  
    return X.SQL(str, vAFM);  
}  
//Return Example: 1, 1234, TestCustomer
```



X.SQL – Example in Advanced JavaScript

STRINGS (StringList: string, Key: string): string

Returns the value of the given StringList key. If no key-value pair is found, then an empty string is returned.

StringList: String List name (e.g. SOSOURCE, SODTYPE)

Key: List key of the lookup record

Examples

```
X.STRINGS("SODTYPE", "96"); //returns: Actions
X.STRINGS("Fprms", 1351); //returns: SALFPRMS
X.STRINGS("MonthNames", 6); //returns: June
X.STRINGS("ShortMonthNames", 8); //returns: Aug
X.STRINGS("CstType", 12); //returns: Data Flow Rules
X.STRINGS("Editors", "UT1"); //returns: UTBL01 (F[SODTYPE=:CSEL.SODTYPE])
X.STRINGS("PayDay", "0"); //returns: Sunday
```

SUM (FieldName: string): real

Summarizes the data of the field "FieldName" for all the rows of the given datagrid.

It's recommended to use the function X.FILTERSUM with parameter Filter '1=1'.

Example

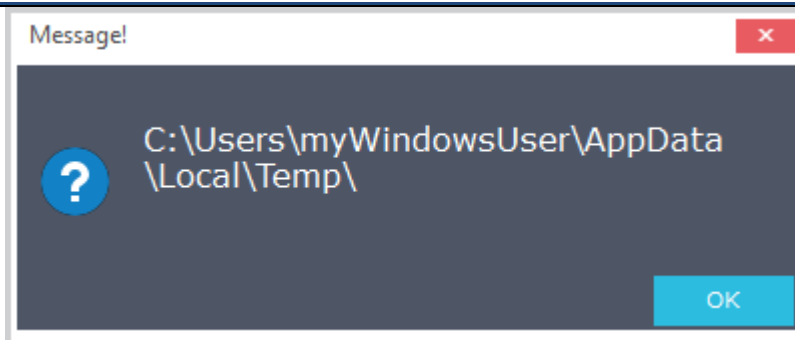
```
X.SUM('ITELINES.NUM01');
//Summarizes the data of the field ITELINES.NUM01
```

TEMPDIR: string

Returns the path of the windows user's temporary folder.

Example

```
X.WARNING(X.TEMPDIR);
```



TIME: string

Returns the current time in hh:mm format

Example

```
X.WARNING("Current time is: " + X.TIME);
```

USERVALIDATE(UserName: string, Password: string): boolean

Returns true if a user with credentials "UserName" and "Password" is found in the database.

WEBREQUEST (JSONRequest: string): string

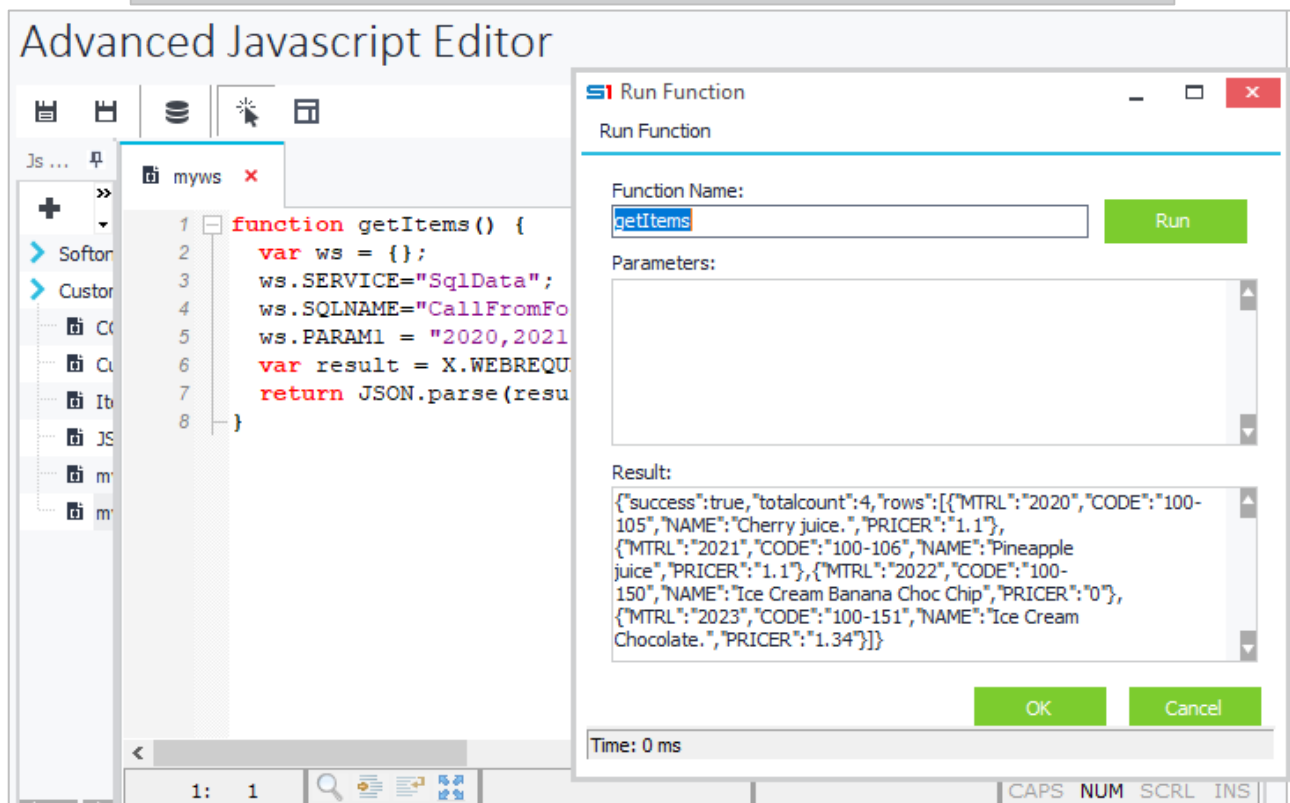
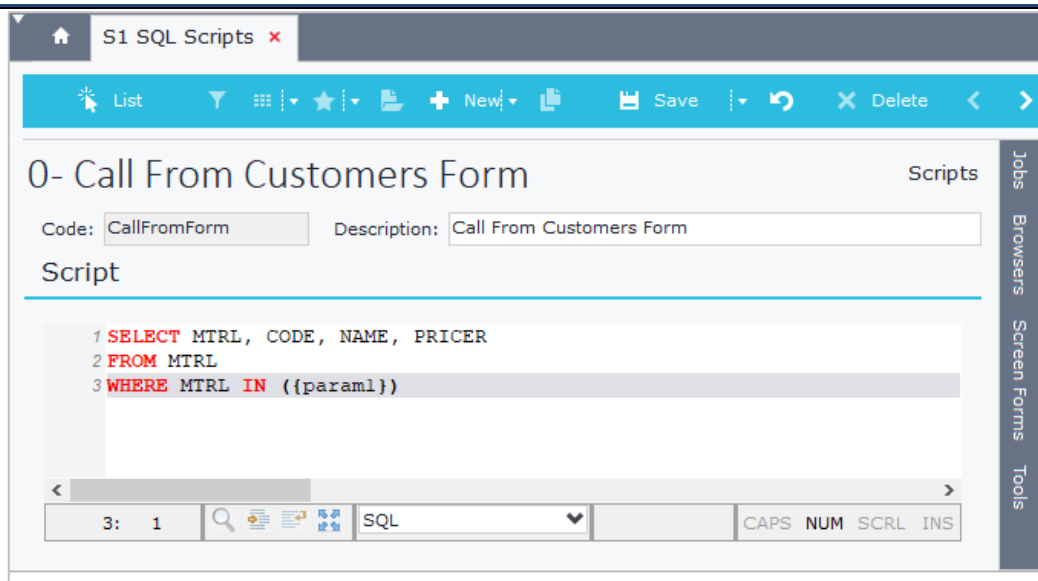
Executes a SoftOne web service request that uses built-in services (SetData, GetData, GetBrowserInfo etc.) and returns the data in JSON format. For custom web services (created in Advanced JavaScript) use the function WSCALL.

It can be used either for external or internal requests (e.g. Add new customer using SETDATA).

Example 1

Function that makes a web request to "S1 SQL Scripts" (SERVICE = "SqlData"), locates a specific record (SQLNAME = "CallFromForm") and returns the data retrieved from the SQL statement.

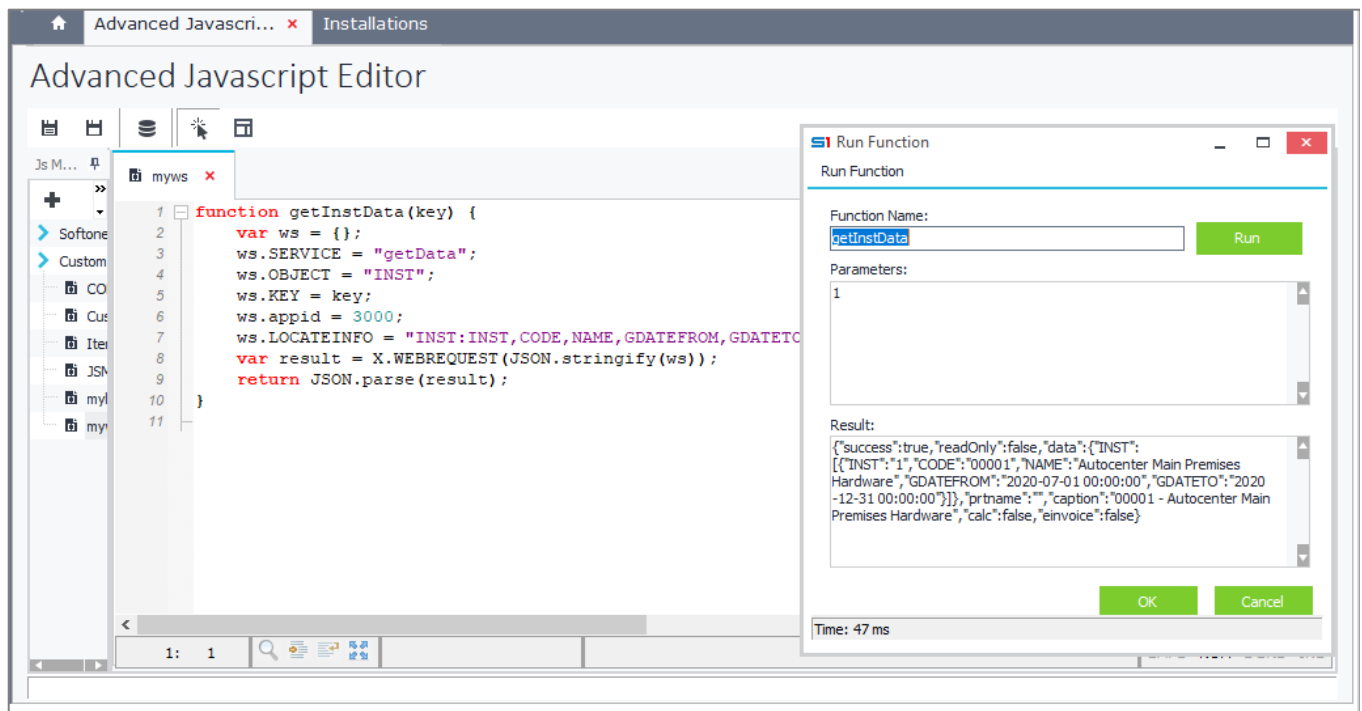
```
function getItems() {  
    var ws = {};  
    ws.SERVICE = "SqlData";  
    ws.SQLNAME = "CallFromForm";  
    ws.PARAM1 = "2020,2021,2022,2023";  
    var result = X.WEBREQUEST(JSON.stringify(ws));  
    return JSON.parse(result);  
}
```



Example 2

Return data of INST (specific columns) using the service "getData".

```
function getInstData(key) {  
    var ws = {};  
    ws.SERVICE = "getData";  
    ws.OBJECT = "INST";  
    ws.KEY = key;  
    ws.appid = 3000;  
    ws.LOCATEINFO = "INST:INST, CODE, NAME, GDATEFROM, GDATETO"; //Return fields  
  
    var result = X.WEBREQUEST(JSON.stringify(ws));  
  
    return JSON.parse(result);  
}
```



Example 3

Update customer data using the service call setData.

```
function updateCustomerData(cusdata) {  
    var ws = {};  
    ws.SERVICE = "setData";  
    ws.OBJECT = "CUSTOMER";  
    ws.appid = 3004;  
    ws.key = cusdata.trdr;  
    ws.DATA = {};  
    ws.DATA.CUSTOMER = [];  
  
    var mycus = {};  
    mycus.payment = cusdata.payment;  
    mycus.email = cusdata.email;  
  
    ws.DATA.CUSTOMER.push(mycus);  
    X.LOG("updateCustomerData: " + JSON.stringify(ws));  
  
    var result = X.WEBREQUEST(JSON.stringify(ws));  
    return JSON.parse(result);  
}
```

WSCALL (scope: variant, Uri: string, postData: string, callbackFunc: variant): string

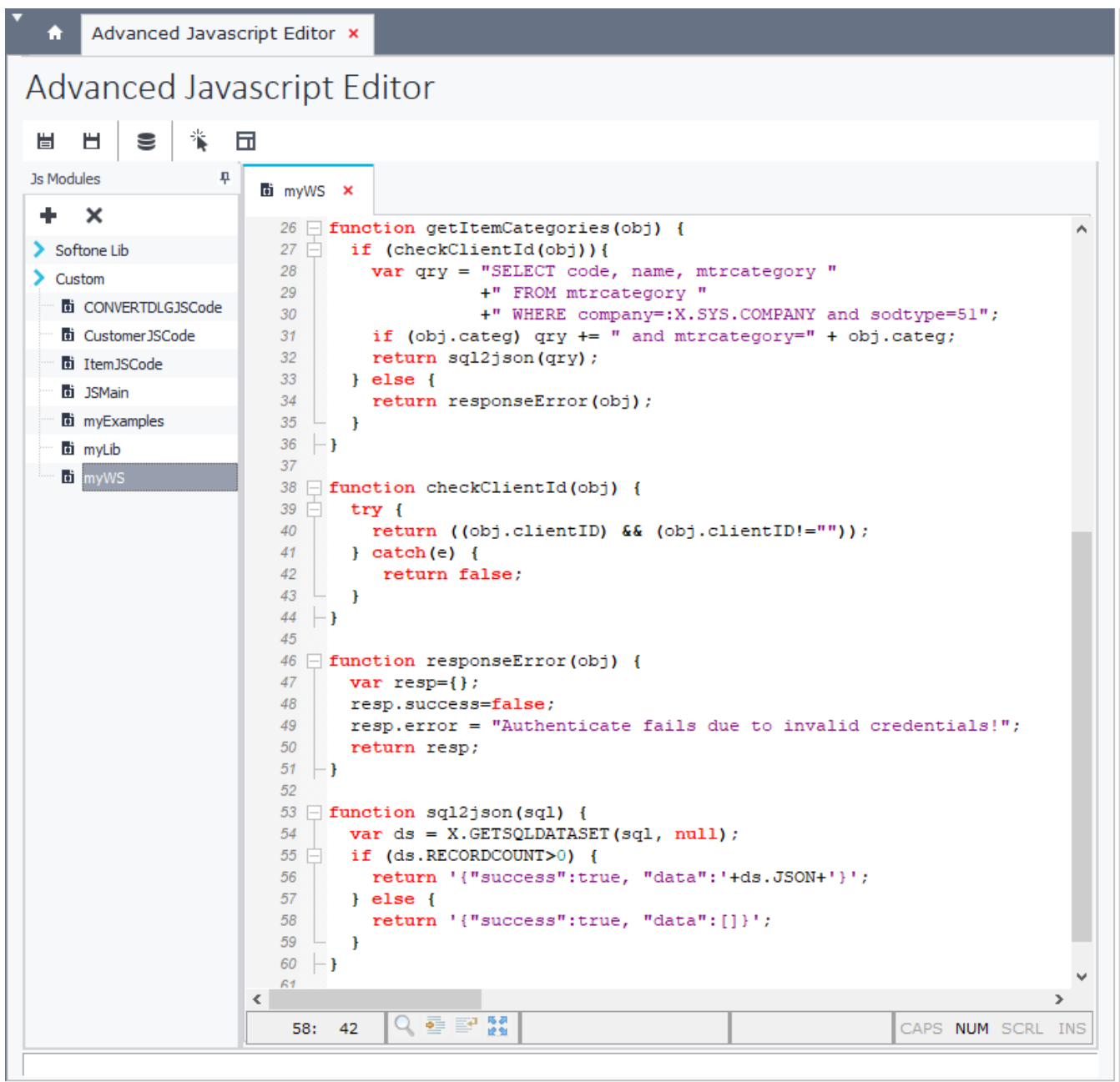
Executes any SoftOne web service request (built-in or custom) and returns the data in JSON format.

For built-in requests leave "Uri" param null, while for custom ones enter the URL part of the custom webservice created in Advanced JavaScript (e.g. "/s1services/JS/mywebcall").

Example 1

Function that makes a web request to Advanced Javascript function "getItemCategories".

```
function CustomWSRequest() {  
    var ws = {};  
    ws.clientID = "xxx";  
    ws.categ = 100;  
    var url = "/s1services/JS/myWS/getItemCategories";  
    var result = X.WSCALL(null, url, JSON.stringify(ws), null);  
    return JSON.parse(result);  
}
```



D. Dataset Methods

This module demonstrates all the Dataset methods you can use in form scripts.

APPEND

Opens a new, empty record at the end of the dataset and makes the empty record the current one so that field values for the record can be entered either by a user or by code.

Example

Function that inserts a record in document item lines for the item with code=200

```
function AppendNewRecord() {
    ITELINES.APPEND;
    ITELINES.MTRL = X.ID('ITEM','200');
    ITELINES.QTY1 = 1;
    ITELINES.LINEVAL = 100;
    ITELINES.POST;
}
```

DELETE

Deletes the current record and puts the Dataset in dsBrowse state. The record that followed the deleted record becomes the current record.

Example

Function that deletes a record in document item lines when the item code=100

```
function DeleteRecord() {
    if (ITELINES.LOCATE('X_CODE',100) == 1)
        ITELINES.DELETE;
}
```

DISABLECONTROLS

Disables the windows controls of the specified dataset.

It can be used in master or detail tables of an object and it increases the performance, especially when iterating datagrids.

Attention

It cannot be used in secondary detail tables.

Always use it in a try/finally block in order to avoid accidental disable of Dataset controls.

Example 1

Loop through document item lines datagrid by deactivating Window Controls

```
function LoopDataset() {
    ITELINES.DISABLECONTROLS;
    try {
        ITELINES.FIRST;
        while (!ITELINES.EOF) {
            //Add Code Here
            ITELINES.NEXT;
        }
    }
    finally {
        ITELINES.ENABLECONTROLS;
    }
}
```

ENABLECONTROLS

Activates the window controls of a given Dataset. Example can be found in “[DisableControls](#)” function.

EDIT

Puts the Dataset into dsEdit state if it is not already in dsEdit or dsInsert states

Example

```
myDataset.EDIT;
```

FIRST

Moves to the first record of the Dataset

Example

Fill the field NUM01 in all document item lines datagrid with a specific value.

```
function FillDatasetRecords() {  
    ITELINES.DISABLECONTROLS;  
    try {  
        ITELINES.FIRST;  
        while(!ITELINES.EOF) {  
            ITELINES.NUM01 = ITELINES.LINEVAL * SALDOC.LRATE;  
            ITELINES.NEXT;  
        }  
    }  
    finally {  
        ITELINES.ENABLECONTROLS;  
    }  
}
```

INSERT

Opens a new, empty record before the current record, and makes the empty record the current one so that field values for the record can be entered either by a user or by code.

Example

Locate the first record of documents lines that has "Item Code"="01-001", then add a new record before this line using: "Item Code"="01-002", "Qty1"=1 and "Lineval"=100.

```
function InsertRecord() {  
    if(ITELINES.LOCATE('X_CODE', '01-001') == 1) {  
        ITELINES.INSERT;  
        ITELINES.MTRL = X.ID('ITEM', '01-002');  
        ITELINES.QTY1 = 1;  
        ITELINES.LINEVAL = 100;  
        ITELINES.POST;  
    }  
}
```

LAST

Moves to the last record of the Dataset

Example

```
ITELINES.LAST;
```

NAME

Returns the name of the current object

Example

```
var objName = X.NAME;  
X.WARNING(objName);
```

NEXT

Moves to the next record of the Dataset

Example

In documents, search for item lines that have discount more than 50% and display a warning message.

```
function FillDatasetRecords() {
    ITELINES.DISABLECONTROLS;
    try {
        ITELINES.FIRST;
        while(!ITELINES.EOF) {
            if (ITELINES.DISC1PRC > 50)
                X.WARNING('Discount is greater than 50% in item code: '+
ITELINES.X_CODE);
            ITELINES.NEXT;
        }
    }
    finally {
        ITELINES.ENABLECONTROLS;
    }
}
```

POST

Attempts to post the new or altered record.

If successful, the Dataset is put in dsBrowse state. If unsuccessful, the Dataset remains in its current state.

Example

Fill document item lines datagrid with the items of the login company that a specific field meets a certain condition (CCCMYFLAG = 1)

```
function AppendNewRecord() {
    ds = X.GETSQLDATASET ('SELECT MTRL FROM MTRL WHERE CCCMYFLAG = 1 AND SODTYPE=51
AND COMPANY='+X.SYS.COMPANY,null);
    if (ds.RECORDCOUNT > 0) {
        ITELINES.APPEND;
        ITELINES.MTRL = ds.MTRL;
        ITELINES.QTY1 = 1;
        ITELINES.LINEVAL = 10;
        ITELINES.POST;
        ds.NEXT;
    }
}
```

PRIOR

Moves to the previous record of the Dataset

Example

```
ITELINES.PRIOR;
```

SETREADONLY (FieldName: string, Value: True/False)

Sets the "FieldName" field in Read Only mode. If you use it in a datagrid then it sets the whole column in read only mode.

Example

Change the status of the field CUSTOMER.TRDCATEGORY to read only.

```
CUSTOMER.SETREADONLY ('TRDCATEGORY', 'TRUE');
//or
CUSTOMER.SETREADONLY ('TRDCATEGORY', 1);
```

SETDATASETLINKS(ModuleHandle, DatasetHandle, Value)

SBSL (ModuleIntf) function that is very useful in form scripts. When you populate a Dataset (e.g. virtual table) that has fields linked to tables, then for every record added, a query runs for each of these fields in order to link them to the tables. This function provides the ability to send the queries once, when all records are filled.

Values: 0= Disable links, 1= Enable links

Syntax: X.EXEC('CODE:ModuleIntf.SetDatasetLinks', X.MODULE, dsHandle, 0 or 1);

Example 1

```
dsHandle = X.EXEC('CODE:ModuleIntf.GetDataset', X.MODULE, 'MyDataset');
X.EXEC('CODE:ModuleIntf.SetDatasetLinks', X.MODULE, dsHandle, 0);
```

Example 2

See → [Case Study 6](#)

SETFIELDPROPERTY(FieldHandle, PropertyIndex, PropertyValue);

SBSL (ModuleIntf) function that is used to modify field properties (Required and Read-only).

PropertyIndex accepts the following values:

5: Required (1=true, 0=false)

6: Read-only (1=true, 0=false)

Example – JavaScript

In conversion dialog object change the "required" property of the field "Delivery Date" (DELIVDATE) to mandatory (using Advanced JS).

```
function ON_CREATE() {
    var ds = X.EXEC('CODE:ModuleIntf.GetDataSet', X.MODULE, 'CONVERTFPRMS');
    var fld = X.EXEC('CODE:ModuleIntf.GetField', ds, 'DELIVDATE');
    X.EXEC('CODE:ModuleIntf.SetFieldProperty', fld, 5, 1);
}
```

GETFIELDPROPERTY(FieldHandle, PropertyIndex);

SBSL (ModuleIntf) function that returns field properties (for the following property indexes).

PropertyIndex accepts the following values:

1: Field Name

2: Data Type

3: Size

4: Display Name

5: Required

6: Read-only

Example

```
function ON_ITEM_CODE() {
    var vProp = GetFieldProp('ITEM', 'CODE', 5);
    X.WARNING("From Adv.JS\nProperty 'Required' of 'Code' is: " + vProp);
}

function GetFieldProp(tableName, fieldName, fieldProperty) {
    //1:FieldName, 2:DataType, 3:Size, 4:DisplayName, 5:Required, 6:ReadOnly
    var ds = X.EXEC('CODE:ModuleIntf.GetDataSet', X.MODULE, tableName);
    var fld = X.EXEC('CODE:ModuleIntf.GetField', ds, fieldName);
    return X.EXEC('CODE:ModuleIntf.GetFieldProperty', fld, fieldProperty);
}
```

Advanced Javascript Editor

The screenshot shows the 'Advanced Javascript Editor' interface. On the left, the 'Js Modules' pane lists several modules: Softone Lib, Custom, CONVERTDLGJSCode, CustomerJSCode, ItemJSCode, JSMain (highlighted with a red box), myExamples, myLib, myWS, S1WSEExamples, and S1WSLib. A red arrow points from the JSMain module in the list to the main editor. The main editor displays the code for JSMain, with the first line `AddCode('ITEM', 'ItemJSCode');` highlighted by a red box.

```

1 AddCode('ITEM', 'ItemJSCode');
2 AddCode('CUSTOMER', 'CustomerJSCode');
3 AddCode('CONVERTDLG', 'CONVERTDLGJSCode'); //Document Conversions

```

Advanced Javascript Editor

The screenshot shows the 'Advanced Javascript Editor' interface. On the left, the 'Js Modules' pane lists several modules: Softone Lib, Custom, CONVERTDLGJSCode, CustomerJSCode, ItemJSCode (highlighted with a red box), JSMain, myExamples, myLib, myWS, S1WSEExamples, and S1WSLib. A red arrow points from the ItemJSCode module in the list to the main editor. The main editor displays the code for ItemJSCode, with the function `ON_ITEM_CODE()` highlighted by a red box.

```

1 function ON_ITEM_CODE() {
2     var vProp = GetFieldProp('ITEM', 'CODE', 5);
3     X.WARNING("From Adv.JS\nProperty Required for field Code is "+vProp);
4 }
5
6 function GetFieldProp(tableName, fieldName, fieldProperty) {
7     //1:FieldName, 2:DataType, 3:Size, 4:DisplayName, 5:Required, 6:ReadOnly
8     var ds = X.EXEC('CODE:ModuleIntf.GetDataSet', X.MODULE, tableName);
9     var fld = X.EXEC('CODE:ModuleIntf.GetField', ds, fieldName);
10    return X.EXEC('CODE:ModuleIntf.GetFieldProperty', fld, fieldProperty);
11 }
12
13

```

The screenshot shows the 'Advanced JS - GetFieldProperty' dialog box. The 'Code' field is set to '100-1001' and the 'Description' is 'Orange juice'. The 'General data' tab is selected, showing fields like 'Base UoM', 'VAT Group', 'Accounting category', 'BarCode', 'Factory code', and 'Technical code'. The 'GROUPING/ CLASSIFICATION' section shows a tree structure with 'Commercial category' (204) and 'Business unit' (108). A message box is displayed in the foreground, stating: 'From Adv.JS Property Required for field Code is true'. The message box has a question mark icon and an 'OK' button.

Figure – Advanced JS - GetFieldProperty

E. Dataset Functions

This module demonstrates all the object functions you can use in form scripts.

ACTIVE: boolean

Returns TRUE or 1 when the data of a dataset can be read or edited.

BOF: boolean

Returns TRUE or 1 if the Dataset is on the first record

Example

```
ITELINES.LAST;
while(!ITELINES.BOF) {
    //Add Code Here
    ITELINES.PRIOR;
}
```

EOF: boolean

Returns TRUE or 1 if the Dataset is on the last record

Example

```
SRVLINES.FIRST;
while(!SRVLINES.EOF) {
    //Add Code Here
    SRVLINES.NEXT;
}
```

FIELDBYNAME (FieldName: string): variant

Returns the value of field "FieldName"

Example

```
for(i = 1; i <= 3; i++) {
    myfield = SALDOC.FIELDBYNAME('CCCMYFIELD'+i);
    if (myfield == 'Value') {
        //Add Code Here
    }
}
```

FIELDCOUNT: integer

Returns the number of fields in the Dataset

Example

```
for (i = 0; i < SALDOC.FIELDCOUNT - 1; i++) {
    //Add Code Here
}
```

FIELDNAME (index: integer): string

Returns the name of the index field

Example

Lock all fields of Installations module (INST) except for field REMARKS. Fields will be read-only for all users except for Administrator group of users.

```
function ON_INSERT() {
    LockRecord(0); //Unlock fields on insert
}

function ON_LOCATE() {
    if (X.SYS.GROUPS != 100) { //Group Administrator
        LockRecord(1); //Lock fields on locate
    }
}

function LockRecord(vReadonly) {
    for (i = 0; i <= INST.FIELDSCOUNT - 1; i++) {
        if (INST.FIELDNAME(i) == 'REMARKS') { //always editable
            X.SETPROPERTY('FIELD', 'INST.' + INST.FIELDNAME(i), 'READONLY', 0);
        } else {
            X.SETPROPERTY('FIELD', 'INST.' + INST.FIELDNAME(i), 'READONLY',
vReadonly);
        }
    }
    for (i = 0; i <= INSTLINES.FIELDSCOUNT - 1; i++) {
        X.SETPROPERTY('FIELD', 'INSTLINES.' + INSTLINES.FIELDNAME(i), 'READONLY',
vReadonly);
    }
}
```

FIELDTYPE (FieldName: string): integer

Returns a number that applies to the type of the field. Values returned are:

1=String, **2**=Smallint, **3**=Integer, **4**=Word, **6**=Float, **11**=DateTime, **16**=Memo

Example

```
MTRSUBSTITUTE.FIELDTYPE('NAME'); //Returns 1 (String)
```

FIELDS (index: integer): variant

Returns the value of the index field

FILTER: string

String used for filtering the Dataset

Example 1

```
vFilter = '{ITELINES.MTRL_ITEM_CODE}='
'+String.fromCharCode(39)+'12345'+String.fromCharCode(39);
ITELINES.FILTER= '('+vFilter+')';
ITELINES.FILTERED= 1;
```

Example 2

See → [Case Study 5](#)

FILTERED: boolean

Enables(1) - Disables(0) filtering and returns the filtered status of the Dataset

Example 1

```
vFilter = '{ITELINES.MTRL_ITEM_NAME}=
'+String.fromCharCode(39)+'MyItem'+String.fromCharCode(39);
ITELINES.FILTER= '('+vFilter+')';
ITELINES.FILTERED= 1;
```

Example 2

See → [Case Study 5](#)

GETGRAPH (LabelFieldName: string; ValueFieldName: string): string

Creates a Chart Image in bmp format and returns the full path and filename. The chart is created with data from the field "LabelFieldName" and values from the field "ValueFieldName"

Example

Function that creates a bmp file and saves the path and filename in the REMARKS field of the located custom table (CCCMYTABLE) record.

```
function Createbmpfile() {
    str = "SELECT TOP 5 A.NAME, ISNULL(B.LBAL,0) AS BAL FROM TRDR A INNER JOIN
TRDFINDATA B ON A.TRDR=B.TRDR WHERE A.COMPANY="+X.SYS.COMPANY+" AND A.SODTYPE=13
ORDER BY 2 DESC";
    ds = X.GETSQLDATASET(str,"");
    CCCMYTABLE.REMARKS = ds.GETGRAPH('NAME','BAL'); //Creates the bmp file and
returns the full path
}
```

GETHTML (FieldNames: string): string

Returns the values of "FieldNames" fields in HTML format

Example

Function that returns the fields "Code" and "Name" of table TRDR (with ids 123 and 124) in html format.

```
function GETHTMLFormat() {
    var ds = X.GETSQLDATASET('SELECT CODE, NAME FROM TRDR WHERE TRDR IN
(123,124)', '');
    var myhtml = ds.GETHTML('CODE;NAME');
    return myhtml;
}
//Returns
<table id="seltbl" width="10" count="1">
<tbody>
    <tr>
        <td>09</td>
        <td>Test Customer 1</td>
    </tr>
    <tr>
        <td>10</td>
        <td>Test Customer 2</td>
    </tr>
</tbody>
</table>
```

GETXML (WriteMetadata: Boolean, DisplayTagsOnNullValues: Boolean): string

Returns the Dataset in XML format

Example

Function that returns the fields "Code" and "Name" of the selected customer in sales documents in xml format.

```
function GetXMLFormat(withmetadata) {  
    var ds = X.GETSQLDATASET("SELECT CODE, NAME FROM TRDR WHERE TRDR=:1",  
        SALDOC.TRDR);  
    return ds.GETXML(withmetadata);  
}
```

//0 (No MetaData) Returns

```
<?xml version="1.0" encoding="ISO-8859-7"? >  
<SODATA>  
    <ROWDATA><ROW CODE="09" NAME="Test Customer 1"/>  
    </ROWDATA>  
</SODATA>
```

//1 (With MetaData) Returns

```
<?xml version="1.0" encoding="ISO-8859-7"? >  
<SODATA>  
    <METADATA>  
        <FIELDS>  
            <FIELD fieldname="CODE" fieldtype="string" WIDTH="25" fieldcaption=""/>  
            <FIELD fieldname="NAME" fieldtype="string" WIDTH="64" fieldcaption=""/>  
        </FIELDS>  
        <PARAMS PRIMARY_KEY=""/>  
    </METADATA>  
    <ROWDATA>  
        <ROW CODE="09" NAME="Test Customer 1"/>  
    </ROWDATA>  
</SODATA>
```

ISNULL (FieldName: string): boolean

Returns True or 1 when field "FieldName" is null

Example

When saving a customer record check if a field(ZIP) is null and display a message

```
function ON_POST() {  
    CheckNull('ZIP');  
}  
  
function CheckNull(fldn) {  
    if (CUSTOMER.ISNULL(fldn) == 1)  
        X.WARNING(fldn + " IS NULL!");  
    else  
        X.WARNING(fldn + " IS NOT NULL!");  
}
```

JSON: string

Returns the columns and the data of a table or dataset in JSON format.

Example 1

```
function MyGetJson() {  
    var vjson;  
    vjson = MTRSUBSTITUTE.JSON;  
    X.WARNING(vjson);  
}
```

Example 2

See → [Advanced Javascript - Debug Functions](#)

LOCATE (KeyFields: string; KeyValues: variant): boolean

Searches the dataset for a specific record and returns TRUE if found. It also makes it the active record.

KeyFields are separated with semicolons while **KeyValues** with commas.

Example 1

In Sales Documents entity, try to locate a line that has an item with id 12345 (MTRL=12345) and the quantity of the line is 10 (QTY1=10). If a line meets the above filters then display a message.

```
if(ITEMINES.LOCATE('MTRL;QTY1',12345,10) == 1)  
    X.WARNING("Located line...");  
else  
    X.WARNING("Record not found");
```

Example 2

Locate a memory table record.

```
if(X.PAYMENT.LOCATE('CODE',1001) == 1)  
    X.WARNING("Located Payment record...");  
else  
    X.WARNING("Record not found");
```

LOOKUP (KeyFields: string; KeyValues: Variant; ResultFields: string): Variant;

Returns field values from a record that matches specified search values, without the use of SQL queries. **KeyFields** are separated with semicolons while **KeyValues** with commas. Note also that this function is used in memory tables.

Example 1

Return the email field of the login user.

```
var vUserCode = X.SYS.USER; //Login User Code  
var vUserEmail = X.USERS.LOOKUP('USERS', vUserCode,'EMAIL'); //Returns e-mail field
```

Example 2

Return the description (name field) of payment with Company =1000, Sodtype =13 and Code =1000.

```
var vPaymentDescr = X.PAYMENT.LOOKUP('COMPANY;SODTYPE;PAYMENT',1000,13,1000  
, 'NAME');  
if (vPaymentDescr != null)  
    X.WARNING(vPaymentDescr);
```

RECNO: integer

Returns the current record number of the Dataset.

Example

```
var curRec = ITEMINES.RECNO;
```

RECORDCOUNT: integer

Returns the total number of records in the Dataset

Example 1

```
if (ITELINES.RECORDCOUNT > 0) {  
    //Add Code Here  
}
```

Example 2

See → [Case Study 2](#)

STATE: integer

Returns the State of the Dataset. The available values are summarized below:

0=dsInactive

Dataset closed. Data unavailable.

1=dsBrowse

Dataset open. Data can be viewed, but not changed. This is the default state of an open Dataset.

2=dsEdit

Dataset open. The current row can be modified.

3=dsInsert

Dataset open. A new row is inserted or appended.

6=dsFilter

Dataset open. Indicates that a filter operation is under way. A restricted set of data can be viewed, and no data can be changed.

SUM (FieldName: string, Filter: string): real

Summarizes the data of the "FieldName" field of the given datagrid for entries that meet the "Filter" expression.

Use filter "1=1" for not applying any filters.

Examples

```
var vQuantA = ITELINES.SUM("QTY1", "MTRUNIT=1");  
var vQuantB = ITELINES.SUM("QTY1", "X_CODE LIKE 'A*'");  
var vQuantC = ITELINES.SUM("QTY1", "1=1");
```

F. Field Events

These events fire when users alter the data of fields and are summarized below.

ON_ObjectName_FieldName_VALIDATE

Fires on field value change (Before Change).

Its main use is to validate data and prevent field value change.

Example

Module: Sales Documents

Job: Prevent users from modifying customer in sales documents (series 7021) that have been posted.

```
function ON_SALDOC_TRDR_VALIDATE() {
    if ((SALDOC.SERIES == 7021) && (SALDOC.FINDOC > 0)) {
        X.EXCEPTION((X.SYS.USERLANG == "EL" ? "Απαγορεύεται η αλλαγή πελάτη, το
        παραστατικό είναι καταχωρημένο!" :
        (X.SYS.USERLANG == "SR" ? "Не можете да промените клијента, документ је
        регистрован" : "Altering Customer is not permitted, the document is posted!")));
    }
}
```

ON_ObjectName_FieldName

Fires on field value change (After Change).

Example 1

Module: Sales Documents

Job: Show/Hide a panel based on Series value and the Login User Code

```
function ON_SALDOC_SERIES() {
    if (((SALDOC.SERIES == 7650) || (SALDOC.SERIES == 7630)) && (X.SYS.USER == 111))
        X.SETPROPERTY('PANEL', 'PANEL14', 'VISIBLE', 'TRUE');
    else
        X.SETPROPERTY('PANEL', 'PANEL14', 'VISIBLE', 'FALSE');
}
```

Example 2

Module: Installations

Job: Alter the data of field "DateTo" based on the data of field "DateFrom"

```
function ON_INST_WDATEFROM() {
    if (INST.ISNULL('WDATEFROM') == 1)
        INST.WDATETO = null;
    else
        INST.WDATETO = X.EVAL(' INCYEAR (INST.WDATEFROM, 1) ');
}
```

G. Datagrid Events

The events that are triggered when users alter datagrids are summarized below.

ON_DatagridName_NEW

Triggered when a new record is added in a datagrid.

Usually used for setting default values.

Example

Module: Sales Documents

Job: Add default values in customer branches datagrid when users insert a new row (only for the 1st one).

```
function ON_CUSBRANCH_NEW() {  
    if(CUSBRANCH.RECORDCOUNT == 0) {  
        CUSBRANCH.CODE = "0000";  
        CUSBRANCH.NAME = CUSTOMER.NAME;  
    }  
}
```

ON_DatagridName_POST

Fires on datagrid before-post, before the changes to the record are posted.

Use it to perform any actions that you want to occur before a datagrid record is posted. It is commonly used to change values, to perform validation and to raise custom error messages.

Example

Module: Sales Documents

Job: Display a message that prevents the user from adding item lines using discount greater than 50%.

```
function ON_ITELINES_POST() {  
    if(ITELINES.DISC1PRC > 50)  
        X.EXCEPTION("Discount can not be greater than 50%.");  
}
```

ON_DatagridName_AFTERPOST

Fires on datagrid after-post, after the changes to a datagrid record have been posted.

Usually used to display / send a message or for calculating purposes in order to update values of other tables.

Example

Module: Sales Documents

Job: Summarize the data of a custom item lines field and display the result in a header field when users post Item Lines record. Use also custom filters and display the filtered sum in a different header field.

```
function ON_ITELINES_POST() {  
    CalculateLinesData();  
}  
  
function CalculateLinesData() {  
    var vTotLinesCCCValue = X.FILTERSUM('ITELINES.CCCLINEVAL', '1=1');  
    SALDOC.CCCTOTCCCVAL = vTotLinesCCCValue;  
  
    var vTotFilteredValue = X.FILTERSUM('ITELINES.CCCLINEVAL', 'MTRCATEGORY=1 AND  
CCCLINESFIELD=1 AND CCCMYCOMPANY=' + X.SYS.COMPANY);  
    SALDOC.CCCTOTFILTVAL = vTotFilteredValue;  
}
```

ON_DatagridName_BEFOREDELETE

Fires on datagrid before-delete, before a datagrid record is deleted.

Use it to perform validation or for raising custom error messages.

Example

Module: Sales Documents

Job: Prevent users from deleting an item line based on certain criteria.

```
function ON_ITELINES_BEFOREDELETE() {  
    if (ITELINES.CCCS1UPD == 1)  
        X.EXCEPTION('This item line cannot be deleted.');
```

ON_DatagridName_AFTERDELETE

Fires on datagrid after-delete, after a datagrid record is deleted.

Use it to raise custom messages or perform any update actions to other tables.

Example

Module: Sales Documents

Job: When users delete Item Lines, summarize the data of a field in lines and display the result in a header field. Use also custom filters and display the filtered sum in a different header field.

```
function ON_ITELINES_AFTERDELETE() {  
    CalculateLinesData();  
}  
  
function CalculateLinesData() {  
    var vTotLinesCCCValue = X.FILTERSUM('ITELINES.CCCLINEVAL', '1=1');  
    SALDOC.CCCTOTCCCVAL = vTotLinesCCCValue;  
  
    var vTotFilteredValue = X.FILTERSUM('ITELINES.CCCLINEVAL', 'MTRCATEGORY=1 AND  
CCCLINESFIELD=1 AND CCCMYCOMPANY=' + X.SYS.COMPANY);  
    SALDOC.CCCTOTFILTVAL = vTotFilteredValue;  
}
```

H. Object Events

All the events that fire on SoftOne objects (modules) are discussed in the following pages.

ON_CREATE

Fires when an object is created (when you open it from the menu).

Usually used for adding params, initializing variables or altering object string lists (like Browser Context Menu).

Example 1

Module: Sales Documents

Job: In new entries, auto open a QuickView from Items that retrieves data from line Item.

```
var vOpenOnce = 0;

function ON_CREATE() {
    vOpenOnce = 1;
}

function ON_INSERT() {
    if (vOpenOnce == 1) { //Check to open quickview only one time
        X.QuickView('ITEM', 'ItemBalanceQV', 'ITELINES.MTRL');
        vOpenOnce = 0;
    }
}
```

Example 2

Module: Sales Documents

Job: Add right click menu jobs under a custom folder "MyJobs": "Customer Financial data" and "NewJob"

```
function ON_CREATE() {
    ChangeBrowserMenu(); //Change context menu of browser
}

function ChangeBrowserMenu() {
    if ((X.SYS.GROUPS == 117) || (X.SYS.GROUPS == 201)) {
        //StringList of most SoftOne Objects Browser is BRMENU
        var vBrowserMenu = X.EXEC('CODE:ModuleIntf.FindXStrings', X.MODULE, 'BRMENU');
        X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '--'); //divider
        X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201809191=1;Customer Financial
Data?My Jobs'); //1; displays the job when one or more rows are selected.
        X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201909251=1;NewJob A?My Jobs');
        X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201909252=1;NewJob B?My Jobs');
        //?Name displays the job under a sub-menu/folder (e.g. My Jobs)
        X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201910023=3;Displayed on Lines
Analysis Browser');
        X.EXEC('CODE:SysRequest.RefreshPopupMenu', X.MODULE, 'BRMENU', 1);
    }
}

function EXECCOMMAND(cmd) {
    if (cmd == 201809191) { //Browser right click - Customer Financial Data
        var vSelRecs;
        vSelRecs = X.GETPARAM('SELRECS');
        vSelRecs = vSelRecs.replace(/\?/g, ",");
        var vds = X.GETSQLDATASET('SELECT DISTINCT TRDR FROM FINDOC WHERE ' +
vSelRecs, null);
        if (vds.RECORDCOUNT == 1)
            X.EXEC('XCMD:CUSTOMER[FORM=Financial Data,AUTOLOCATE=' + vds.TRDR + ']');
    }
    if (cmd == 201909251) {
        //Add code here
    }
}
```

ON_FORMLOAD

Fires on form load (after the event ON_CREATE).

ON_DESTROY

Fires on destroy of an object.

ON_CANCEL

Fires before cancelling the record changes inside an object

ON_LOCATE

Fires after locating a record of an object.

Usually used for initiating variables, displaying messages or changing the layout of the form (show/hide controls).

Example 1

Module: Sales Documents

Job: Show/hide panels when certain conditions are met.

```
function ON_LOCATE() {
    CheckVisiblePanels();
}

function CheckVisiblePanels() {
    if ((SALDOC.FINDOC > 0) && (SALDOC.ISCANCEL == 0)) {
        if (SALDOC.CCCAPPRVSTAT == 0) {
            X.SETPROPERTY('PANEL', 'CCCPanelApproveButtons', 'VISIBLE', 'TRUE');
            X.SETPROPERTY('PANEL', 'CCCPanelBtnSendApprv', 'VISIBLE', 'TRUE');
        } else {
            X.SETPROPERTY('PANEL', 'CCCPanelApproveButtons', 'VISIBLE', 'FALSE');
            X.SETPROPERTY('PANEL', 'CCCPanelBtnSendApprv', 'VISIBLE', 'FALSE');
        }
    }
}
```

Example 2

Module: Sales Documents

Job: Disable record editing (make all fields read-only) when certain conditions are met.

```
function ON_INSERT() {
    X.SetProperty("EDITOPTIONS", "READONLY=FALSE"); //Unlocks Editing
}

function ON_CANCEL() {
    X.SetProperty("EDITOPTIONS", "READONLY=FALSE"); //Unlocks Editing
}

function ON_LOCATE() {
    CheckLockRecord();
}

function CheckLockRecord() {
    if ((SALDOC.ISPRINT == 1) && (X.SYS.ISADMIN != 1))
        X.SetProperty("EDITOPTIONS", "READONLY=FALSE"); //Unlocks Editing
    else
        X.SetProperty("EDITOPTIONS", "READONLY=TRUE"); //Locks Editing
}
```

ON_POST

Fires before the changes of a record are posted to the database.

For example, it might be used to perform validity checks on data changes before posting them to the database. Posting is aborted when X.EXCEPTION function is used.

ON_AFTERPOST

Fires after the record is saved in the database.

For example, it fires when users edit / change the contents of a record and then click on "Save" button.

ON_DELETE

Fires before deleting a record of an object.

Usually used for preventing users from deleting a record, because it fires before delete is committed.

Delete of record is cancelled with the use of X.EXCEPTION function.

ON_AFTERDELETE

Fires after deleting a record of an object.

Usually used for performing actions (send messages / add log data) when users delete a record, because it is triggered after the changes take effect.

ON_INSERT

Fires when a new record of an object is created, **not saved** (Click on toolbar button "New")

Usually used for setting default values, changing field editors, hiding panels, etc.

Example

Module: Purchase Documents

Job: Change the editor of item lines (field MTRL) according to Project field data.

```
function ON_INSERT() {
    SetNewEditors();
}

function ON_LOCATE() {
    SetNewEditors();
}

function ON_PURDOC_PRJC() {
    SetNewEditors();
}

function SetNewEditors() {
    if (PURDOC.ISNULL('PRJC') == 0) //not null
        X.SETFIELDEDITOR('ITELINES.MTRL', 'ITEM(W[EXISTS(SELECT 1 FROM CCCMYTABLE A1
WHERE A1.MTRL=A.RELITEM AND A1.PRJC=' + PURDOC.PRJC +
')] , F[SODTYPE=&ITESODTYPE], I[QTY1], U[VAT;MTRUNIT;MTRSEASON;CDIM1;CDIM2;CDIM3;MTRPRJCB
LD;MTRCATEGORY=VAT;MTRUNIT4;MTRSEASON;CDIM1;CDIM2;CDIM3;MTRPRJCBLD;MTRCATEGORY@]) ');
    else
        X.SETFIELDEDITOR('ITELINES.MTRL',
'ITEM(W[l=0], F[SODTYPE=&ITESODTYPE], I[QTY1], U[VAT;MTRUNIT;MTRSEASON;CDIM1;CDIM2;CDIM3
;MTRPRJCBLD;MTRCATEGORY=VAT;MTRUNIT4;MTRSEASON;CDIM1;CDIM2;CDIM3;MTRPRJCBLD;MTRCATGO
RY@]) ');
}
```

ON_DOCPRINT

Fires before printing a print-out form

ON_OPENREPORT

Fires before the execution of a report

EXECCOMMAND (cmd)

Fires when a command is triggered (e.g. button click, right-click menu option)

EXECCOMMAND(cmd) (cmd=-1)

Fires before a record is copied to another record (before "Copy from last")

EXECCOMMAND(cmd) (cmd=-2)

Fires after a record is copied to another record (after "Copy from last")

ON_RESTOREEVENTS

Fires when a new record is added from conversion of another record (e.g. right click menu-job "Sales Conversion")

Example

Module: Sales Documents

Job: Preserve document pay-terms data on conversion.

```
function ON_RESTOREEVENTS () {
    GetPreviousDocPayTerms ();
}

function GetPreviousDocPayTerms () {
    ITELINES.FIRST;
    if (ITELINES.FINDOCS > 0) {
        var str = "SELECT FINPAYTERMS, FINALDATE, AMNT FROM FINPAYTERMS WHERE
FINDOC=:1";
        var ds = X.GETSQLDATASET(str, ITELINES.FINDOCS);
        if (ds.RECORDCOUNT > 0) {
            while (FINPAYTERMS.RECORDCOUNT > 0) {
                FINPAYTERMS.DELETE;
            }
            VDOSEIS.EDIT;
            while (!ds.EOF) {
                FINPAYTERMS.APPEND;
                FINPAYTERMS.FINALDATE = ds.FINALDATE;
                FINPAYTERMS.AMNT = ds.AMNT;
                FINPAYTERMS.POST;
                ds.NEXT;
            }
        }
    }
}
```

ON_EDIT

Fires when users alter the data of any field inside a record of an object

Example

Module: Sales Documents

Job: Prohibit a specific group of users of editing records in purchase documents.

```
function ON_EDIT() {  
    if (X.SYS.GROUPS == 100) //Checks if user's group is 100  
        X.CANCELEDITS;  
}
```

ON_AFTEREXECUTE

Event that fires in dialog objects after batch jobs are completed.

Example

Batch Job: CONVERTDLG

Convert sales documents using mass conversion (Figure H3) and get the ids of the created documents.

```
function ON_AFTEREXECUTE() { //After Batch Job Execution  
    X.WARNING('ON_AFTEREXECUTE Event Fired');  
    //debugger;  
    if (X.GETPARAM('SOSOURCE') == 1351) {  
        if (CONVERTFPRMS.VIEWTRANSMODE == 2) { //Mass Convert  
            var vNewFindocIds = "";  
            VFIN.FIRST(); //VFIN Table stores the new ids (Field VFIN.FINDOC)  
            while (!VFIN.EOF) {  
                if (VFIN.FINDOC != 0) {  
                    vNewFindocIds += VFIN.FINDOC + ",";  
                    // Create SALDOC object, Locate record, Alter data, DBPost  
                }  
                VFIN.NEXT;  
            }  
            if (vNewFindocIds != "") {  
                vNewFindocIds = vNewFindocIds.slice(0, -1); //Remove last comma  
                //X.RUNSQL("UPDATE FINDOC SET FINSTATES=10 WHERE FINDOC IN (" +  
vNewFindocIds + ")", null);  
                X.WARNING("New document ids: " + vNewFindocIds);  
            }  
        }  
    }  
}
```

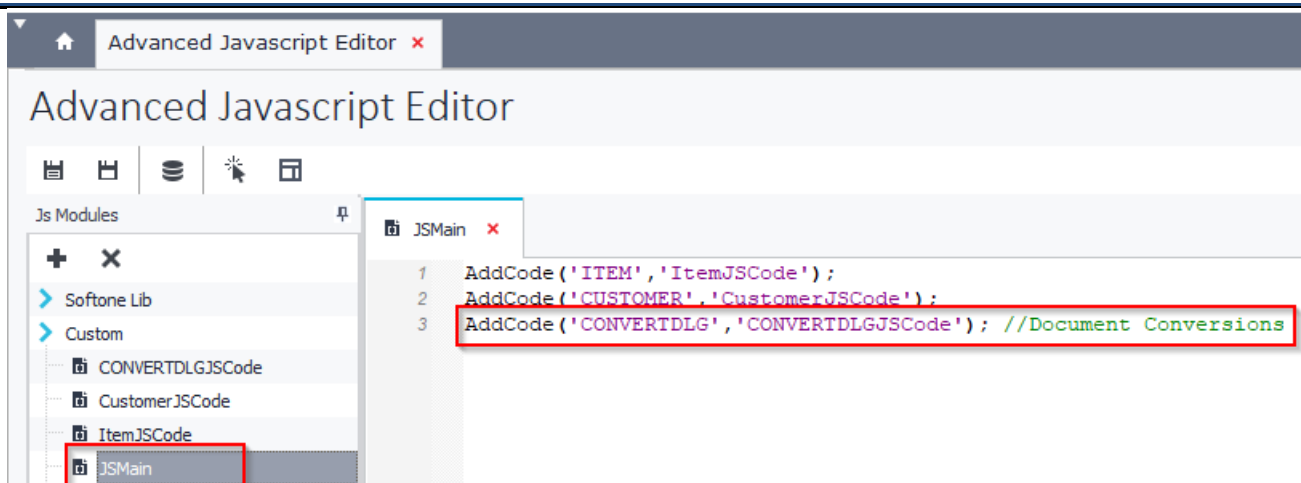


Figure H1 – JS Main

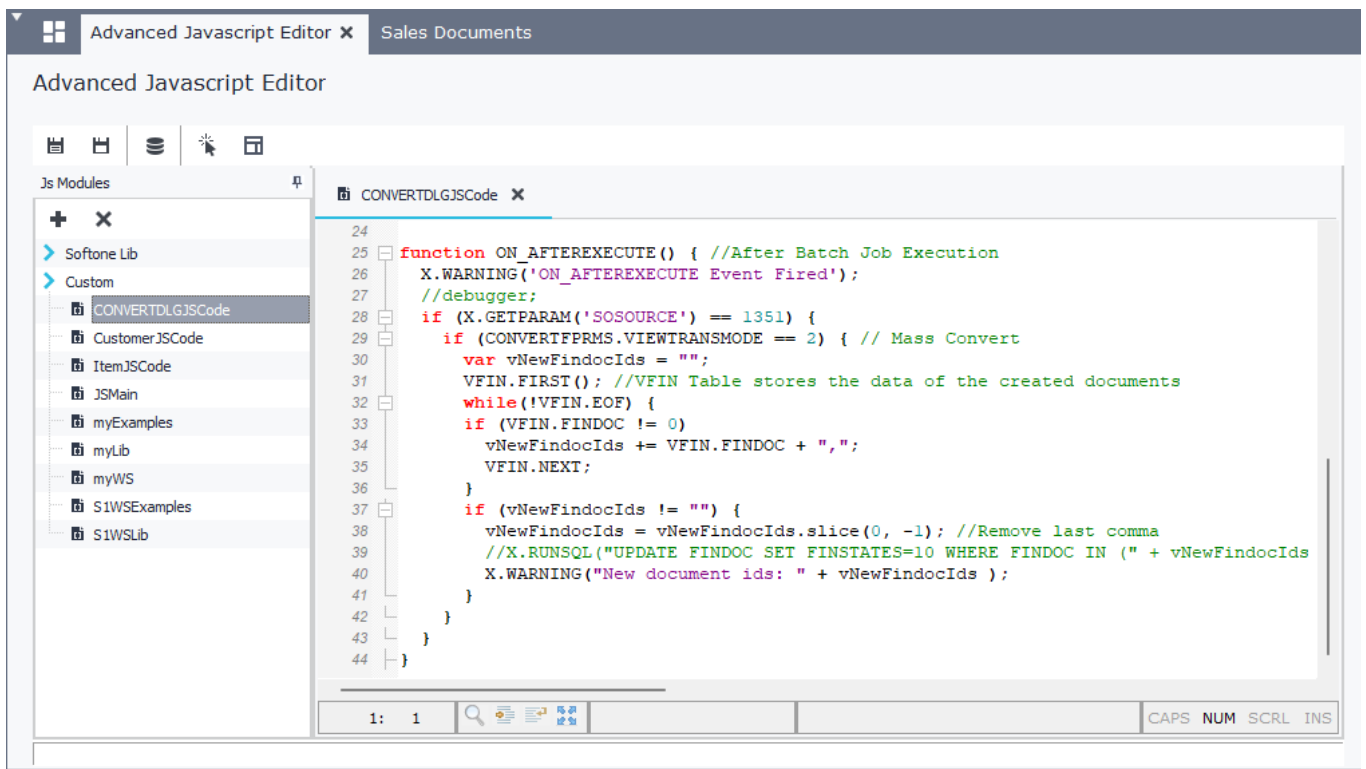


Figure H2 – Event “ON_AFTEREXECUTE”

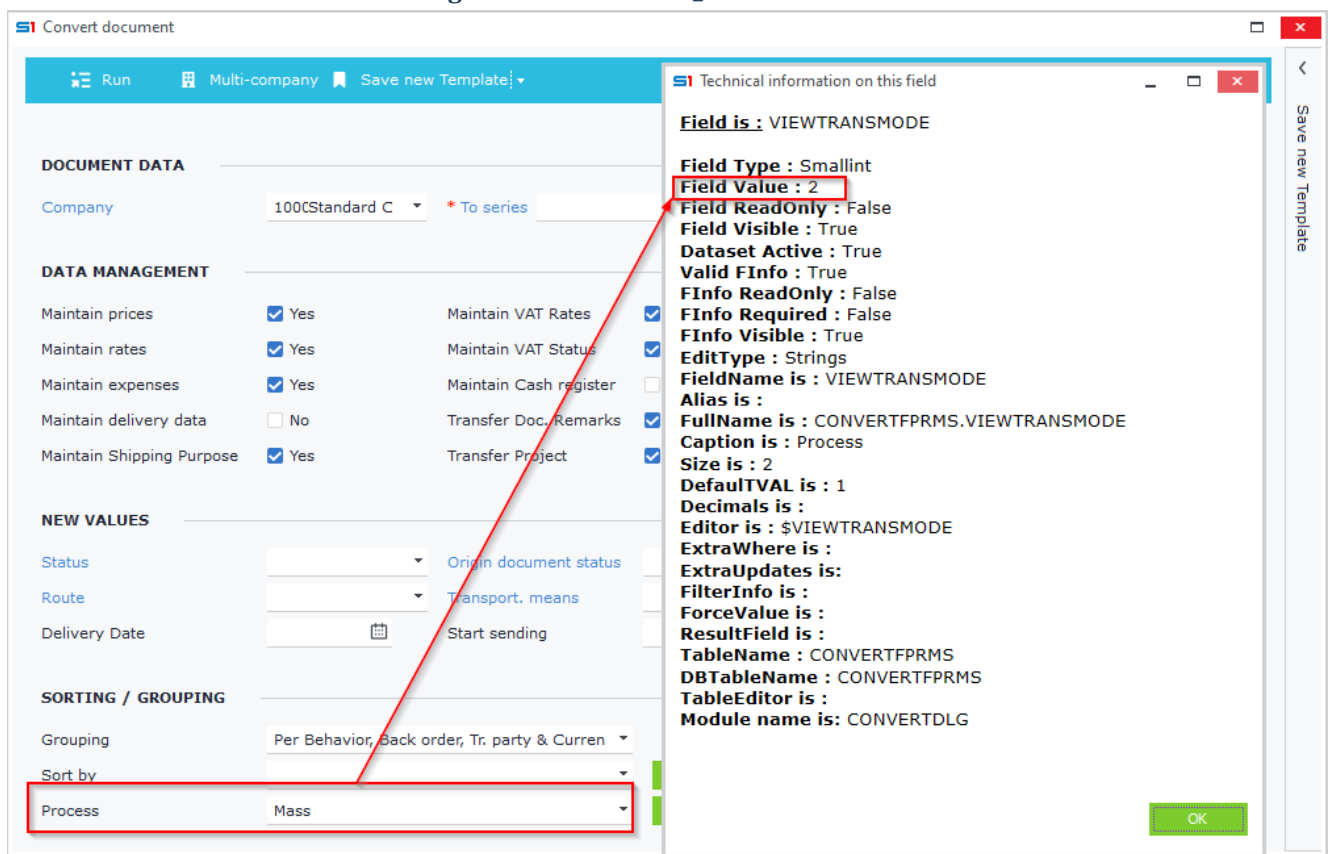


Figure H3 – Dialog Filter “Mass Conversion”

I. Sub Form Events

This module demonstrates all the events that occur in sub-forms (pop-up forms).

Attention: Sub-Form names must not contain underscores.

ON_SubFormName_SHOW

Fires on before-show of a sub form.

Example

Module: Sales Documents

Job: Populate a custom virtual table CCCPICKITEMS using specified SQL query. The data (Item, Qty, etc.) are displayed in a pop-up form (FORMPICKITEMS).

```
function ON_FORMPICKITEMS_SHOW() {
    CCCPICKITEMS.FIRST;
    while (CCCPICKITEMS.RECORDCOUNT > 0) {
        CCCPICKITEMS.DELETE;
    }
    if (SALDOC.TRDR > 0) {
        vToDate = X.EVAL('SQLDate(LoginDate+1)');
        vMonthDate = X.EVAL('SQLDate(IncMonth(LoginDate,-1))');
        vYearDate = X.EVAL('SQLDate(IncYear(LoginDate,-1))');
        var ds = X.GETSQLDATASET('SELECT M.MTRL, M.CODE, M.NAME, ' +
                                '(SELECT SUM(V1.EXPQTY1) ' +
                                'FROM VMTRSTAT V1 ' +
                                'WHERE V1.MTRL=M.MTRL ' +
                                'AND V1.TRNDATE>=' + vMonthDate + ' ' +
                                'AND V1.TRNDATE<' + vToDate + ') AS QTY1MONTH, ' +
                                '(SELECT SUM(V2.EXPQTY1) ' +
                                'FROM VMTRSTAT V2 ' +
                                'WHERE V2.MTRL=M.MTRL ' +
                                'AND V2.TRNDATE>=' + vYearDate + ' ' +
                                'AND V2.TRNDATE<' + vToDate + ') AS QTY1YEAR ' +
                                'FROM MTRL M ' +
                                'WHERE M.COMPANY=' + X.SYS.COMPANY + ' ' +
                                'AND M.SODTYPE=51 ' +
                                'AND EXISTS(SELECT 1 ' +
                                'FROM MTRTRN N ' +
                                'WHERE N.MTRL=M.MTRL ' +
                                'AND N.TRDR=' + SALDOC.TRDR + ') ' +
                                'ORDER BY M.CODE, M.NAME', null);

        if (ds.RECORDCOUNT > 0) {
            ds.FIRST;
            while (!ds.EOF()) {
                CCCPICKITEMS.APPEND;
                CCCPICKITEMS.MTRL = ds.MTRL;
                CCCPICKITEMS.CODE = ds.CODE;
                CCCPICKITEMS.NAME = ds.NAME;
                CCCPICKITEMS.QTY1 = 0;
                CCCPICKITEMS.QTY1OLD = ds.QTY1MONTH;
                CCCPICKITEMS.QTY1CONTRACT = ds.QTY1YEAR;
                CCCPICKITEMS.POST;
                ds.NEXT;
            }
        } else {
            X.WARNING('No transactions found for the specific customer!');
        }
    } else {
        X.WARNING('Please select a customer!');
    }
}
```

ON_SubFormName_ACTIVATE

Fires on after-show of a sub form. Usually used to change the layout (hide panels, change colors).

```
function ON_MYPOPUPFORM_ACTIVATE () {  
    X.FIELD COLOR ('CCCMYVIRTUALTABLE.CCCS1INIT_CCCS1INIT_MYITEMSNO', 50000);  
}
```

ON_SubFormName_ACCEPT

Fires when users click on the "OK" button of a sub form.

Example

Module: Sales Documents

Job: Add items to sales document from custom virtual table (CCCPICKITEMS), that is displayed through pop up form (FORMPICKITEMS). The virtual table is populated when the pop-up form is displayed (event ON_SHOW, see above example).

```
function ON_FORMPICKITEMS_ACCEPT () {  
    CCPICKITEMS.FIRST;  
    while (!CCCPICKITEMS.EOF ()) {  
        if (CCCPICKITEMS.QTY1 > 0) {  
            ITELINES.APPEND;  
            ITELINES.MTRL = CCPICKITEMS.MTRL;  
            ITELINES.QTY1 = CCPICKITEMS.QTY1;  
            ITELINES.POST;  
        }  
        CCPICKITEMS.NEXT;  
    }  
}
```

ON_SubFormName_CANCEL

Fires when users click on the "CANCEL" button of a sub form.

J. Report Objects Events

This module demonstrates all the line printing events that you can use in report objects.

ON_BANDSTART(BandName)

Start Band Print

ON_BANDEND(BandName)

End Band Print

ON_BANDLINE(BandName)

Line Band Print

ON_BANDTRANSFER(BandName)

Print Transfer Values

Example

```
function ON_OPENREPORT () {
    X.SETPARAM('STARTPAGEFIELD', USR.FPAGEL);
    return true;
}

function ON_BANDTRANSFER(BandName) {
    X.SETTRANSFERVALUE(BandName, 'MYACNTRN.X1', 1);
    X.SETTRANSFERVALUE(BandName, 'MYACNTRN.P1', 2);
    return true;
}

function ON_BANDEND(BandName) {
    if ( BandName=='MYACNTRN' ) {
        X.SETPRINTPARAM(BandName, "CAPTIONFIELDNAME", "THECAPTION");
        ds=X.GETENDBANDDATASET('MYACNTRN');
        ds.APPEND;
        for (i=0; i<ds.FIELDSCOUNT; i++) ds.FIELDSDS(i)= i;
        ds.THECAPTION = "MyCaption";
        ds.POST;
    }
    return true;
}

function ON_BANDLINE(BandName) {
    if ( BandName=='MYACNTRN' ) {
        MYACNTRN.EDIT;
        if ( MYACNTRN.FINCODE == 'Sum' )
            MYACNTRN.FINCODE = 'NewName';
    }
    return true;
}
```

K. Advanced JavaScript

Advanced JavaScript is an embedded tool for writing scripts in JavaScript. Its main purpose is to override SoftOne objects (internal or custom) functionality, as form scripts, but the main difference is that it affects all the forms of an object (as [EDA](#)). It uses the same principles, events, methods and functions as form scripts and it's categorized in modules and packages, where one package may contain many modules. There is also another way to use it, which is for creating custom web services that is discussed in Chapter Web Services.

Modules are saved as records inside the table CSTINFO (CSTTYPE=16) and can be imported / exported as ".cst", ".auv" files using the "[Custom Administration](#)" tool.

K.1 JSMain

The Module that is triggered on Login is **JSMain** and it needs to be added manually. Right click on "Custom" Package, select "New Module" and enter the name JSMain (Figure K1.1).

Inside this module enter the SoftOne objects and the names of the corresponding modules using the function below (use any name for the modules).

```
AddCode ( 'ObjectName' , 'ModuleName' ) ;
```

In the example of Figure K1.2, JSMain adds JavaScript code that overrides the functionality of the objects ITEM, CUSTOMER and CONVERTDLG (Documents Conversion).

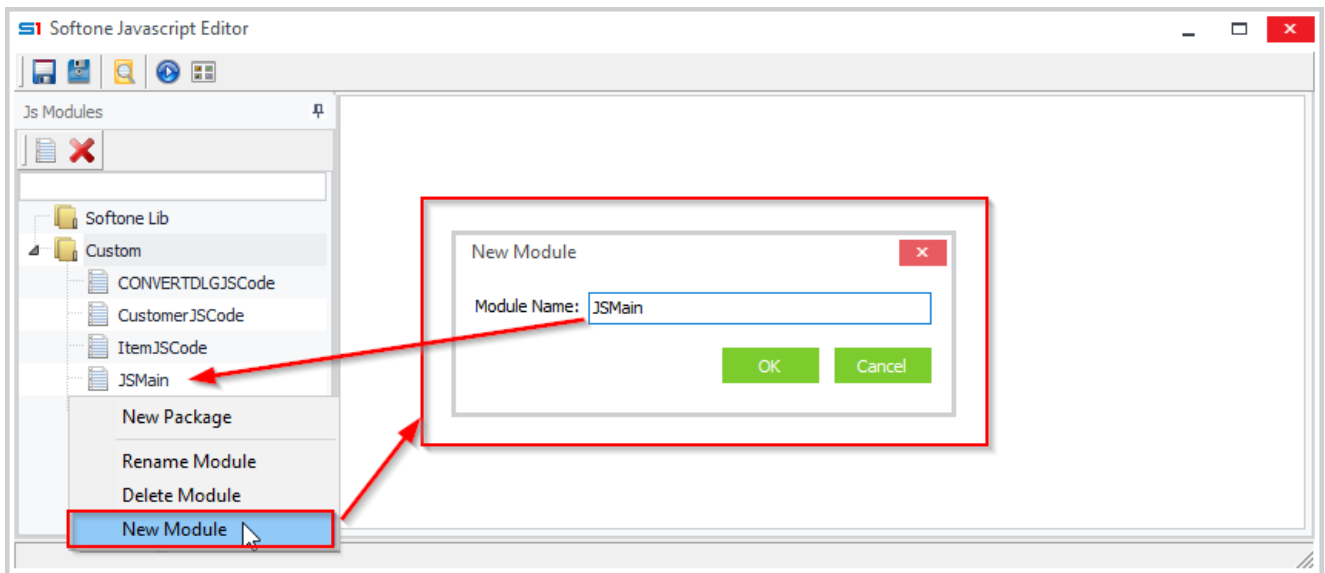


Figure K1.1 - JS Main

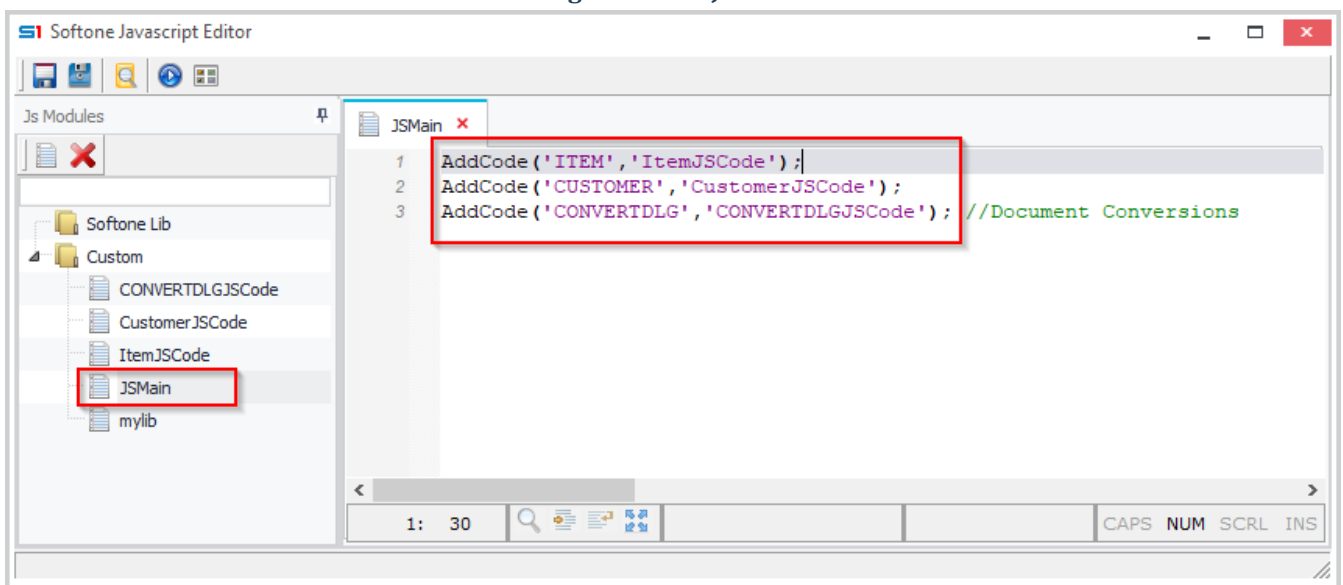


Figure K1.2 - Custom Library

Module **CustomerJSCode** (Figure K1.3) contains code that displays a warning message when a new customer record is posted. Notice in Figure K1.4 that the selected form is the default one.

The custom function **CheckUsersAccess**, that is used in **ON_FORMLOAD** and **ON_LOCATE** events, uses the function **X.FORM** in order to ban specific users from using the form SOACTIONDBView (Figure K1.5).

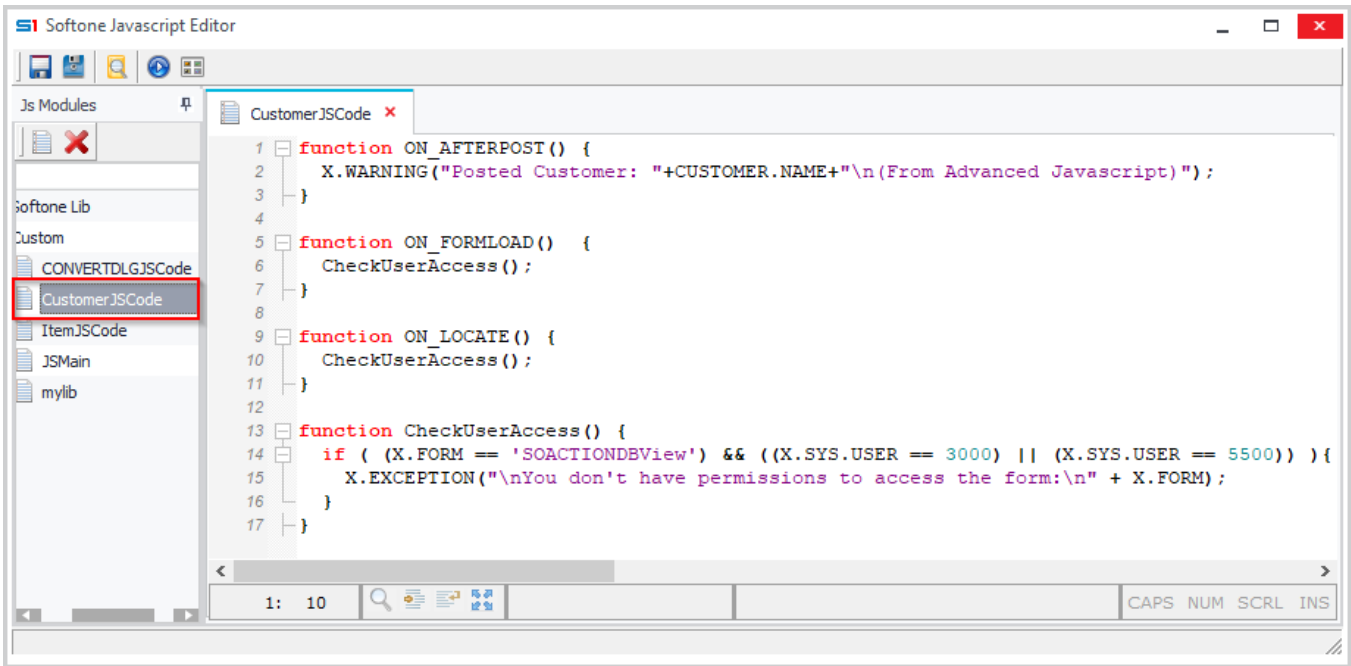


Figure K1.3 – Module CustomerJSCode

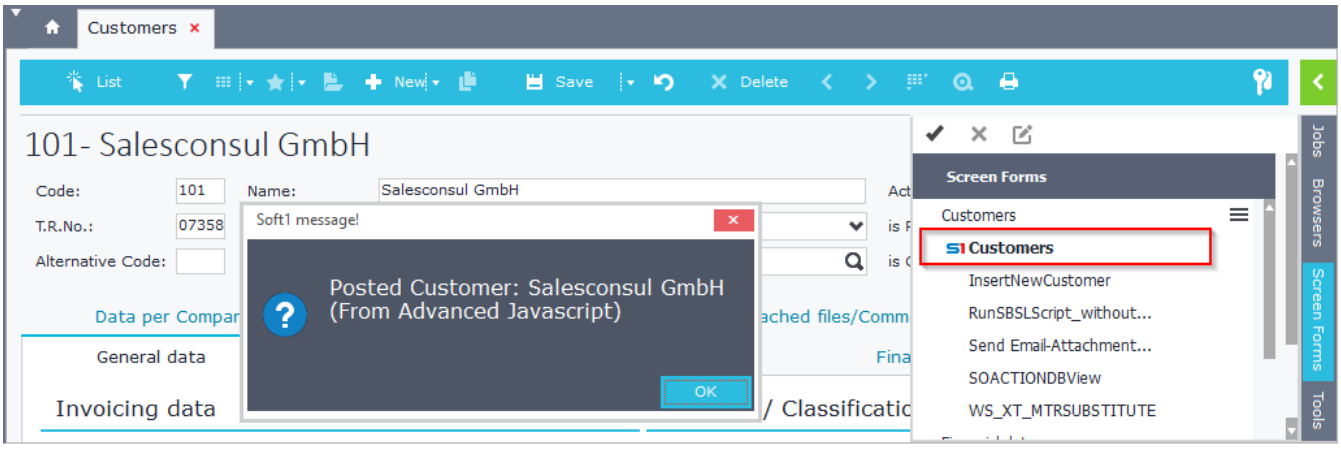


Figure K1.4 – Warning Message on Customer post

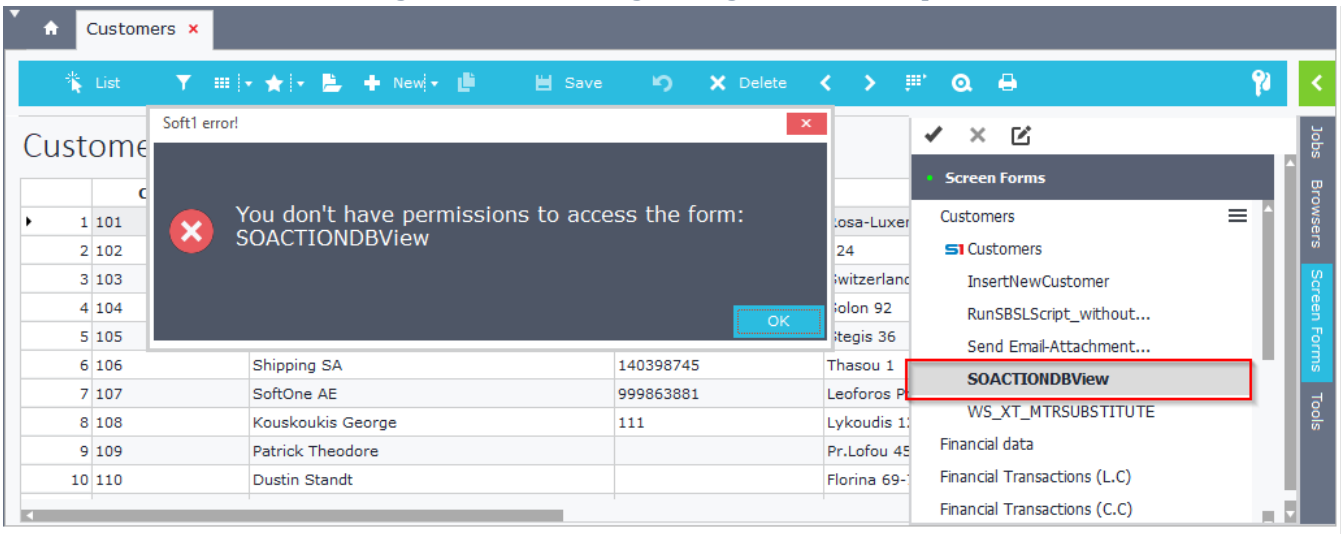


Figure K1.5 – Prohibition Message

The module **CONVERTDLGJSCode** (Figure K1.6) contains code that affects the dialog of documents conversion.

It uses the ON_CREATE event to change the Expenses filter flag to Yes, only for Sales Documents Conversion (sosource=1351).

It also overrides the change event of the filter field Company, in order to alter the value of Series (Figure K1.8). Notice the custom warning messages (sosource and selected records) that are displayed when the dialog is created (Figure K1.7).

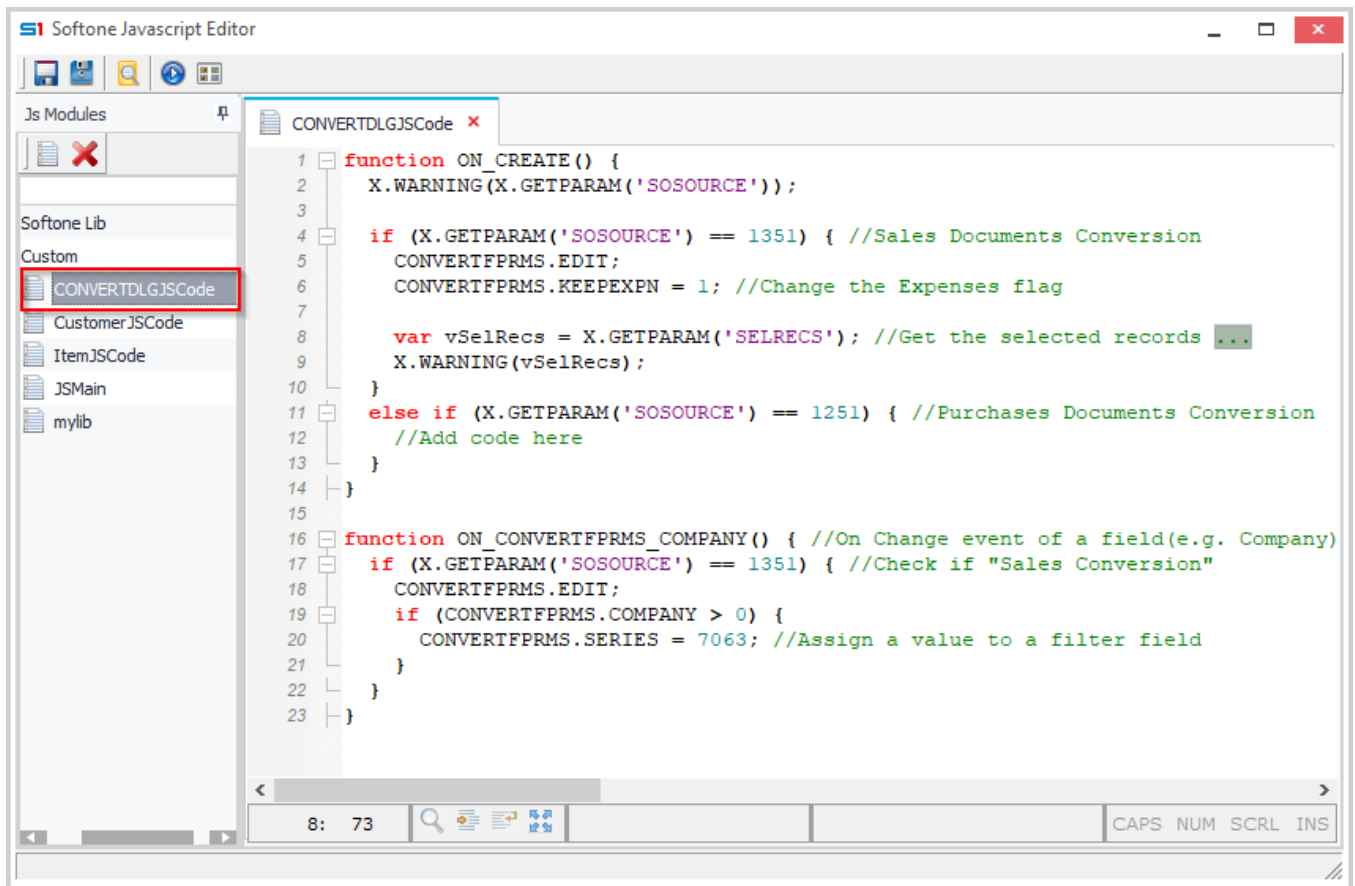


Figure K1.6 – Conversion Dialog script

```
function ON_CREATE() {
    X.WARNING(X.GETPARAM('SOSOURCE'));

    if (X.GETPARAM('SOSOURCE') == 1351) { //Sales Documents Conversion
        CONVERTFPRMS.EDIT;
        CONVERTFPRMS.KEEPEXP = 1; //Change the Expenses flag

        var vSelRecs = X.GETPARAM('SELRECS'); //Get the selected records ...
        X.WARNING(vSelRecs);
    }
    else if (X.GETPARAM('SOSOURCE') == 1251) { //Purchases Documents Conversion
        //Add code here
    }
}

function ON_CONVERTFPRMS_COMPANY() { //On Change event of a field(e.g. Company)
    if (X.GETPARAM('SOSOURCE') == 1351) { //Check if "Sales Conversion"
        CONVERTFPRMS.EDIT;
        if (CONVERTFPRMS.COMPANY > 0) {
            CONVERTFPRMS.SERIES = 7063; //Assign a value to a filter field
        }
    }
}
```

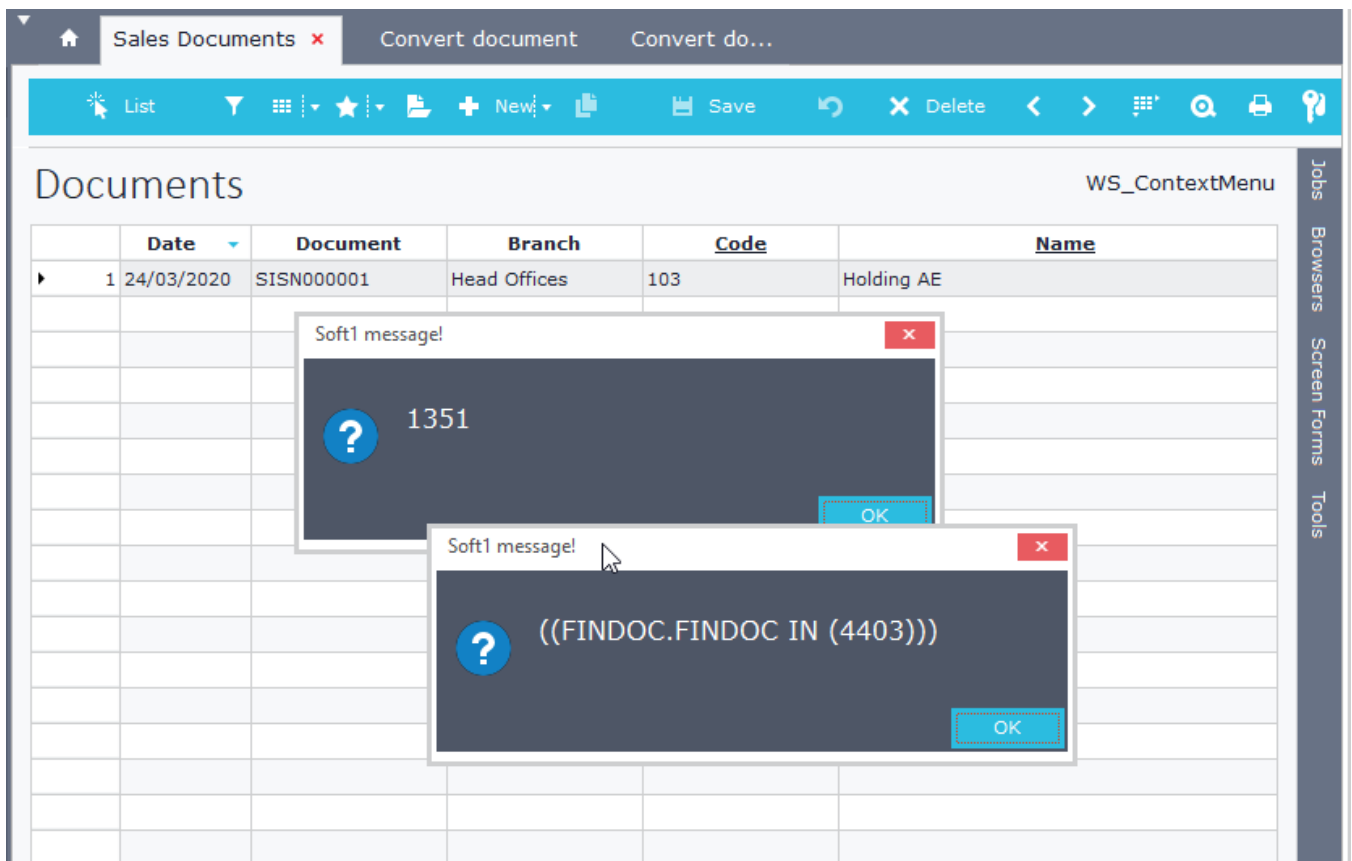


Figure K1.7 – Conversion Sosource parameter and selected records

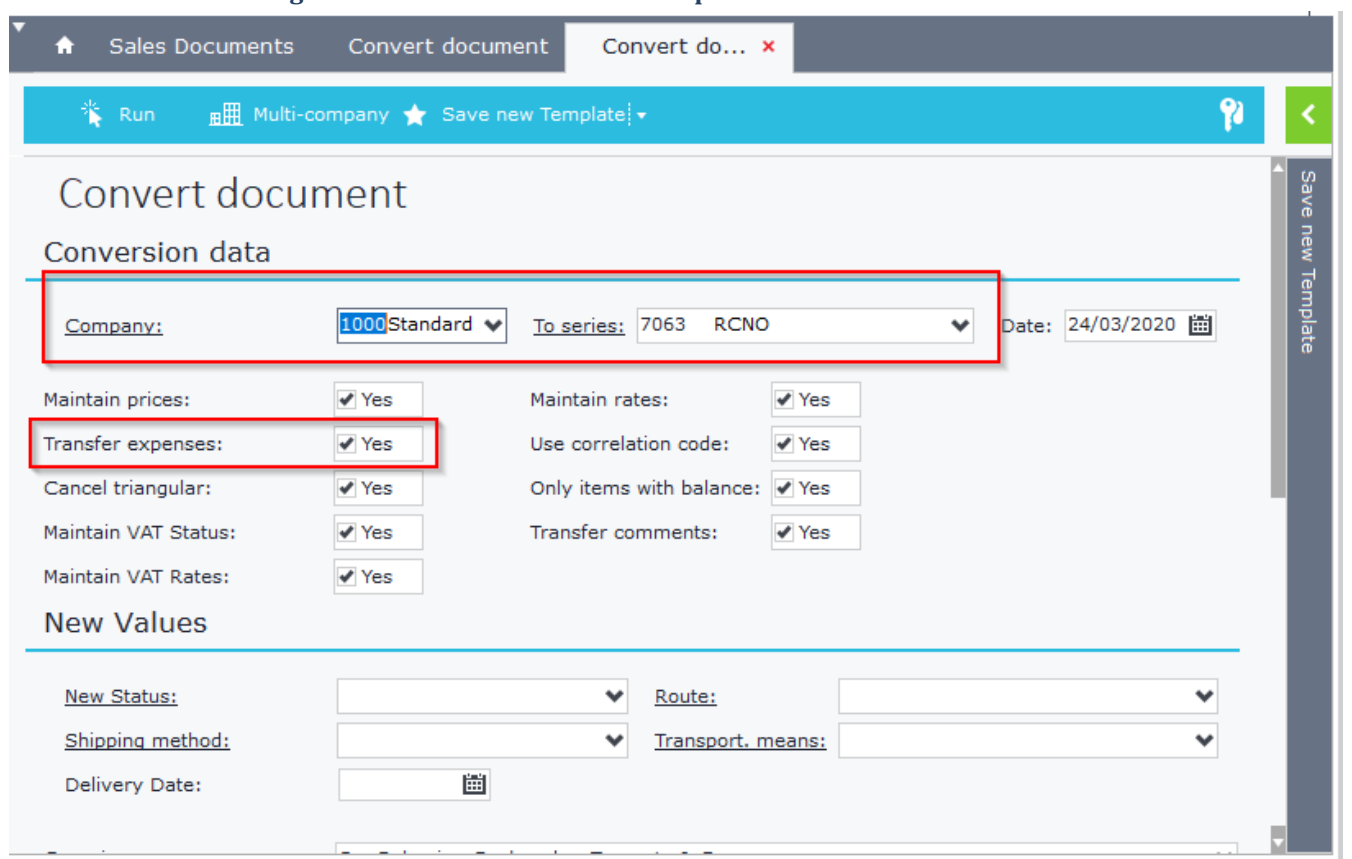


Figure K1.8 – Conversion Dialog Filters

K.2 Library

Another feature of Advanced JS is that it can be used as library that holds custom functions, which can be later referenced in other Adv.JS or Form scripts through the command **lib.include('MyCustomLib')** (Figures K2.1 & K2.2).

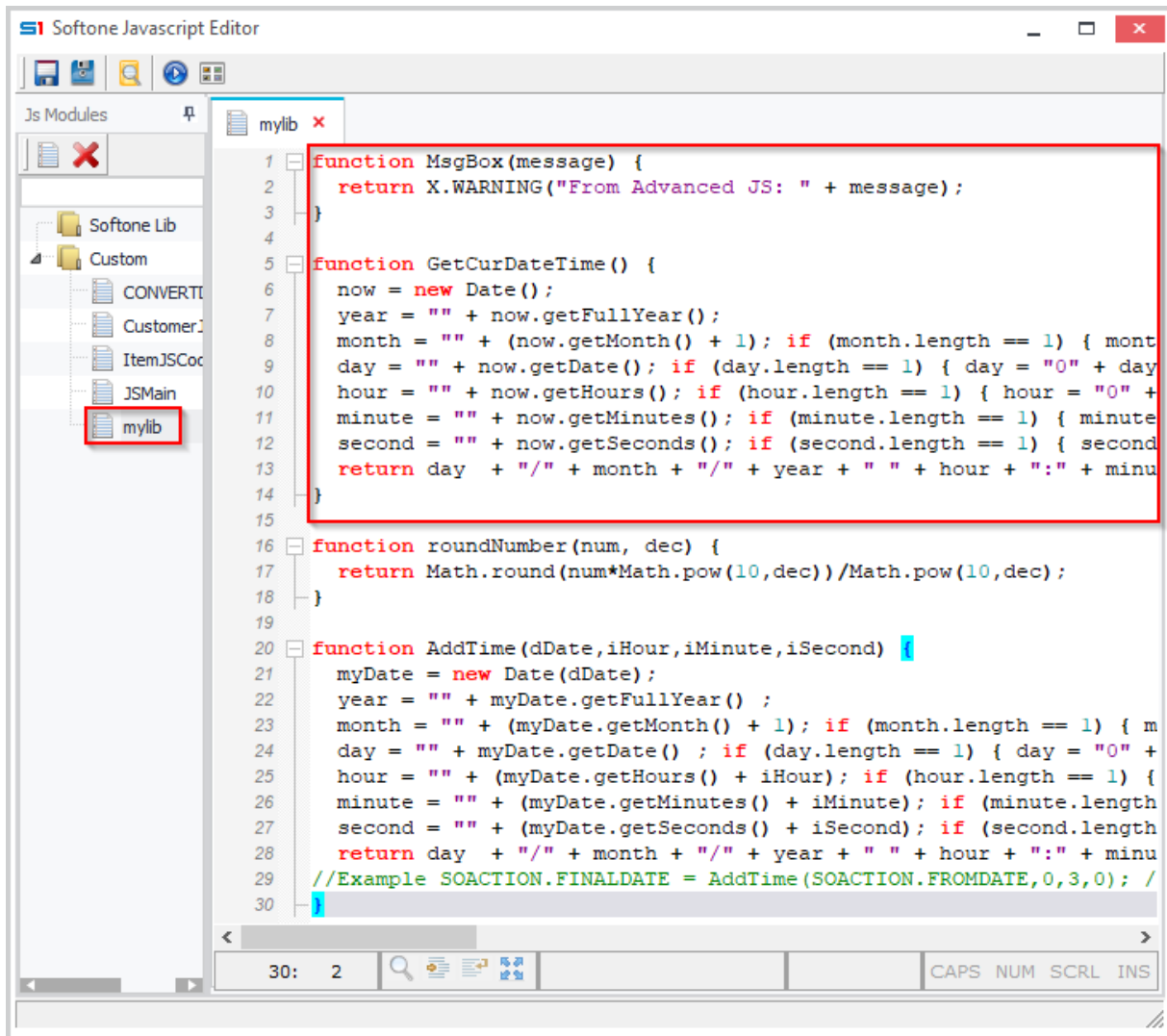


Figure K2.1 – Custom Library

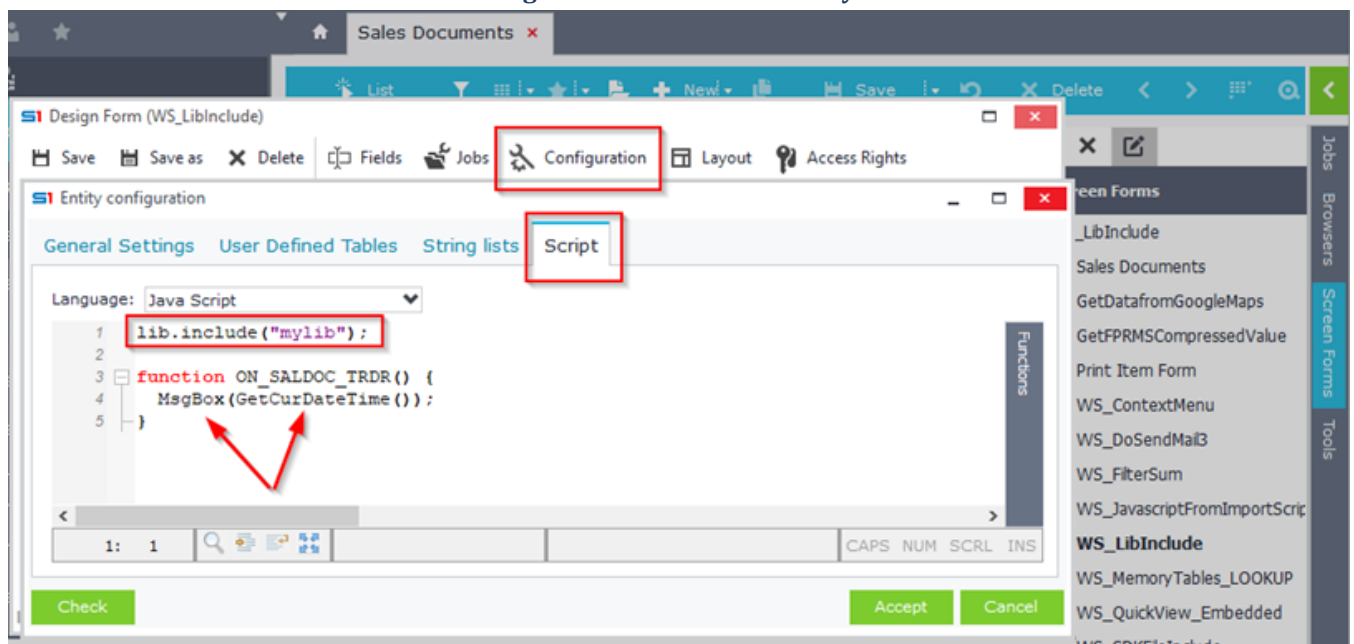



Figure K2.2 – Library reference in Sales Documents script

K.3 Debug Functions

All the functions created in Advanced JavaScript can be tested using the button  "Run JavaScript". Enter the name of the function inside "Function Name" and click on "Run". The Result will appear as in Figure K3.1.

Functions with parameters can be tested as in Figure K3.2.

```
function GetOrdersJSON(vFiscprd, vPeriod) {  
    var str = "SELECT FINDOC, SERIES, TRDR, SUMAMNT FROM FINDOC WHERE COMPANY=:1 AND  
    TFPRMS=201 AND FISCPRD=:2 AND PERIOD=:3";  
    var ds = X.GETSQLDATASET(str, X.SYS.COMPANY, vFiscprd, vPeriod);  
    return ds.JSON;  
}
```

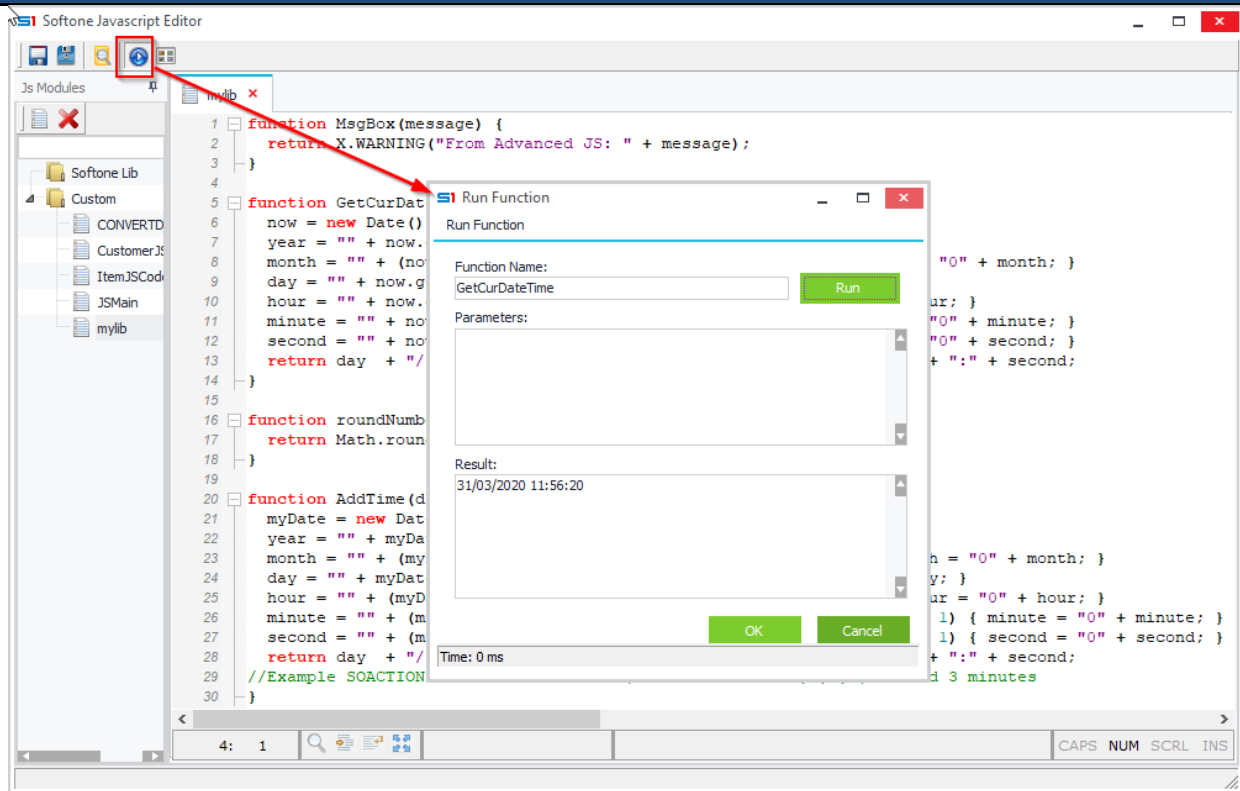


Figure K3.1 – Test Function

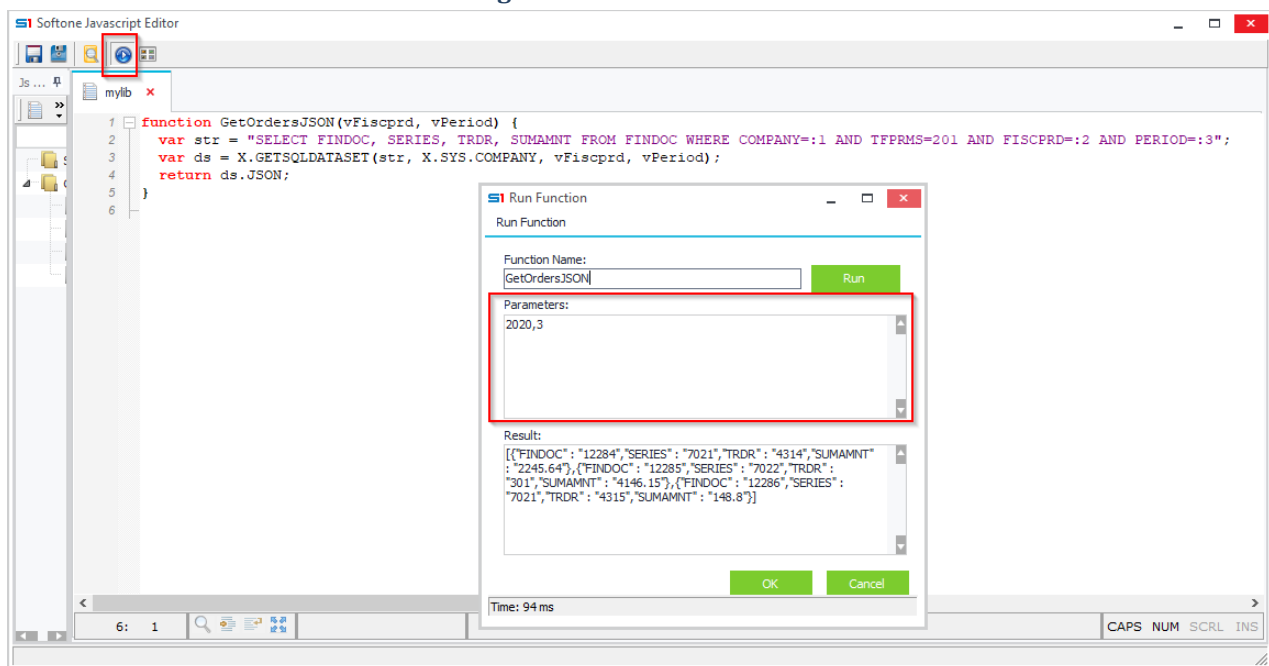


Figure K3.2 – Test Function with Parameters

Case Studies

1. Print Form using dialog that asks for number of copies

Module: Sales Documents

Job: Print item templates form using custom button for the selected item line.

Extra features: Prompt user to enter number of copies to print, through a dialog message box. Default value is the quantity of the selected item line.

```
function EXECCOMMAND(cmd) { //Controls Commands
    if(cmd == 20180101) { //Checks the ID of the button clicked
        iCopies = X.INPUTQUERY('Print item', 'Number of copies: ', ITELINES.QTY1, 0);
        //The above line displays a dialog box, which allow users to specify the
        number of copies to print. Default value is the quantity of the item line.
        if(iCopies != '') {
            var ObjItem;
            i = 1;
            ObjItem = X.Createobj('ITEM'); //Create Object Item
            try {
                ObjItem.DBLocate(ITEMINES.MTRL); //Locate ITEM using the selected
                item from document lines
                while(i <= iCopies) {
                    ObjItem.PRINTFORM(100, 'HP LaserJet 4100 Series PCL', '');
                    i++;
                }
            }
            catch(e) {
                if (myObj != null)
                    X.WARNING("General Error: " + e.message + "\nObject Error: "
                    + myObj.GETLASTERROR);
                else
                    X.WARNING("General Error: " + e.message);
            }
            finally {
            }
        }
    }
}
```

Standard Company: 100- Item Form

Code: 100 Description: Item Form

Form type: Internal (Fast Report)

Design report form: Parameters

Page 1: TfrxReportPage

Properties: lot assigned, BackPicture, BackPicturePrintable, BackPictureStretched, BackPictureVisible, BottomMargin, Color, Columns, DataSet, Duplex, EndlessHeight, EndlessWidth, Font, Frame, PageHeader: PageHeader1, PageFooter: PageFooter1, Page#

Code	Description	Barcode
ITEM.COD	ITEM.NAME	ITEM.COD

Figure CS1.1 - Item Printout Form

The screenshot shows the 'Design Form' window with a 'Properties Area' on the left. The 'Edit field' dialog box is open, displaying the text 'RUNB_20180101=Print Labels'. A red arrow points from the 'Print Labels' button in the main form to the dialog box.

Figure CS1.2 – Form button design

The screenshot shows the 'Sales Documents' window with a 'Print Item' dialog box open. The dialog box contains the text 'Number of copies:' and a text box with the value '5'. A red arrow points from the 'Print Labels' button in the main form to the dialog box.

Code	Description	Qty 1	Price	Disc. ...	Value	VA
1 20001	Sunglasses	5	52,00		260,00	VAT 23%

Figure CS1.3 – Dialog "Number of Copies", on click of button "Print Labels"

2. Delete related converted documents

Module: Sales documents

Job: Button that deletes the converted documents related to the located record.

```
function EXECCOMMAND(cmd) { //Controls Commands
    if(cmd == 100002) { //Check the ID of the button clicked
        DelConvertedDocs();
    }
}

function DelConvertedDocs() { //Finds and deletes converted documents.
    //SQL statement for searching converted documents based on the located
    strqry = 'SELECT DISTINCT FINDOC AS FINDOC FROM MTRLINES WHERE SOSOURCE=1351 '
        + 'AND COMPANY='+X.SYS.COMPANY+' AND FINDOCS=' + SALDOC.FINDOC;
    ds = X.GETSQLDATASET(strqry,null);
    ds.FIRST;
    while(!ds.EOF()) { //Loop and delete the documents found
        ObjConv = X.CreateObj('SALDOC'); //Create Sales Object
        try {
            ObjConv.DBLocate(ds.FINDOC); //Locate record
            ObjConv.DBDelete; //Delete record
        }
        catch(e) {
            if (ObjConv != null)
                X.WARNING("General Error: " +e.message+"\nObject Error: "+
ObjConv.GETLASTERROR);
            else
                X.WARNING("General Error: " +e.message);
        }
        finally {
        }
        ds.NEXT;
    }
}
```

Open “compressed” field SOVAL (table FPRMS), find the value of the accounting field (ACNMSK) and display it in a message box.

```
function GetACNMSK() {
    var dsFPRMS = X.GETSQLDATASET("SELECT SOVAL "
                                   + "\n FROM FPRMS "
                                   + "\n WHERE SOSOURCE=1351 "
                                   + "\n AND COMPANY="+X.SYS.COMPANY+" AND

FPRMS="+SALDOC.FPRMS,"");
    var strSOVAL = dsFPRMS.SOVAL;

    var sVal = X.EVAL("GETQUERYVALUE('"+ strSOVAL +"',0,25)");
    //Returns the value of the 1st line (zero based) and 26th place of the field
    SOVAL delimited by character |. (The above example uses 0,25)

    X.WARNING("Account field value for Type "+ SALDOC.FPRMS +" is : " + sVal);
}
```

The screenshot shows the SAP Sales Documents design interface. The 'Accounting' tab is selected, and the 'Account' field is highlighted. A technical information popup is open, displaying the following details for the field 'ACNSMK':

- Field is : ACNSMK
- Field Type : String
- Field Value : 95.00
- Field ReadOnly : False
- Field Visible : True
- Datasec Active : True
- Valid Info : True
- Control ReadOnly : False
- Control Enabled : True
- Info ReadOnly : False
- Info Required : False
- Info Visible : True
- EditType : Selector
- FieldName is : ACNSMK
- Alias is :
- FullName is : SALFPRMS.ACNSMK
- Caption is : Account
- Size is : 25
- DefaultTVAL is :
- Decimals is :
- Editor is : ACNTGL
- ExtraWhere is :
- ExtraUpdates is :
- FilterInfo is :
- ISACTIVE;ACNSHEMA;SODTYPE=1;;X.SYS.ACNSHEMA;99
- ForceValue is :
- ResultField is : CODE
- TableName : SALFPRMS
- DBTableName : FPRMS
- TableEditor is : ACNTGL

Figure CS3.1 – FPRMS form – Account Field (CTRL + SHIFT + F12)

Figure CS3.2 - SQL statement that returns the data of the field SOVAL (Position of the field ACNMSK)

4. Run Batch Job

Module: Items

Job: Execute the batch job "Modify sales prices" using a custom button.

Extras: 1) Change the value of the job filter "Current prices and given percentage".
2) Prompt user to enter the percentage that will be used as job filter.

```
function ChangePrices() {
    ObjIteCost = X.CreateObj('IteCalcMarkUp');
    try {
        ObjIteCost.DBInsert;
        TblIteCost = ObjIteCost.FindTable('MTRDIALOG');
        TblIteCost.Edit;
        TblIteCost.FROMCODE = ITEM.CODE;
        TblIteCost.TOCODE = ITEM.CODE;
        iSODOUBLE1 = X.INPUTQUERY('Modify sales prices', 'Percentage value',0,0);
        if(iSODOUBLE1!='') { //Not Cancel
            TblIteCost.SODTYPE = 1; //Calculation based on "Current prices and given
percentage"
            TblIteCost.SODOUBLE1 = iSODOUBLE1;
            ObjIteCost.BatchExecute;
            X.EXEC('button:Save'); //Saves current item record and relocates to
display the calculation
            X.WARNING("Job completed!");
        }
    }
    finally {
    }
}
```

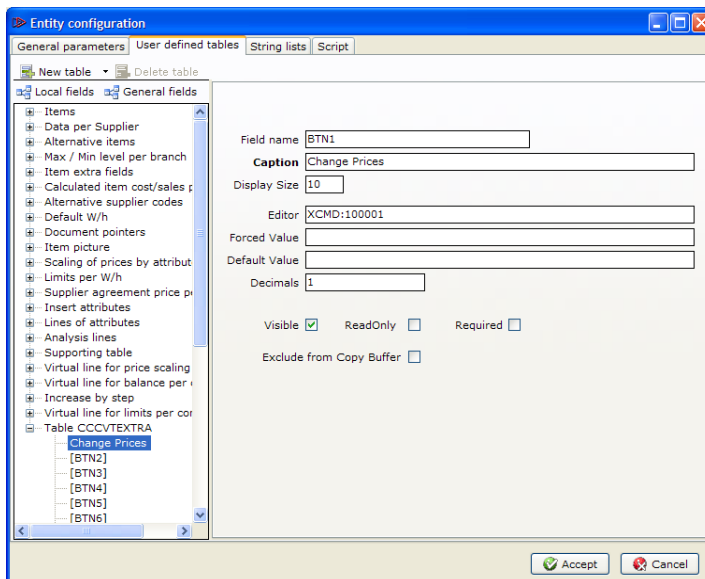


Figure CS4.1 – Design button (using virtual table field)

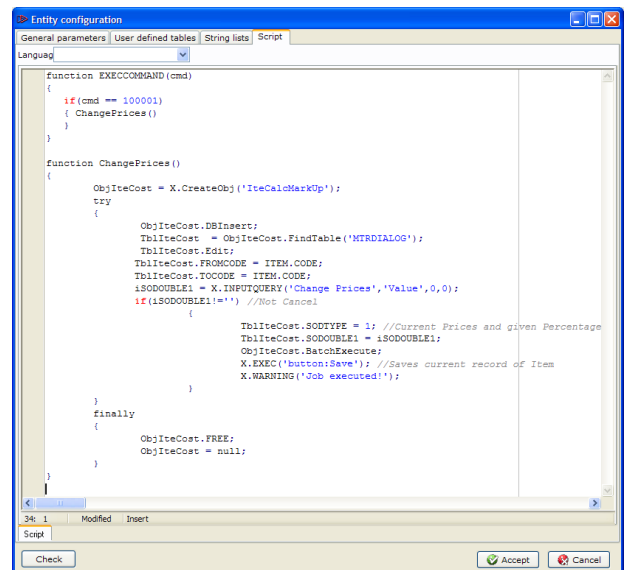


Figure CS4.2 - JavaScript code inside Items form

Code: 001 Name: ITEM 1 Active: Yes ☒

Acctg categ.: 101 Material Comm. category: U.O.M.: 101 PCS

Type: Normal Group: VAT: 1210 21%

General data Invoicing Sales Purchases Other data Special data User-defined fields Comments/Picture Related files

Alternative codes

Barcode: Correlation: Factory: Manufacturer:

Prices/Markup

Wholesale: 10,00 Retail: 0,00 W/sale markup: Retail markup:

Special attributes of item qty

Business unit: Season: Brand: Country of origin: Name 2: in KEPYO report: Yes ☒

Accounts

Purchases: 20 Sales: Stock: Gross operating profit/loss: Production cost: Prod / Purch cost of goods sold: Sales analysis: **Change Prices**

Change Prices

Value: 20

OK Cancel

Figure CS4.3 - Items form - Button click- Display of dialog window

Code: 001 Name: ITEM 1 Active: Yes ☒

Acctg categ.: 101 Material Comm. category: U.O.M.: 101 PCS

Type: Normal Group: VAT: 1210 21%

General data Invoicing Sales Purchases Other data Special data User-defined fields Comments/Picture Related files

Alternative codes

Barcode: Correlation: Factory: Manufacturer:

Prices/Markup

Wholesale: 12,00 Retail: 0,00 W/sale markup: Retail markup:

Special attributes of item qty

Business unit: Season: Brand: Country of origin: Name 2: in KEPYO report: Yes ☒

Accounts

Purchases: 20 Sales: 70 Stock: Gross operating profit/loss: Production cost: Prod / Purch cost of goods sold: Sales analysis: **Change Prices**

Soft1 message!

Job executed!

OK

Figure CS4.4 - Results (Price is increased by 20%)

5. Custom Screen Filters

Module: Sales documents

Job: Filter line items by name, based on custom field inside the form.

```
function ON_SALDOC_V1 () {  
  
vFilterString='{ITELINES.MTRL_ITEM_NAME}='+String.fromCharCode(39)+SALDOC.V1+String.fromCharCode(39)  
    if(SALDOC.V1=='') {  
        ITELINES.FILTERED = 0; //Deactivates all filters on ITELINES  
    }  
    else {  
        ITELINES.FILTER = '(' + vFilterString + ')'; //Filter value  
        ITELINES.FILTERED = 1; //Runs filter  
    }  
}  
  
function ON_POST() {  
    ITELINES.FILTERED = 0; //Deactivates all filters on ITELINES  
}  
  
function ON_CANCEL() {  
    ITELINES.FILTERED = 0; //Deactivates all filters on ITELINES  
}
```

The screenshot displays a software interface for managing sales documents. At the top, there are fields for Series (7001), Type (7001 Προσφορά), Number (2), and Document (ΠΡ20000002). Below these are tabs for General data, Shipping Details, and Other data. The General data tab is active, showing fields for Date (18/05/2010), Branch (999 Έδρα), W/h (2 SERVICE), Customer (1207 A.B. MARITIME INC), Salesperson, Payment (1000 Τοίς Μετρητοίς / Cash), Transit, Shipment, VAT status (Normal), Currency (999 EURO), TR ex. rate (1), TRP ex. rate (1), and a Comment field. Below the General data tab is the Items tab, which contains a table with columns: Code, Name, Qty 1, Price, Disc. 1(%), and Net value. The table lists 9 items, including undershirts, work trousers, and batteries. Below the table is a red-bordered box labeled "Name Filter:". At the bottom of the screen, there are summary fields: Total qty (9,00), Disc. 1(%) (0,00), Disc. 1 (0,00), Disc. 2(%) (0,00), Net value (168,21), VAT value (35,32), Expenses (0,00), and Total value (203,53).

	Code	Name	Qty 1	Price	Disc. 1(%)	Net value
1	00573350012	UNDERSHIRT TRIPLE-LAYER SINGLE LUE	1	26,20		26,20
2	0057986014	WORK TROUSERS MONTANA ORANGE - S	1	0,00		0,00
3	00573350014	UNDERSHIRT TRIPLE-LAYER SINGLE BLUE	1	26,20		26,20
4	00586380013	TROUSERS (UNDER) ARLANDA TRIPLE-LA	1	28,50		28,50
5	33122104	ALCOHOL METER AT-2000	1	0,00		0,00
6	331223P02	ALCOHOL SENSOR FOR ALCOSCAN AL-60	1	30,00		30,00
7	33270304P01	ALUMINUM 40x60cm	1	24,00		24,00
8	00572350012	TROUSERS (UNDER) ARLANDA SINGLE-LA	1	29,80		29,80
9	792421	ALKALINE BATTERY LR20 MAXIMA SUNLIC	1	3,51		3,51

Name Filter:

Total qty: 9,00 Disc. 1(%): 0,00 Disc. 1: 0,00 Disc. 2(%): 0,00
Net value: 168,21 VAT value: 35,32 Expenses: 0,00 Total value: 203,53

Figure CS5.1 - Documents screen with lines filter field

Series: 7001 OFF Type: 7001 Quotation Number: 2 Document: ΠΡΣ0000002

General data Shipping Details Other data

Date: 18/05/2010 Branch: 999 Έδρα W/h: 2 SERVICE

Customer: 1207 A.B. MARITIME INC Cust. branch:

Salesperson: Payment: 1000 Τοις Μετρητοίς / Cash

Transit: Shipment:

VAT status: Normal Currency: 999 EURO TR ex. rate: 1 TRP ex. rate: 1

Comment: Appr.: Yes ☒ Incl.: No ☐ Inted: No

Items

	Code	Name	Qty 1	Price	Disc. 1(%)	Value
5	33122104	ALCOHOL METER AT-2000	1	0,00		0,00
6	331223P02	ALCOHOL SENSOR FOR ALCOSCAN AL-60	1	30,00		30,00
7	33270304P01	ALUMINUM 40x60cm	1	24,00		24,00
8	792421	ALKALINE BATTERY LR20 MAXIMA SUNLIC	1	3,51		3,51

Name Filter: AL*

Total qty: 9,00 Disc. 1(%): 0,00 Disc. 1: 0,00 Disc. 2(%): 0,00

Net value: 168,21 VAT value: 35,32 Expenses: 0,00 Total value: 203,53

Figure CS5.2 - Lines display after using the filter

Select fields

Locally calculated fields. Πίνακας : Sales documents

Available functions

Call	Caption	Operation	Field type	Calculation formula	Displayed
V1	Name Filter	Calculation	Alphanumeric		Yes

Calculation formula (Detailed)

Accept Cancel

Figure CS5.3 - The user-defined field must be inserted in FINDOC (Sales documents) table

2) Button that adds the calculated price as line price in item lines.

Create a virtual table in database designer named CCCITEMPURDOC with fields as in Figure CS6.1.

Figure CS6.1 - Virtual Table Fields

In sales documents create a new panel containing a datagrid with fields from the virtual table you created. Add also a new button (Figure CS6.2).

Figure CS6.2 - Sales View – Virtual table design

In form script enter the following code that finds the latest purchases for the selected line item and displays them in a datagrid of a pop up window.

```

var vIncrement; //Increment percentage
function ON_MYPOPUPFORM_SHOW() {
    vIncrement = 40;
    GetItemPurdoc(); //Gets the purchases for the selected Item
    CCCVTITEMPUR.FIRST;
}

function InsPRCtoMTRLINES(vTable) { //Insert Price to Mtrlines
    ITELINES.PRICE = vTable.PRICE;
}

function GetItemPurdoc() { //Last 20 Purchases for selected item
    strqry="SELECT TOP 20 A2.TRDR, A1.MTRLINES, A1.FINDOC,"
        +"\n      (A1.LINEVAL/case A1.QTY1 when 0 then 1 else  A1.QTY1 end) AS LINEVAL,"
    A1.OTY1"

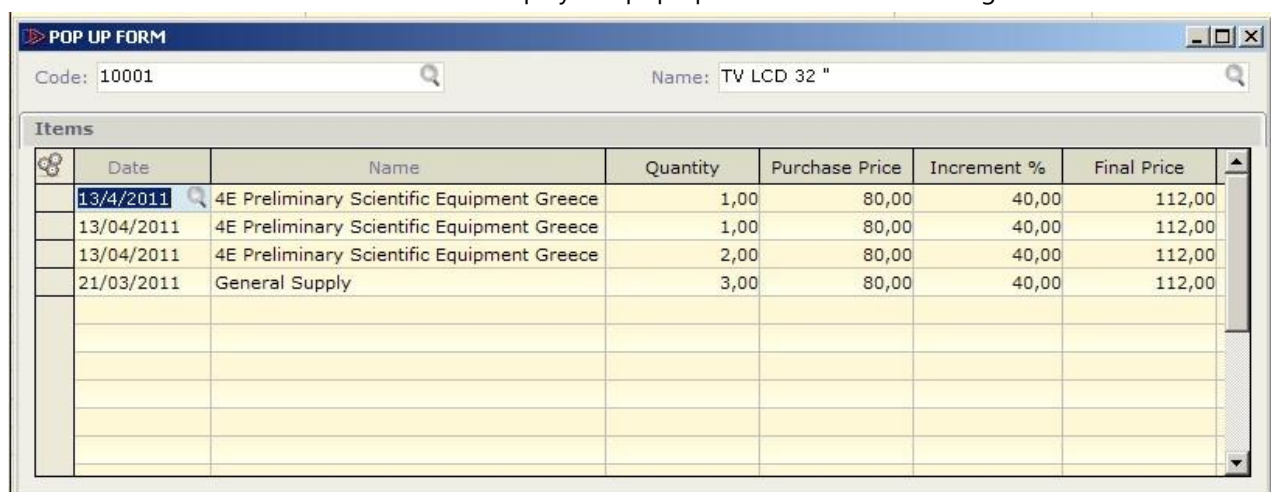
```

```

+" \n FROM MTRLINES A1 INNER JOIN FINDOC A2 ON A1.FINDOC=A2.FINDOC AND
A1.COMPANY=A2.COMPANY"
+" \n WHERE "
+" \n A2.SOSOURCE=1251 "
+" \n AND A2.TFPRMS IN(102,103) "
+" \n AND A1.COMPANY="+X.SYS.COMPANY
+" \n AND A1.MTRL="+ITELINES.MTRL
+" \n ORDER BY A2.TRNDATE DESC";
ds=X.GETSQLDATASET(strqry, '');
CCCVTITEMPUR.FIRST;
while (CCCVTITEMPUR.RECORDCOUNT > 0) {
    CCCVTITEMPUR.DELETE;
}
if (ds.RECORDCOUNT>0) {
    try {
        dsLink = X.EXEC('CODE:ModuleIntf.GetDataset', X.MODULE, 'CCCVTITEMPUR');
        X.EXEC('CODE:ModuleIntf.SetDatasetLinks', X.MODULE, dsLink, 0);
//Disables Links
        while (!ds.EOF()) {
            CCCVTITEMPUR.APPEND;
            CCCVTITEMPUR.TRDR = ds.TRDR;
            CCCVTITEMPUR.FINDOC = ds.FINDOC;
            CCCVTITEMPUR.MTRLINES = ds.MTRLINES;
            CCCVTITEMPUR.PRICESTART = ds.LINEVAL;
            CCCVTITEMPUR.INCREMENT = vIncrement;
            CCCVTITEMPUR.QTY = ds.QTY1;
            CCCVTITEMPUR.POST;
            ds.NEXT;
        }
    }
    finally {
        X.EXEC('CODE:ModuleIntf.SetDatasetLinks', X.MODULE, dsLink, 1); //Enables
Links - Sends at the end 1 query for all
    }
}
}
function ON_CCCVTITEMPUR_INCREMENT() { //On change of the increment percentage the
final price will be updated
    CCCVTITEMPUR.PRICE = CCCVTITEMPUR.PRICESTART + (CCCVTITEMPUR.PRICESTART *
CCCVTITEMPUR.Increment /100);
}

```

Double click on each line of the document to display the pop up window as shown in Figure D6.3



Date	Name	Quantity	Purchase Price	Increment %	Final Price
13/4/2011	4E Preliminary Scientific Equipment Greece	1,00	80,00	40,00	112,00
13/04/2011	4E Preliminary Scientific Equipment Greece	1,00	80,00	40,00	112,00
13/04/2011	4E Preliminary Scientific Equipment Greece	2,00	80,00	40,00	112,00
21/03/2011	General Supply	3,00	80,00	40,00	112,00

Figure CS6.3 - Pop up Window from double clicking on the item lines

7. HTTP Request – Google Maps Distance

Module: Sales documents

Job: Use Google maps API to calculate distance and shipping cost.

Distance is calculated in km and uses the address and the postal code of the "WHouse" that is in the header of the document (MTRDOC.WHOUSE) and the shipment address and postal code of the document.

There is also a button "Show map" that displays the distance in Google maps web page.

Design

Create a new form in sales documents and design panels with the following buttons and fields:

Button «Calc Trans Cost»(RUNB_700001=Calc Ship Cost)

Virtual field that will be used to calculate the distance from Google Maps.

Fields SALDOC.NUM01 «Distance(Km)» and SALDOC.NUM02 «Duration(min)»

Fields that will be updated from Google Maps (button Calc Ship Cost).

Fields SALDOC.NUM03 «Cost/Km» and SALDOC.NUM04 «Total Cost»

Fields that will be calculated by field Distance.

Button «Add to Expenses»(RUNB_700002=Add to Expenses)

Virtual field that will be used to insert a new record in table expenses with the value of the field "Total Cost" (SALDOC.NUM04).

Button «ShowMap»(RUNB_700003=ShowMap)

Virtual field that will be used to open the google maps web page with the selected SoftOne values.

The form will look as in figure CS7.1. In the form script insert the code (JavaScript) of the next pages.

The screenshot shows the SAP Sales Document form for Standard Company: OFFR000018- 24/05/2018. The form is divided into several sections:

- General data:** Includes fields for Series (7001OFFR), Type (7001), Quotation, Document (OFFR000018), Number (18), Date (24/05/2018), Branch (1000), Head Offices, Warehouse (1000), Central, and various other details like Customer (200), Salesperson (00007), Movement type, Shipment (103), and VAT Status.
- Calculate Transfer Cost (Directions from Google):** This section contains a button labeled "Calc Trans Cost" (1), a field for Distance(km) (2) with a value of 8,67, a field for Duration(min) (5) with a value of 14,48, a button labeled "ShowMap" (5), a field for Cost/Km (3) with a value of 0,10, a field for Total Cost (4) with a value of 0,87, and a button labeled "Add to Expenses" (4).
- Items:** A table titled "Purchased Items" showing 10 items with columns for Code, Description, Qty 1, Retail Pr, Disc. ..., and Value. The items include TV LED 32 LG RTDXA32, TV LCD 21, Digital Camera 8 MP, Photo michachi DSLR 10 MP, Flash Camcorder, Flash Full HD Camcorder, HDD 60GB Camcorder, Digital Photo Frame 8", Home cinema 1000 Watt, and GPS.
- Gross:** A summary section at the bottom showing Total qty (40,00), Net (22.489,38), Disc. 1(%) (0,04), VAT (5.172,74), Disc. 1 (10,00), Expenses (0,87), Disc. 2(%) (0,00), and Total (27.662,99).

Figure CS7.1

How it works

When user clicks on button "Calc Ship Cost" then distance will be calculated using:

From address:

Address and postal code are retrieved from the "WHouse" selected in the document. The country is retrieved from the branch of the document.

To address:

Calculated from the fields shipping address (MTRDOC.SHIPPINGADDS) and shipping postal code (MTRDOC.SHPZIP) from the sales document.

After the calculation the fields Distance(km) and Duration(min) are updated with data from Google Maps. Also the field Cost/km is updated with the value 0.10 if the km are below 50 or else with the value 0.20.

Finally, the field Total Cost is calculated from the fields Distance and Cost/km.

Clicking on button «Add to Expenses» creates a new record in table expenses using the expense with code 101 and the value of the field Total Cost.

```
function distancematrix(from, to) {
    this.from= from || "";
    this.to = to || "";
    this.onSuccess = function(){};
    this.onNoReply = function(){};
    this.onError= function(){};
    this.request= null;
    var me= this,
    url;
    this.execute =
    function() {
        from = encodeURIComponent(from);
        to   = encodeURIComponent(to);
        url =
'http://maps.googleapis.com/maps/api/distancematrix/json?sensor=false&origins='
        + from + '&destinations=' + to;
        this.request = new ActiveXObject("MSXML2.XMLHTTP");
        this.request.open('GET', url, true);
        this.request.onreadystatechange=
        function() {
            if (me.request.readyState==4) {
                if (me.request.status==200)
                    me.onSuccess(me.request.responseText);
                else
                    me.onNoReply();
            }
        }
        try {
            this.request.send();
        }
        catch(e) {
            this.onError(e);
        }
    }
}
```



```

function GetDirections() {
    var distance=0;
    var duration=0;
    str = " SELECT ADDRESS, ZIP"
        +"\n , (SELECT NAME FROM COUNTRY "
        +"\n      WHERE COUNTRY=(SELECT COUNTRY FROM BRANCH "
        +"\n                        WHERE COMPANY="+SALDOC.COMPANY
        +"\n                        AND BRANCH="+SALDOC.BRANCH+") ) AS COUNTRY"
        +"\n FROM WHOUSE "
        +"\n WHERE COMPANY="+SALDOC.COMPANY+" AND WHOUSE="+MTRDOC.WHOUSE;
    ds = X.GETSQLDATASET(str,null);
    var from = ds.ADDRESS + ',' + ds.ZIP + ',' + ds.COUNTRY,
    to      = MTRDOC.SHIPPINGADDR + ',' + MTRDOC.SHPZIP + ',' + ds.COUNTRY;
    dm      = new distancematrix(from,to);
    dm.onSuccess =
    function(reply) {
        eval('var data = ' + reply);
        if (data.status === 'OK') {
            distance = roundNumber((data.rows[0].elements[0].distance.value / 1000),2);
            duration = roundNumber((data.rows[0].elements[0].duration.value / 60),2);
            SALDOC.NUM01 = distance;
            SALDOC.NUM02 = duration;
        }
        else {
            X.WARNING("Google maps API Error: " + data.status + "\n\nError Message: "+
data.error_message);
        }
    };
    dm.execute();
}

function roundNumber(num, dec) {
    var result = Math.round(num*Math.pow(10,dec))/Math.pow(10,dec);
    return result;
}

function EXECCOMMAND(cmd) {
    if (cmd=='700001')
        GetDirections();
    if (cmd=='700002') {
        X.FOCUSFIELD('SALDOC.NUM01');
        AddtoExpenses();
    }
    if (cmd==700003)
        ShowMap();
}

function AddtoExpenses() {
    if(SALDOC.NUM04 != '') {
        EXPANAL.APPEND;
        EXPANAL.EXPEN = 101;
        EXPANAL.EXPVAL = SALDOC.NUM04;
        EXPANAL.POST;
    }
}

function ON_SALDOC_NUM01() {
    CalculateShipCost();
}

```

```

function CalculateShipCost() {
    if(SALDOC.NUM01 <= 50) {
        SALDOC.NUM03 = 0.10;
    }
    else {
        SALDOC.NUM03 = 0.20;
    }
    SALDOC.NUM04 = SALDOC.NUM03 * SALDOC.NUM01;
}

function ShowMap() {
    str = " SELECT ADDRESS"
        +"\n , ZIP"
        +"\n , (SELECT NAME FROM COUNTRY "
        +"\n WHERE COUNTRY=(SELECT COUNTRY FROM BRANCH "
        +"\n WHERE COMPANY="+SALDOC.COMPANY
        +"\n AND BRANCH="+SALDOC.BRANCH+") ) AS COUNTRY"
        +"\n FROM WHOUSE "
        +"\n WHERE COMPANY="+SALDOC.COMPANY+" AND WHOUSE="+MTRDOC.WHOUSE;
    ds = X.GETSQLDATASET(str,null);
    var from = ds.ADDRESS + ',' + ds.ZIP + ',' + ds.COUNTRY,
    var to = MTRDOC.SHIPPINGADDR + ',' + MTRDOC.SHPZIP + ',' + ds.COUNTRY;

    if ( (MTRDOC.SHIPPINGADDR == '') || (MTRDOC.SHPZIP == '') || (ds.ADDRESS == '') ||
(ds.ZIP == '')){
        X.EXCEPTION('Not valid data');
    }

    from = from.replace(' ','+');
    to = to.replace(' ','+');

    var shellObj = new ActiveXObject("WScript.Shell");
    var aCommand = 'chrome
http://maps.google.com/maps?saddr='+from+'&daddr='+to+'&hl=en';
    shellObj.Run(aCommand,true);
}

```

8. Run External Application

Module: Stock Items

Job: Run external application on button click, using the data of the field ITEEXTRA.VARCHAR01 as command line arguments.

```
function EXECCOMMAND(cmd) {
    if (cmd == 201909081) {
        RunExternalApplication();
    }
}

function RunExternalApplication() {
    var oShell = new ActiveXObject("Shell.Application");
    var sFile = "C:\\SoftoneVersions\\Soft1Current\\Xplorer.exe";
    var vArguments = "/" + ITEEXTRA.VARCHAR01;
    var vDirectory = "";
    var vOperation = "open";
    var vShow = "1";
    oShell.ShellExecute(sFile, vArguments, vDirectory, vOperation, vShow);

    // == vShow Parameters == //
    // https://docs.microsoft.com/en-us/windows/desktop/shell/shell-shellexecute
    //0=Open the application with a hidden window.
    //1=Open the application with a normal window. If the window is minimized or
    maximized, the system restores it to its original size and position.
    //2=Open the application with a minimized window.
    //3=Open the application with a maximized window.
    //4=Open the application with its window at its most recent size and position. The
    active window remains active.
    //5=Open the application with its window at its current size and position.
    //7=Open the application with a minimized window. The active window remains active.
    //10=Open the application with its window in the default state specified by the
    application.
}
```

9. Add Related Jobs in SoftOne Objects

Module: Sales Documents

Job: Add two custom related jobs in Sales Documents. The first job executes a Dialog object, that is previously created in SoftOne Database Designer and the second one is handled by the EXECCOMMAND function.

```
function ON_CREATE() {
    AddRelativeJob();
}

function EXECCOMMAND(cmd) {
    if (cmd == 201809091) {
        X.WARNING("Custom Relative Job 2 clicked");
        //Add code here (open sub form, run scripts, etc.)
    }
}

function AddRelativeJob() {
    var fObjectName = 'SALDOC';
    var fCmd = '';

    fCmd = 'CCCMYDIALOGOBJECT'; //Dialog object inside SoftOne Database Designer
    if (RelJobExists(fObjectName, fCmd) == 0) //Check to prevent double entries
        X.EXEC('CODE:SysRequest.InsRelativeJob', fObjectName, 'Rel Job 1', fCmd);

    fCmd = '201809091';
    if (RelJobExists(fObjectName, fCmd) == 0)
        X.EXEC('CODE:SysRequest.InsRelativeJob', fObjectName, 'Rel Job 2', fCmd);
}

function RelJobExists(fObjectName, fRelJobCmd) {
    var fRelJobExists = 0;
    var fRelJob = '';
    var i = 0;

    while (i == 0 || fRelJob != '') {
        fRelJob = X.EXEC('CODE:SysRequest.GetRelativeJob', fObjectName, i);
        if (fRelJob.indexOf(fRelJobCmd) > 0)
            fRelJobExists = 1;
        i = i + 1;
    }
    return fRelJobExists;
}
```

10. Run SBSL script without UI (filters screen)

Module: Sales Documents

Job: Auto-execute SBSL script after posting a sales documents record without displaying the script filters screen (use the default script filter values).

```
function ON_AFTERPOST() {  
    RunSBSLScript();  
}  
  
function RunSBSLScript () {  
    var obj = X.CREATEOBJFORM('CLIENTIMPORT2,SCRIPTNAME:MYSBSLSCRIPT');  
    obj.BATCHEXECUTE;  
}
```

Note that you can also pass parameters to the script (see section [E.4 Run using Parameters](#)).

```
X.CREATEOBJFORM('CLIENTIMPORT,SCRIPTNAME:MYSCRIPT,par1:10,par2:'+SALDOC.CCCMYFIELD);
```

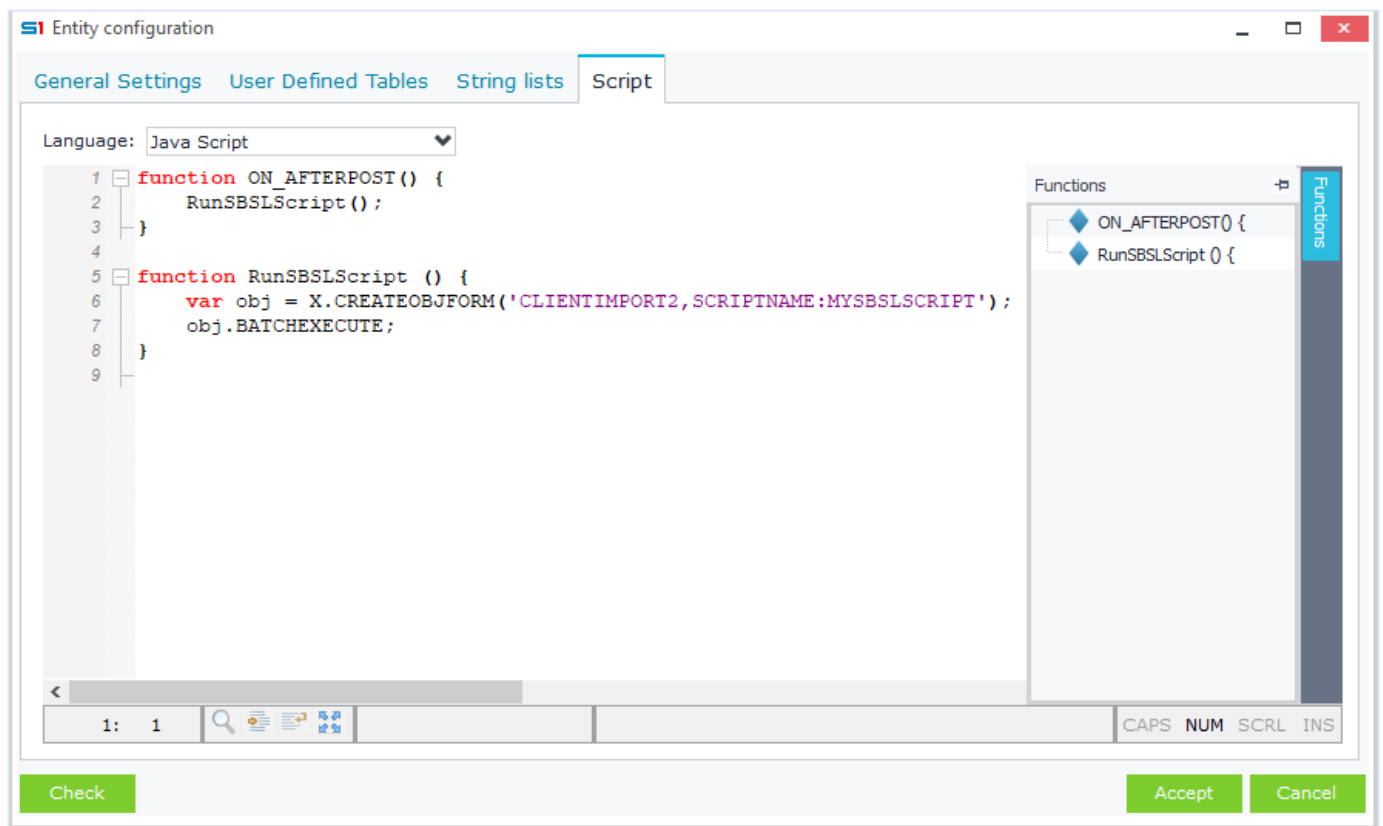


Figure CS10.1 – Run SBSL script without UI

11. Auto apply ABC model.

Module: Supplier Other Transactions

Job: Auto apply ABC **model** in lines of "Supplier Other Transactions". The ABC Model that is applied in LINLINES is taken either from LINEITEM object or from PRNSIN, depending on lines data.

The function that applies the ABC Model is the following:

X.CALLPUBLISHED('ProgLibIntf.ModuleCommand', X.MODULE, 10120, sCommand)

10120 is an internal command that applies ABC Model using table name and ABC dimension model IDs from the parameter sCommand.

sCommand = S1Tablename+';'+dim1ABCModelID+';'+dim2ABCModelID+';'+dim3ABCModelID +';'+dim4+';'+dim5+';'+dim6+';'+dim7+';'+dim8+';'+dim9+';'+dim10;

Example: sCommand = 'LINLINES;10;43;0;0;0;0;0;0;0;0;';

```
function ON_POST() {
    CheckToAUTOCreateABC();
    //Gets ABC Model from PRSNIN or LINEITEM (if LINLINES.PRSN is null)
}

function CheckToAUTOCreateABC() {
    LINLINES.DISABLECONTROLS();
    FINDOC.NOTCALC = 1;
    try {
        LINLINES.FIRST;
        while(!LINLINES.EOF) {
            if ( (LINLINES.LINEVAL != 0)
                && (LINLINES.ISNULL('MTRL') == 0)
                && (LINLINES.ISNULL('SALESMAN') == 1) )
                //53=Debits/Credits-->Model from object LINEITEM
                AutoFillABCfromModel('LINLINES',LINLINES.MTRL,53);
            if ( (LINLINES.LINEVAL != 0)
                && (LINLINES.ISNULL('MTRL') == 0)
                && (LINLINES.ISNULL('SALESMAN') == 0) )
                //20=Company employees --> Model from object PRSNIN
                AutoFillABCfromModel('LINLINES',LINLINES.SALESMAN,20);
            LINLINES.NEXT;
        }
    }
    finally {
        FINDOC.NOTCALC = 0;
        LINLINES.ENABLECONTROLS();
    }
}
```

```

function AutoFillABCfromModel(S1Tablename, iID, iSodtype) {
//ABC Dimensions (1,2,13,14) are defined in ABC Settings --> Replace with your own
str = ";WITH cte AS"
      +" (SELECT 1 x"
      +" UNION ALL"
      +" SELECT x + 1"
      +" FROM cte"
      +" WHERE x < 10)"
      +" SELECT x,ISNULL(a.ABCDIMMDL,0) AS ABCDIMMDL"
      +" FROM cte c "
      +" LEFT OUTER JOIN ABCMDLMASTER a on c.x = (CASE a.DIMENSION "
      +"      WHEN 1 THEN 1 "
      +"      WHEN 2 THEN 2 "
      +"      WHEN 13 THEN 3 "
      +"      WHEN 14 THEN 4 "
      +"      ELSE 0 END)"
      +" AND a.COMPANY="+X.SYS.COMPANY
      +" AND a.ABCDTYPE="+iSodtype
      +" AND a.MASTERID="+iID
      +" ORDER BY 1";
ds = X.GETSQLDATASET(str,null);
var strModels = '';
if (ds.RECORDCOUNT > 0) {
  ds.FIRST;
  while (!ds.EOF) {
    strModels += ';' + ds.ABCDIMMDL ;
    ds.NEXT;
  }
  sCommand = S1Tablename + strModels; //X.WARNING("sCommand="+sCommand);
  //sCommand = S1Tablename+';'+dim1+';'+dim2+';'+dim3+';'+dim4+';'+dim5+';'+
+dim6+';'+dim7+';'+dim8+';'+dim9+';'+dim10;
  X.CALLPUBLISHED('ProgLibIntf.ModuleCommand', X.MODULE, 10120, sCommand);
}
}

```

The screenshot shows the SAP Parameters configuration window for Activity Based Costing. The left sidebar contains a navigation menu with the following items: Financials, Cash Accounts & Ban..., Other Transactions, International Trans..., Accounting, Cost Accounting, Revenue & Expenses, Fixed Assets, Activity Based Costing, and Parameters. The Parameters item is highlighted. The main window displays the 'Parameters' configuration for 'User-defined dimensions'. It includes five input fields for Dimension 1 through 5. Dimension 1 is set to 'Department' and Dimension 2 is set to 'Cost Centers'. Below these, the 'Dimensions' field is set to '1,2,13,14', 'Allocation type' is 'Per Dimension', 'Update entries derived from commercial documents' is checked 'Yes', and 'Delete during reversal' is unchecked 'No'.

Figure CS11.1 – ABC Dimension Parameters

12. SubMenu in Button

Module: Sales Documents

Job: Create and show a sub menu when user clicks on a button (id "20202020") and display warning messages when sub menu items are clicked.

Design

Create a form in sales documents and add a new button: **RUNB_20202020=Button with SubMenu** (Figure CS12.2). In Configuration window add a new StringList named "**MyButtonMenu**" and insert two items as in Figure CS12.3. This string list defines the sub menu items that will be displayed on button click. The event that fires when users click on these items is the **EXECCOMMAND**.

```
function EXECCOMMAND(cmd) {  
  if (cmd == 20202020) {  
    var mysubmenu = X.EXEC('CODE:ModuleIntf.FindXStrings', X.MODULE, 'MyButtonMenu');  
    var mypopup = X.EXEC('CODE:SysRequest.CreateApplPopup', X, 0);  
    X.EXEC('CODE:SysRequest.ShowApplPopup', mypopup, mysubmenu);  
  }  
  else if (cmd == 20202021) {  
    X.WARNING("Button SubMenu 1 Clicked");  
  }  
  else if (cmd == 20202022) {  
    X.WARNING("Button SubMenu 2 Clicked");  
  }  
}
```

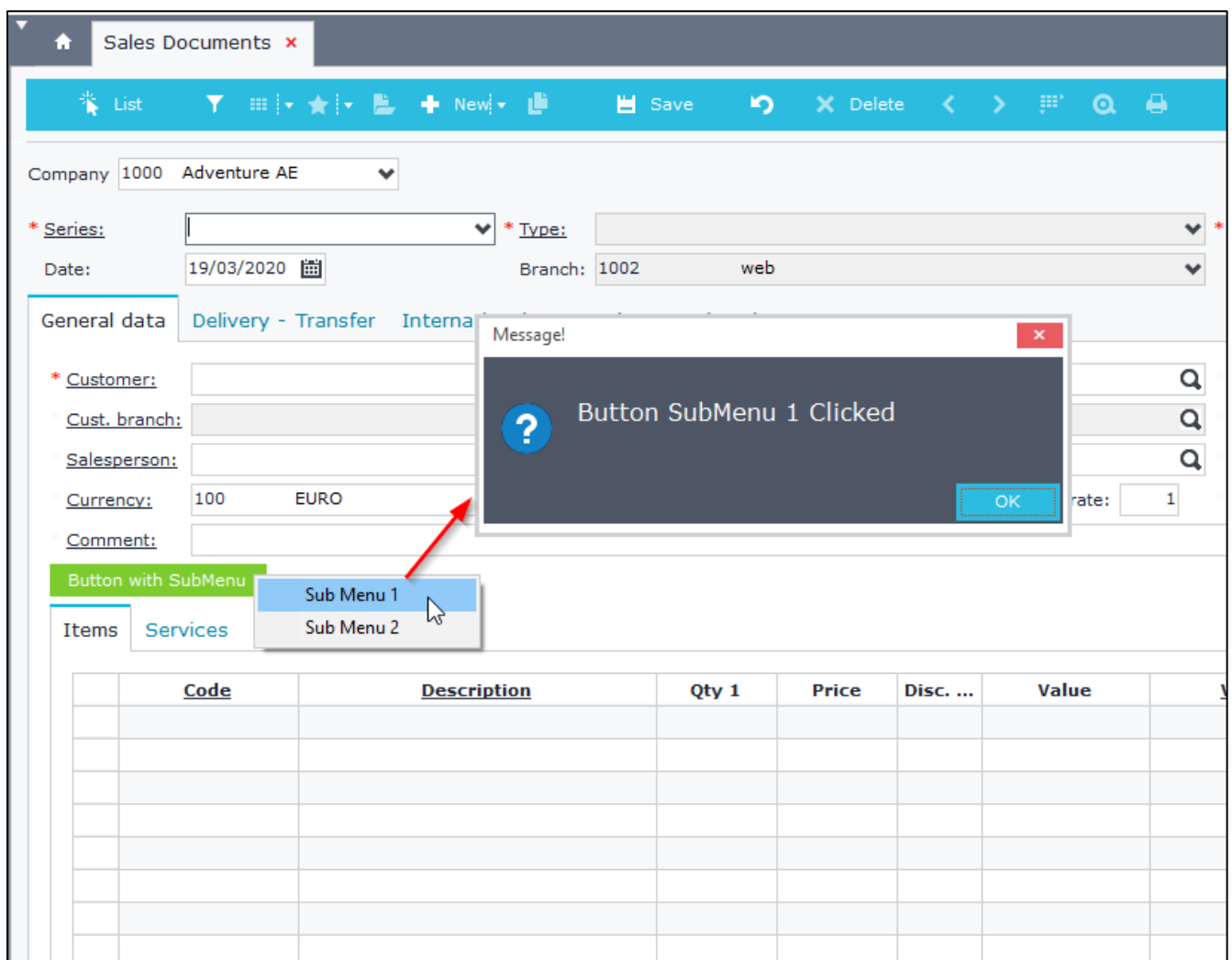


Figure CS12.1 – Submenu on button click

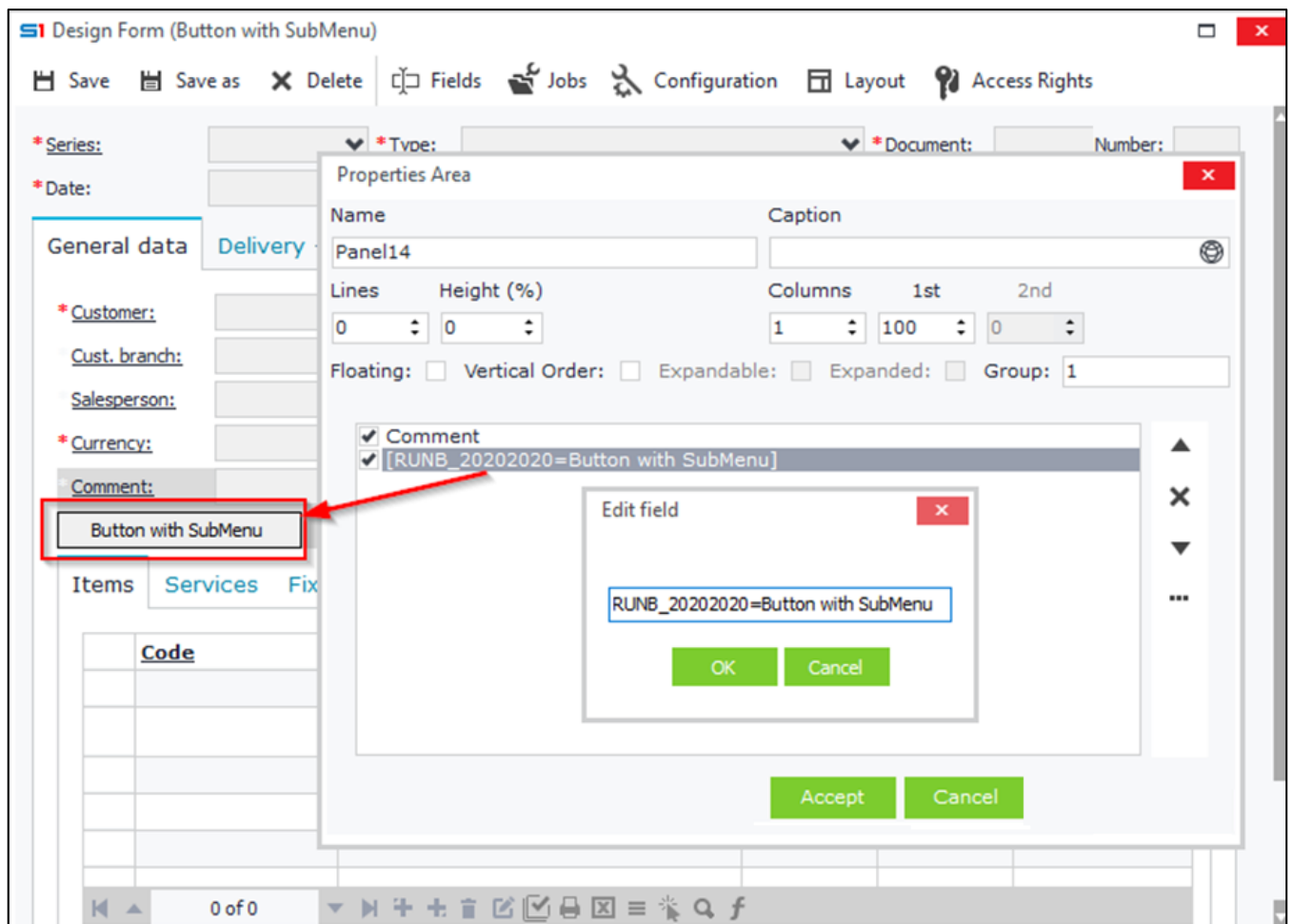


Figure CS12.2 – Button Properties in Design Form

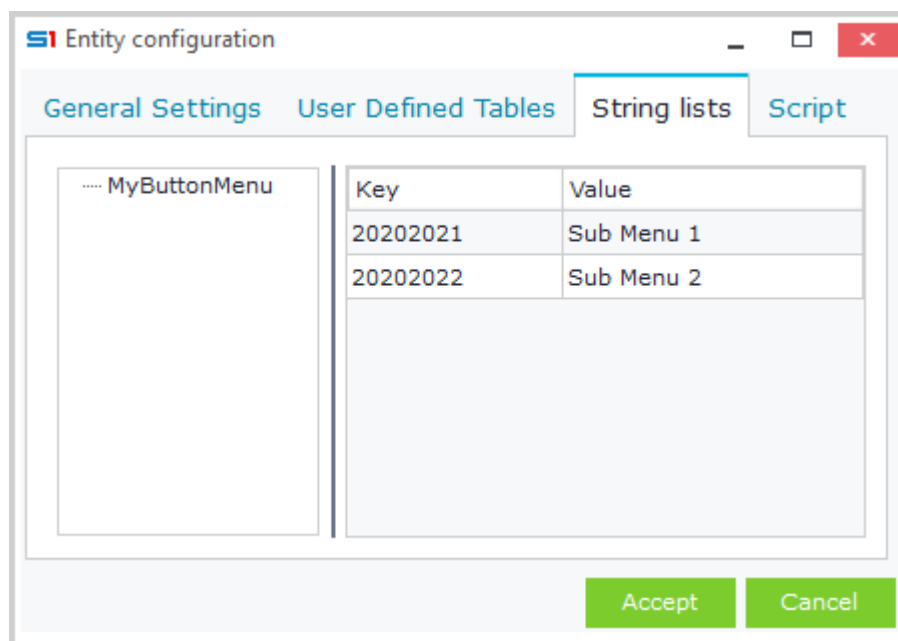


Figure CS12.3 – SubMenu Items (created as StringList)

13. ASCII Export / Import encoded file

Module: Sales Documents

Job: Button 1 that creates an output encoded Ascii file, that contains data from item lines (Code,Name,Qty1,Price,Lineval) and uses specific delimiter.

Button 2 that reads Ascii file using specific codepage and displays the data in message boxes.

Design

Create a form inside sales documents and add two new buttons:

RUNB_20200729=ASCII Export Lines and **RUNB_20200730=ASCII Read Lines** (Figure CS13.1).

Add the following code inside the tab "Script" of the "Configuration" window.

```
function EXECCOMMAND(cmd) {
    if (cmd == 20200729){
        AsciiExportLines();
    }
    if (cmd == 20200730){
        AsciiReadLines();
    }
}

function AsciiExportLines() {
    var vCodePage = 65001; //UTF-8=65001, ANSI Greek=1253,
    var vTextWrite = X.EXEC("CODE:PILib.CreateText", 'C:\\Temp\\AsciiExport.txt',
vCodePage);
    var vLine = "";
    var vSep = "|";
    ITELINES.DISABLECONTROLS;
    try {
        ITELINES.FIRST;
        while (!ITELINES.EOF) {
            vLine = ITELINES.X_CODE + vSep + ITELINES.X_NAME + vSep + ITELINES.QTY1 +
vSep + ITELINES.PRICE + vSep + ITELINES.LINEVAL;
            X.CALLPUBLISHED('PILib.WriteLine', vTextWrite, vLine);
            ITELINES.NEXT;
        }
        X.EXEC('CODE:PILib.CloseText', vTextWrite);
    }
    catch (e) {
        X.WARNING(e.message);
    }
    finally {
        ITELINES.ENABLECONTROLS;
    }
    X.WARNING("Data Exported succesfully");
}

function AsciiReadLines() {
    var vTextFile = X.EXEC('CODE:PILib.OpenText', 'C:\\Temp\\AsciiFile.txt', 65001);
    var vLine = "";
    try {
        while (!X.EXEC('CODE:PILib.EOF', vTextFile)) {
            vLine = X.EXEC('CODE:PILib.ReadLine', vTextFile);
            X.WARNING(vLine);
        }
    }
    catch (e) {
        X.WARNING(e.message);
    }
    finally {
        X.EXEC('CODE:PILib.CloseText', vTextFile);
    }
}
```

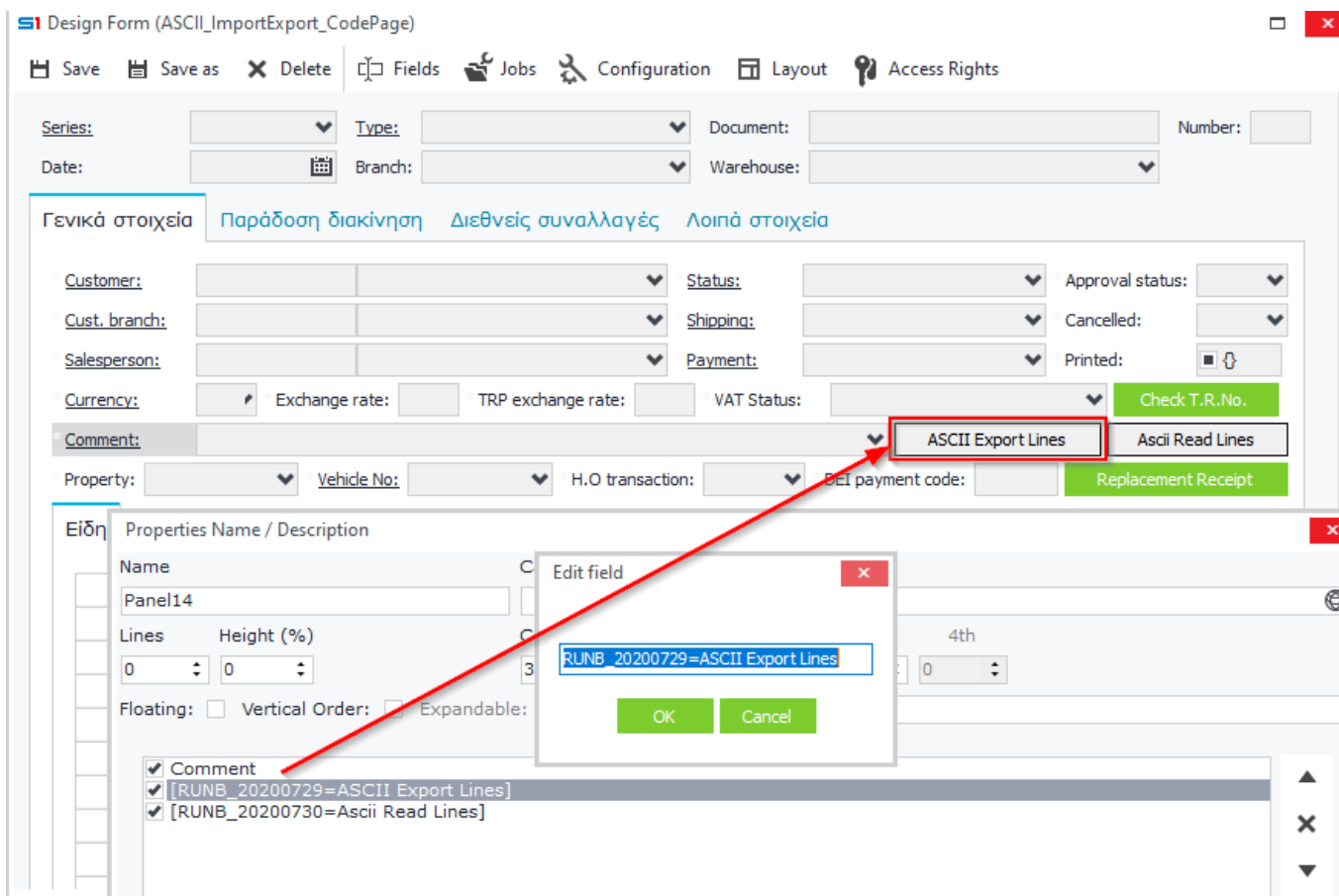


Figure CS13.1 – Button “ASCII Export Lines”

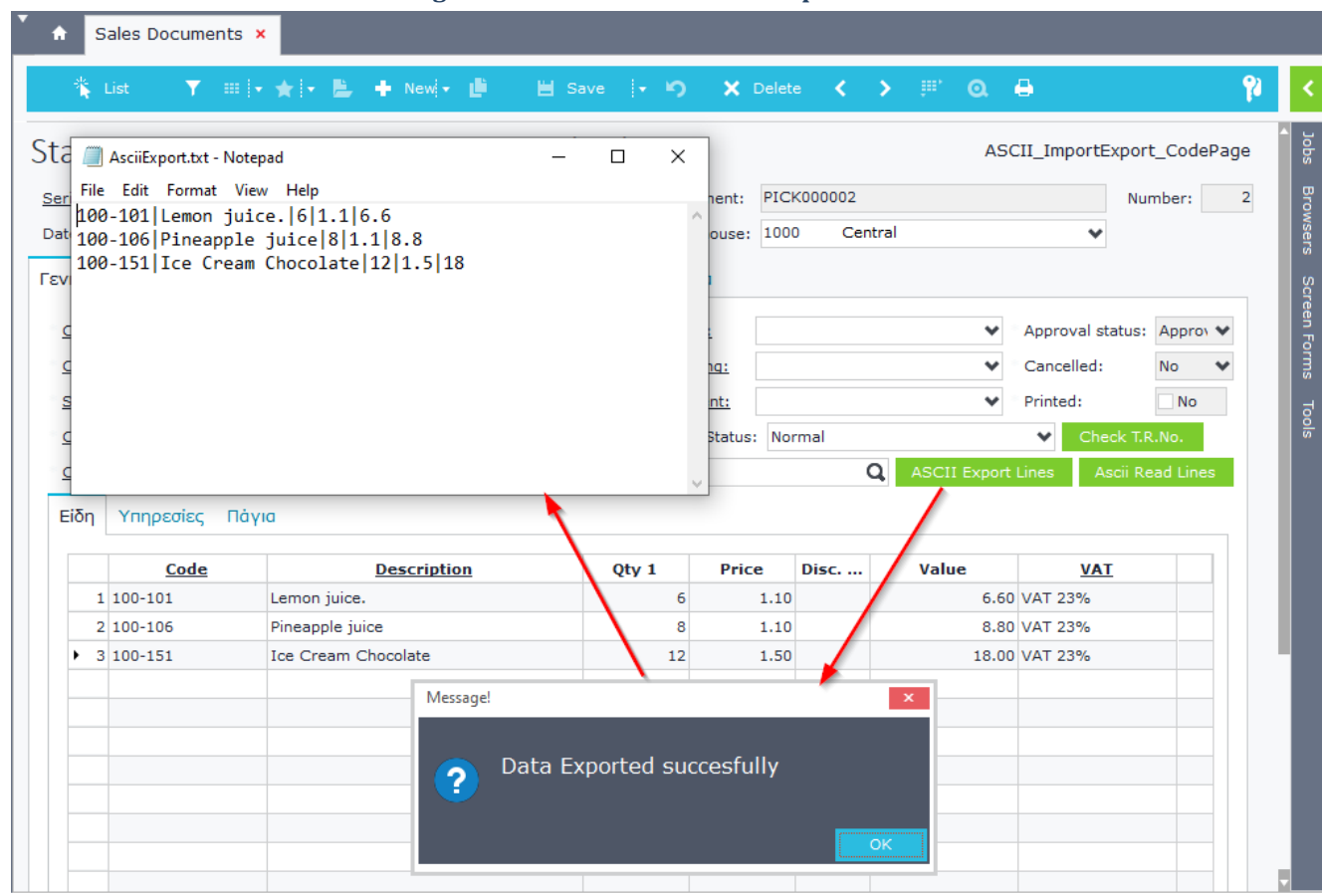


Figure CS13.2 – Ascii Export - Job Results

14. Get Browser field data

Module: Customers

Job: Browser right-click option that retrieves the selected browser records and displays the data of all the fields (**including user-defined ones**). The following example fires only on browser “WS_MultiResultsSQL” (Figure CS14.1), which uses the user-defined fields (Debit, Credit) created in section [Multi Results SQL](#).

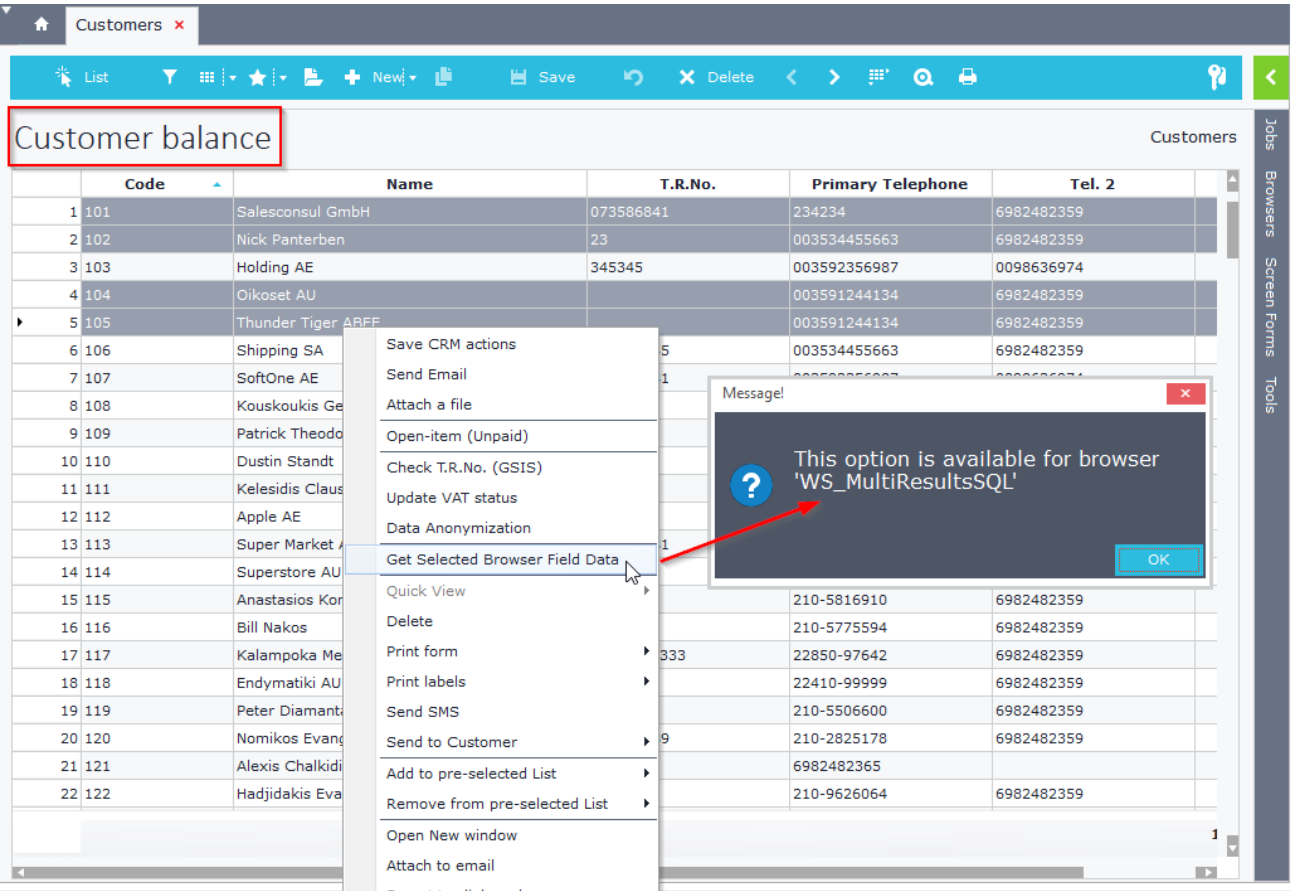


Figure CS14.1 – Job runs only on specific browser

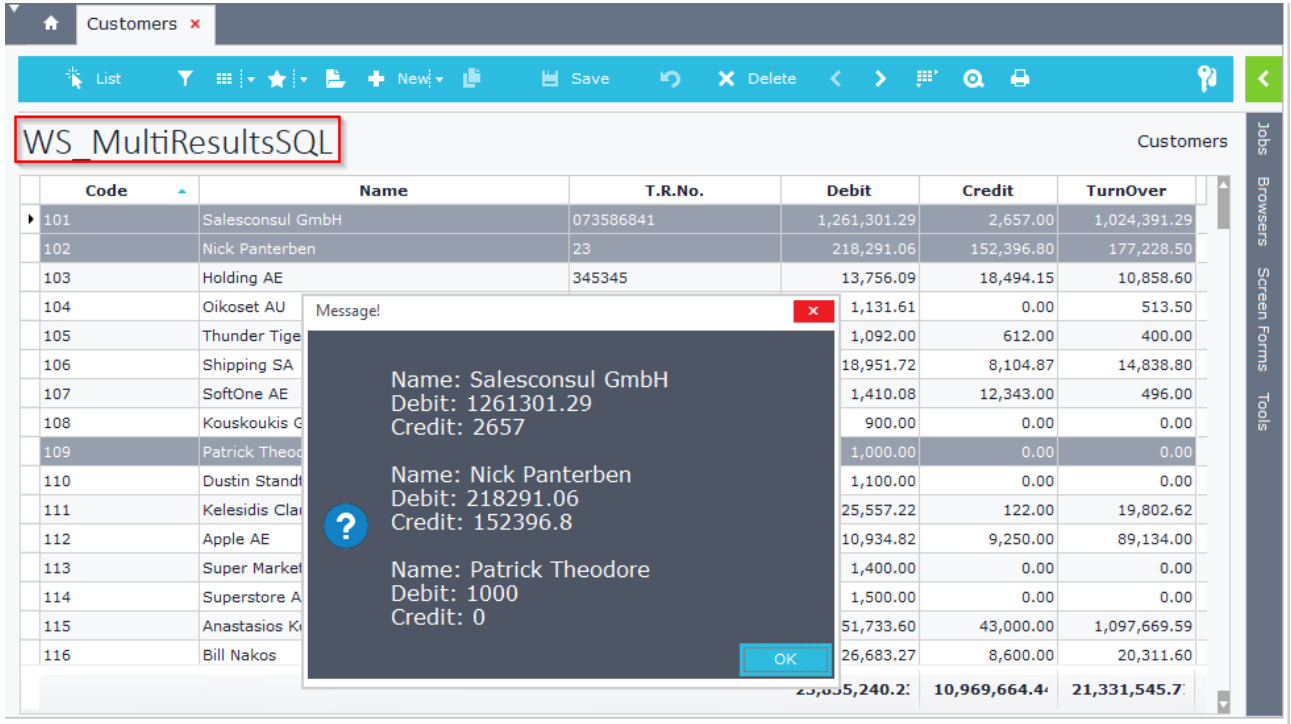


Figure CS14.2 – Selected records data

Design

Create an Advanced JavaScript module for object CUSTOMER (e.g. *CustomerJSCode*) and add a reference in *JSMain* module (Figure CS14.3). Open *CustomerJSCode* and enter the following code.

```
AddCode ('CUSTOMER', 'CustomerJSCode');
```

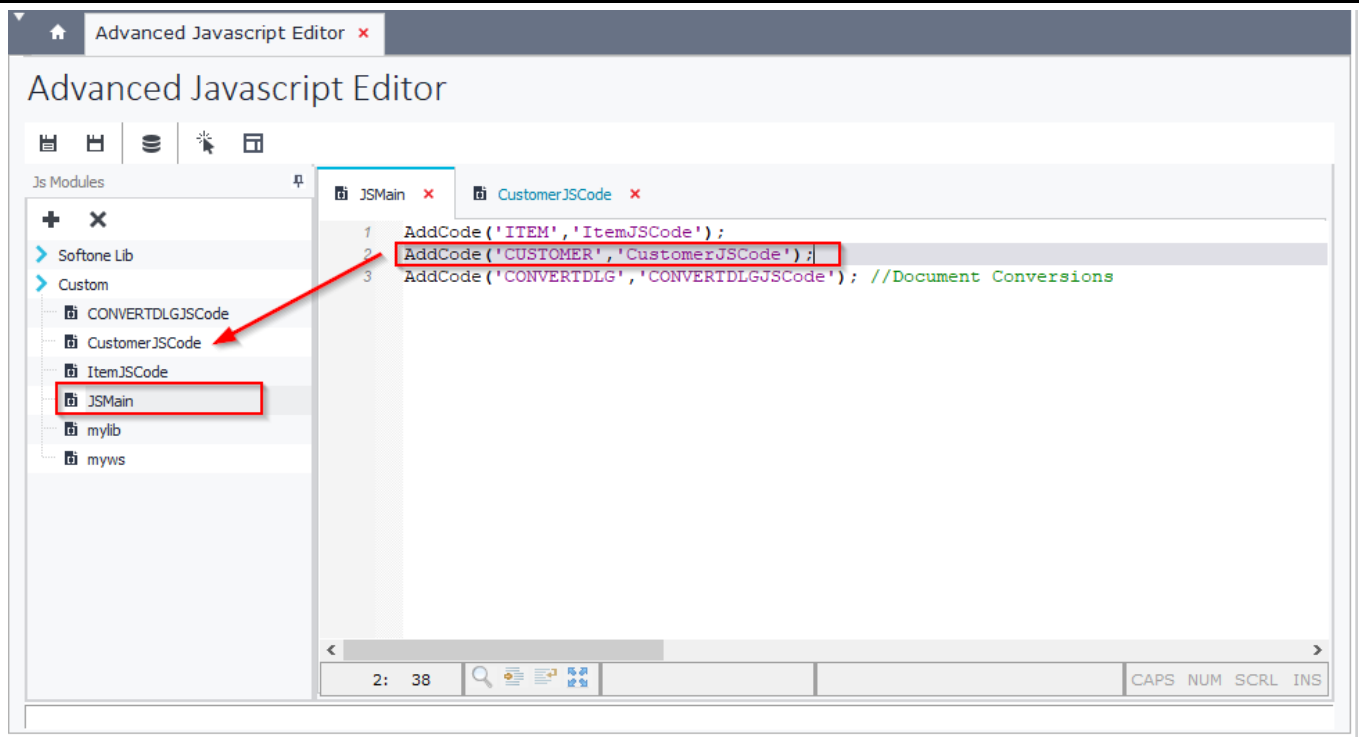


Figure CS14.3 – Ascii Export - Job Results

How it works

Browser data are stored in a dataset (e.g. *dsBrowser*) using the function *GetDataset*:

```
var dsBrowser = X.EXEC('CODE:ModuleIntf.GetDataset', X.Module, 'BROWSER');
```

Field values are retrieved using the function *GetFieldValue* (e.g. field *TRDR*):

```
fldTRDR = X.EXEC('CODE:ModuleIntf.GetFieldValue', dsBrowser, 'TRDR');
```

Iteration through all Browser records can be achieved using the following

```
X.EXEC('CODE:ModuleIntf.DatasetFirst', dsBrowser); //Move to the first record
while (X.EXEC('CODE:ModuleIntf.DatasetEof', dsBrowser) == 0) { //Iterate All Browser
Data
    fldTRDR = X.EXEC('CODE:ModuleIntf.GetFieldValue', dsBrowser, 'TRDR');
    fldName = X.EXEC('CODE:ModuleIntf.GetFieldValue', dsBrowser, 'NAME');
    X.EXEC('CODE:ModuleIntf.DatasetNext', dsBrowser);
}
```

In our example we store the selected records (*SELRECS*) in a variable (*vSelRecs*) and then we create a dataset (*vds*) that contains the selected record ids.

Then we iterate over its data and locate the related *dsBrowser* records using the following:

```
if (X.EXEC('CODE:ModuleIntf.DatasetLocate', dsBrowser, 'TRDR', vds.TRDR) == 1) {
    fldTRDR = X.EXEC('CODE:ModuleIntf.GetFieldValue', dsBrowser, 'TRDR');
    //...
    vds.NEXT;
}
```

Full Code

```
function ON_CREATE() {
    var vBrowserMenu = X.EXEC('CODE:ModuleIntf.FindXStrings', X.MODULE, 'BRMENU');
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '---');
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '20200903=1;Get Browser Data');
    X.EXEC('CODE:SysRequest.RefreshPopupMenu', X.MODULE, 'BRMENU', 1);
}

function EXECCOMMAND(cmd) {
    if (cmd == 20200903) {
        if (X.LIST == "WS_MultiResultsSQL")
            GetBrowserFieldData();
        else
            X.WARNING("This option is available only for browser 'WS_MultiResultsSQL'");
    }
}

function GetBrowserFieldData() {
    var vSelRecs;
    var fldTRDR;
    var fldName;
    var fldDebit;
    var fldCredit;
    var SelData = "";
    try {
        vSelRecs = X.GETPARAM('SELRECS');
        vSelRecs = vSelRecs.replace(/\?/g, ",");

        var dsBrowser = X.EXEC('CODE:ModuleIntf.GetDataset', X.Module, 'BROWSER');
        //Browser dataset
        var vds = X.GETSQLDATASET('SELECT DISTINCT TRDR FROM TRDR WHERE ' + vSelRecs,
null);
        vds.FIRST;
        while (!vds.EOF()) {
            if (X.EXEC('CODE:ModuleIntf.DatasetLocate', dsBrowser, 'TRDR', vds.TRDR) == 1) {
                fldTRDR = X.EXEC('CODE:ModuleIntf.GetFieldValue', dsBrowser, 'TRDR');
                fldName = X.EXEC('CODE:ModuleIntf.GetFieldValue', dsBrowser, 'NAME');
                fldDebit = X.EXEC('CODE:ModuleIntf.GetFieldValue', dsBrowser, 'Debit');
                fldCredit = X.EXEC('CODE:ModuleIntf.GetFieldValue', dsBrowser, 'Credit');

                fldDebit = S1P.Round(fldDebit, 2);
                fldCredit = S1P.Round(fldCredit, 2);

                SelData += "Name: " + fldName + "\nDebit: " + fldDebit + "\nCredit: " +
fldCredit + "\n\n";
            }
            vds.NEXT;
        }
        if (SelData.length > 0)
            SelData = SelData.slice(0, -2); //Remove last 2 characters
        X.WARNING(SelData);
        // X.EXEC('CODE:ModuleIntf.DatasetFirst', dsBrowser); //Move to the first record
        // while (X.EXEC('CODE:ModuleIntf.DatasetEof', dsBrowser)==0) { //Iterate All
Browser Data
        //     fldTRDR = X.EXEC('CODE:ModuleIntf.GetFieldValue', dsBrowser, 'TRDR');
        //     fldName = X.EXEC('CODE:ModuleIntf.GetFieldValue', dsBrowser, 'NAME');
        //     X.EXEC('CODE:ModuleIntf.DatasetNext', dsBrowser);
        // }
    } catch (e) {
        X.WARNING("Error: " + e.message);
    }
    return SelData;
}
```

15. Add / Retrieve HTML data

Job: Functions that locate a SoftOne Object record (e.g. SOEMAIL) and add or retrieve HTML data from a blob field (e.g. SOMAIL.SOBODY).

They can also be used inside a [custom Advanced JS library](#) and included in any form script (Figures below).

```
function GetHTMLFromField(ObjectName, RecordID, TableName, FieldName) {
    var strHTML = "";
    try {
        var S1Module = X.EXEC('CODE:ModuleIntf.CreateModule', ObjectName);
        X.EXEC('CODE:ModuleIntf.LocateModule', S1Module, RecordID);
        var vdsTable = X.EXEC('CODE:ModuleIntf.GetDataSet', S1Module, TableName);
        var vBlobField = X.EXEC('CODE:ModuleIntf.GetField', vdsTable, FieldName);
        var vBlobPointer = X.EXEC('CODE:SOHTMLDOC.CreateDoc', 2, vBlobField);
        strHTML = X.EXEC('CODE:SOHTMLDOC.GetDocHTMLPart', vBlobPointer);
        X.EXEC('CODE:SOhtmlDoc.DestroyDoc', vBlobPointer);
        X.EXEC('CODE:ModuleIntf.DestroyModule', S1Module);
    } catch (e) {
        X.WARNING(e.message);
    }
    return strHTML;
}

function AddHTMLToField(ObjectName, RecordID, TableName, FieldName, HTMLCode) {
    try {
        var S1Module = X.EXEC('CODE:ModuleIntf.CreateModule', ObjectName);
        X.EXEC('CODE:ModuleIntf.LocateModule', S1Module, RecordID);
        var vdsTable = X.EXEC('CODE:ModuleIntf.GetDataSet', S1Module, TableName);
        var vBlobField = X.EXEC('CODE:ModuleIntf.GetField', vdsTable, FieldName);
        var vBlobPointer = X.EXEC('CODE:SOHTMLDOC.CreateDoc', 1, HTMLCode);
        X.EXEC('CODE:ModuleIntf.DatasetEdit', vdsTable);
        X.EXEC('CODE:SOHTMLDOC.SaveDoctoBlob', vBlobPointer, vBlobField);
        X.EXEC('CODE:ModuleIntf.DatasetPost', vdsTable);
        X.EXEC('CODE:ModuleIntf.PostModule', S1Module);
        X.EXEC('CODE:SOhtmlDoc.DestroyDoc', vBlobPointer);
        X.EXEC('CODE:ModuleIntf.DestroyModule', S1Module);
    } catch (e) {
        X.WARNING(e.message);
    }
}
```

How it works

Both functions use SBSL methods to create the object (CreateModule) and locate the record (LocateModule). The following methods are used to add HTML data to a blob field.

```
var vBlobPointer = X.EXEC('CODE:SOHTMLDOC.CreateDoc', 1, HTMLCode);
X.EXEC('CODE:SOHTMLDOC.SaveDoctoBlob', vBlobPointer, vBlobField);
```

The following methods are used to retrieve HTML data from a blob field.

```
var vBlobPointer = X.EXEC('CODE:SOHTMLDOC.CreateDoc', 2, vBlobField);
strHTML = X.EXEC('CODE:SOHTMLDOC.GetDocHTMLPart', vBlobPointer);
```

Use the following functions when you need to get/set blob fields for a current module (already located in a record). Replace S1Module with X.Module and remove CreateModule, LocateModule, PostModule and DestroyModule.

```
function GetHTMLFromModuleField(TableName, FieldName) {
    var vdsTable = X.EXEC('CODE:ModuleIntf.GetDataSet', X.MODULE, TableName);
    var vBlobField = X.EXEC('CODE:ModuleIntf.GetField', vdsTable, FieldName);
    var vBlobPointer = X.EXEC('CODE:SOHTMLDOC.CreateDoc', 2, vBlobField);
    var strHTML = X.EXEC('CODE:SOHTMLDOC.GetDocHTMLPart', vBlobPointer);
    X.EXEC('CODE:SOhtmlDoc.DestroyDoc', vBlobPointer);
    return strHTML;
}

function AddHTMLToModuleField(TableName, FieldName, HTMLCode) {
```

```

var vdsTable = X.EXEC('CODE:ModuleIntf.GetDataSet', X.MODULE, TableName);
var vBlobField = X.EXEC('CODE:ModuleIntf.GetField', vdsTable, FieldName);
X.EXEC('CODE:ModuleIntf.DatasetEdit', vdsTable);
var vBlobPointer = X.EXEC('CODE:SOHTMLDOC.CreateDoc', 1, HTMLCode);
X.EXEC('CODE:SOHTMLDOC.SaveDocToBlob', vBlobPointer, vBlobField);
X.EXEC('CODE:SOhtmlDoc.DestroyDoc', vBlobPointer);
}

```

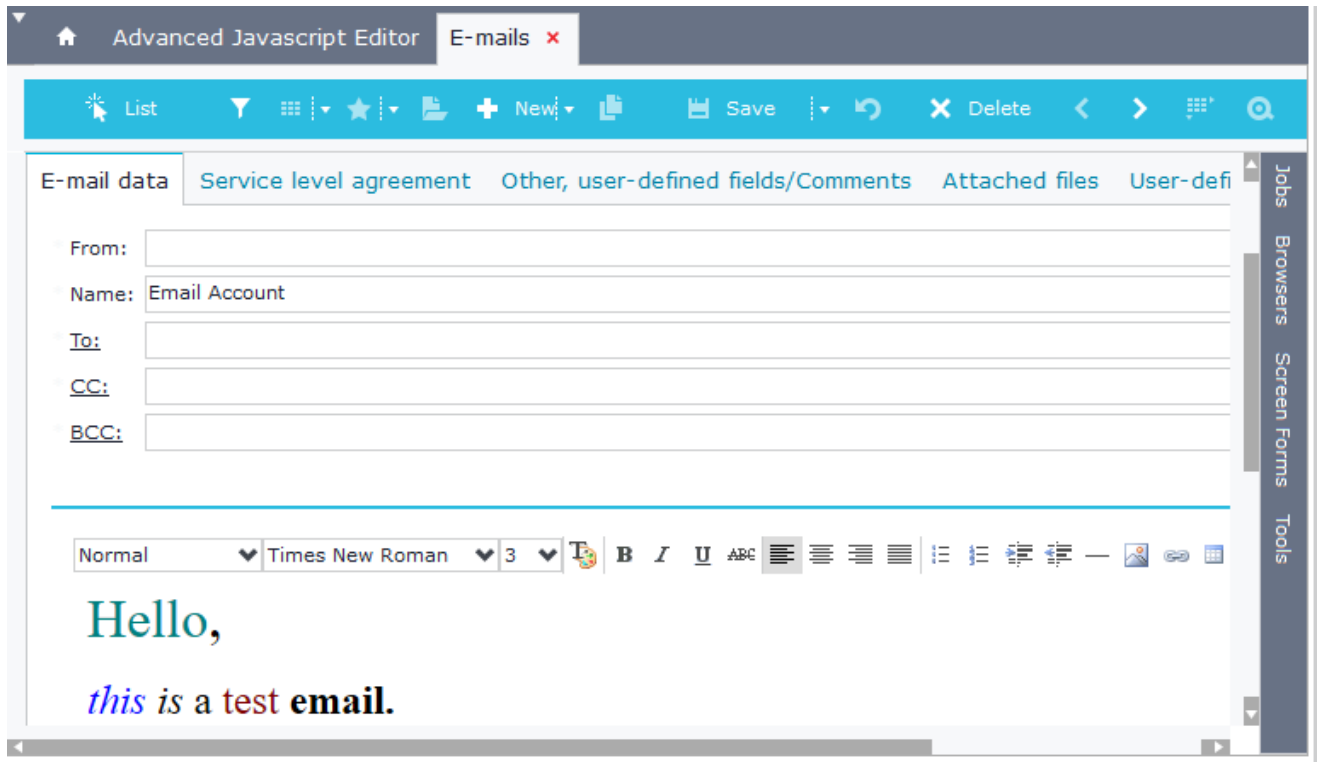


Figure CS15.1 – Email record (ID=3358)

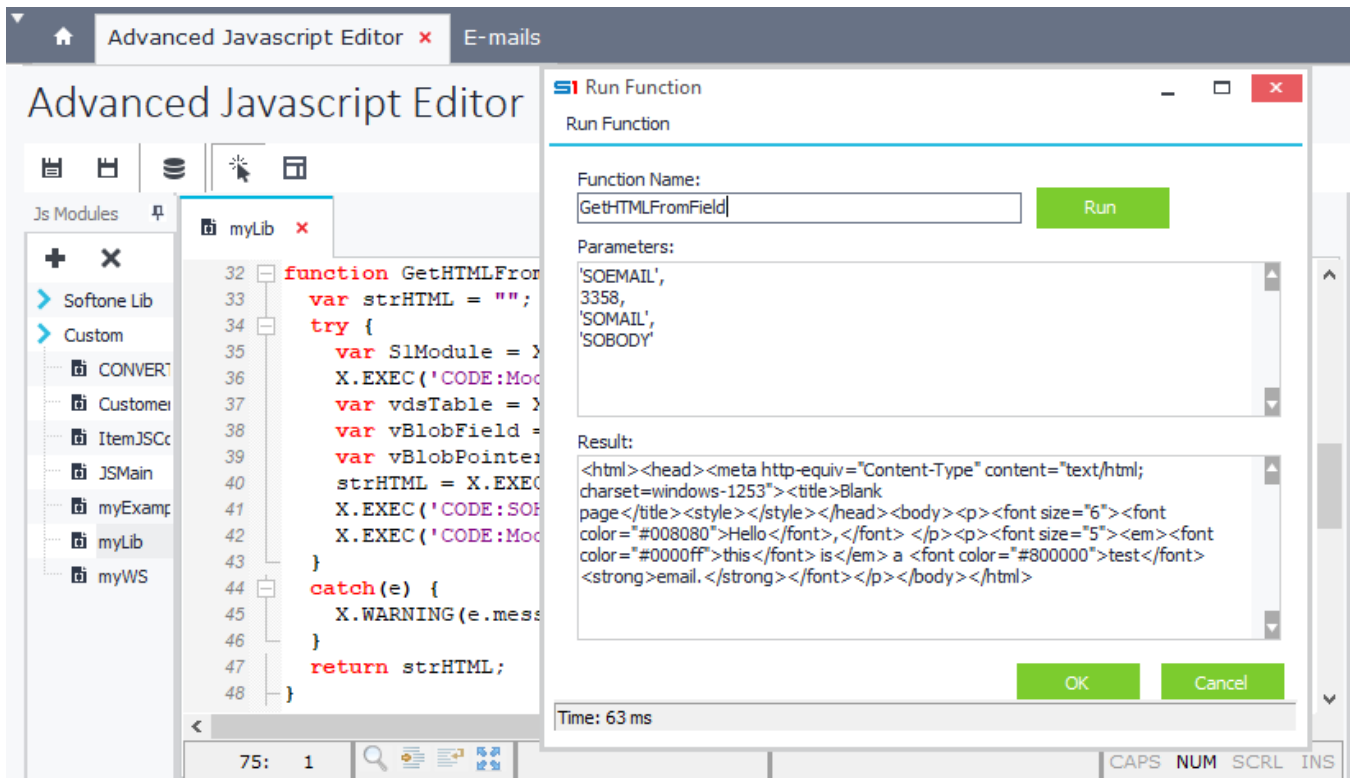


Figure CS15.2 – Get HTMLData using GetHTMLFromField from Advanced JavaScript

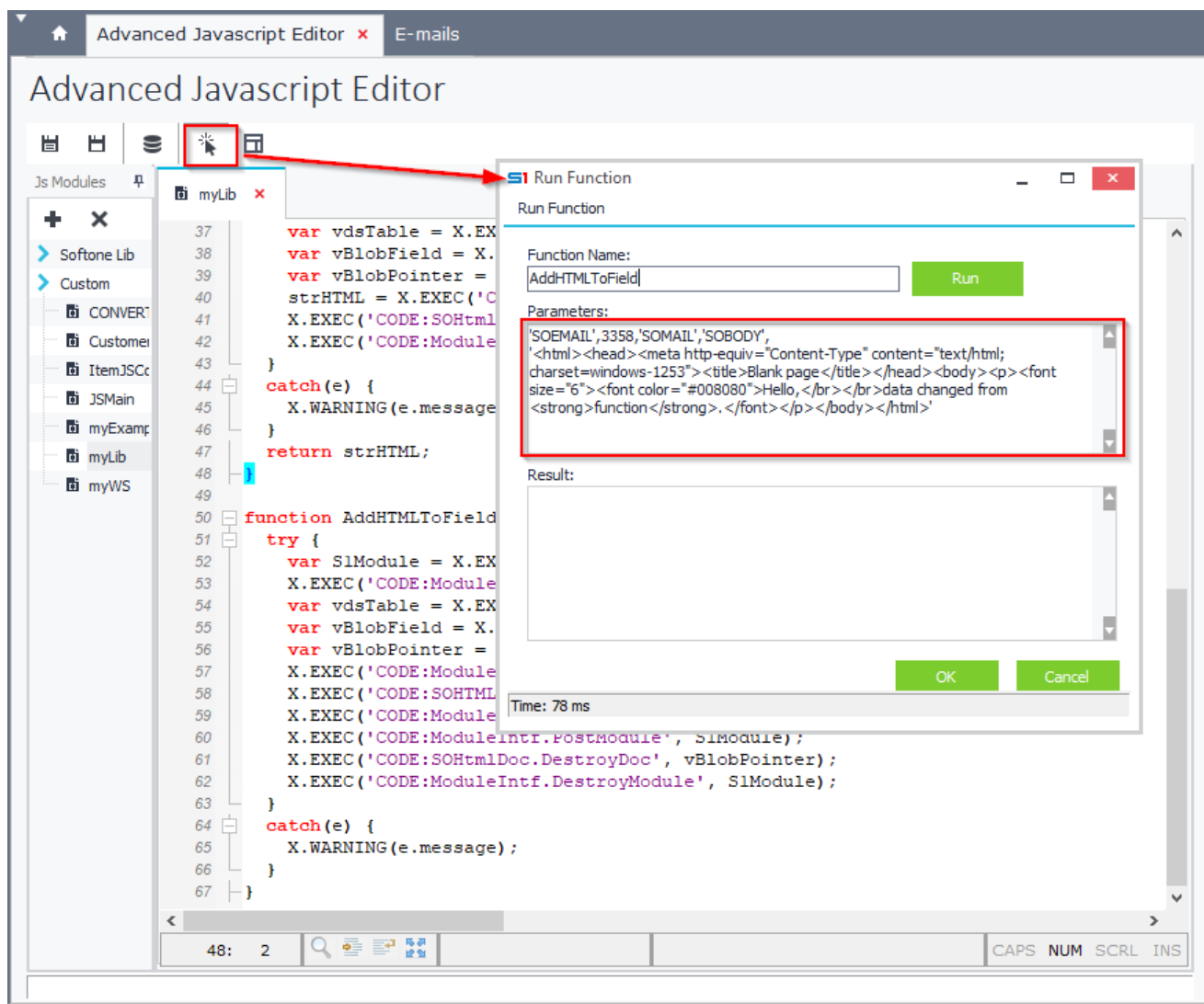


Figure CS15.3 – Add HTMLData using AddHTMLToField

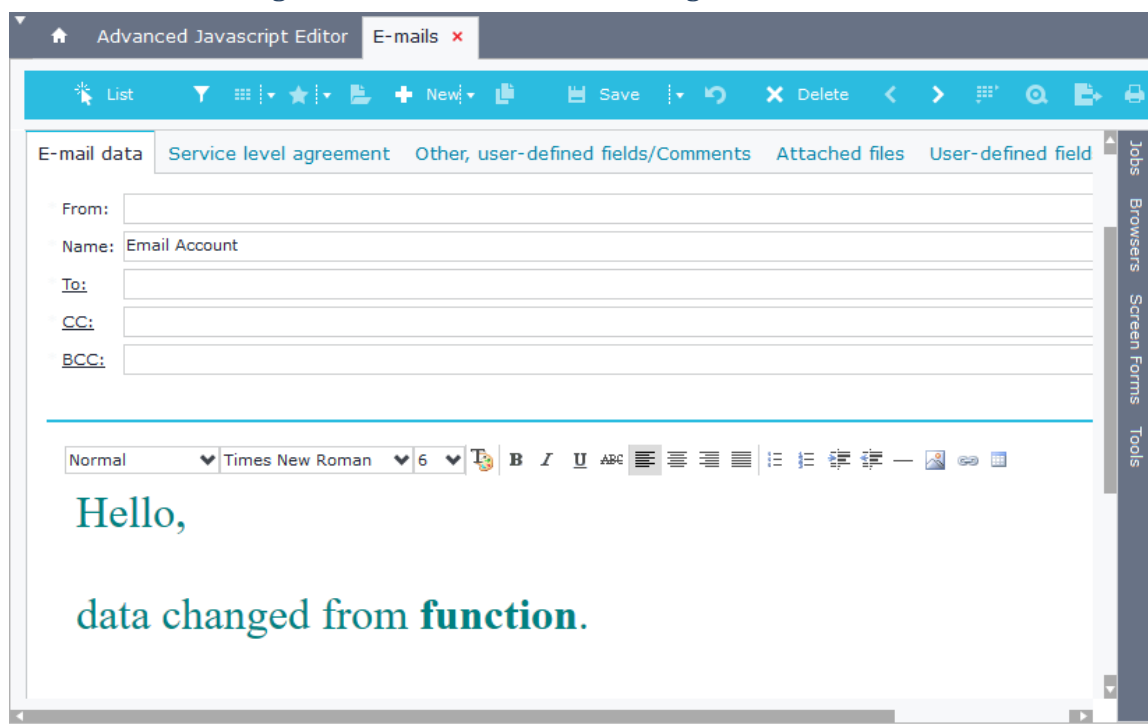


Figure CS15.4 – Email record (New HTML data)

16. Check / Add / Delete ABC (Activity Based Costing) data

Job: Function "CheckABCData" checks if users have entered data in ABC Lines.

Function "DeleteAllABCData" deletes all ABC Lines. Function "AddABCData" adds data in ABC Lines.

```
function GetObjectABC(vModule, vLineObject) {
    var ptrABC = X.EXEC('CODE:ModuleIntf.GetChildModule', X.MODULE, 'ABCTRNLINES');
    //try to get the childmodule ABCTRNLINES
    if (ptrABC <= 0) {
        X.CALLPUBLISHED('ProgLibIntf.ModuleCommand', X.MODULE, 10122, vLineObject);
        //Activates the ChildModule when users have not already opened it at least
one time (right-click "Activity Based Costing")
        ptrABC = X.EXEC('CODE:ModuleIntf.GetChildModule', X.MODULE, 'ABCTRNLINES');
    }
    return X.GetObject(ptrABC); //returns the ABC object, which contains all the data
of the table ABCTRNLINES (e.g. objABC.ABCTRNLINES.ABCST1)
}

function CheckABCData() {
    var objABC = GetObjectABC(X.MODULE, 'LINLINES');
    // X.WARNING(ABCobj.ABCTRNLINES.RECORDCOUNT);
    if (objABC.ABCTRNLINES.RECORDCOUNT == 0)
        X.EXCEPTION('Please fill in the ABC data for all the document lines!');
}

function DeleteAllABCData() {
    var objABC = GetObjectABC(X.MODULE, 'LINLINES');
    LINLINES.FIRST;
    while (!LINLINES.EOF) {
        DeleteABCData(objABC);
        LINLINES.NEXT;
    }
}

function DeleteABCData(objABC) {
    objABC.VPAGEFIELD.EDIT;

    objABC.VPAGEFIELD.PAGEFIELD = 0;
    while (!objABC.VDIMTRNLINES1.EOF)
        objABC.VDIMTRNLINES1.DELETE;

    objABC.VPAGEFIELD.PAGEFIELD = 1;
    while (!objABC.VDIMTRNLINES2.EOF)
        objABC.VDIMTRNLINES2.DELETE;

    objABC.VPAGEFIELD.PAGEFIELD = 2;
    while (!objABC.VDIMTRNLINES3.EOF)
        objABC.VDIMTRNLINES3.DELETE;
}

function AddABCData() {
    var objABC = GetObjectABC(X.MODULE, 'LINLINES');
    objABC.VPAGEFIELD.PAGEFIELD = 0;

    objABC.VDIMTRNLINES1.APPEND;
    objABC.VDIMTRNLINES1.ABCST = 101;
    objABC.VDIMTRNLINES1.COEF = 70;
    objABC.VDIMTRNLINES1.POST;

    objABC.VDIMTRNLINES1.APPEND;
    objABC.VDIMTRNLINES1.ABCST = 102;
    objABC.VDIMTRNLINES1.COEF = 30;
    objABC.VDIMTRNLINES1.POST;
}
```

10. DATA FLOWS

- A. [Overview](#)
- B. [Data Flow Rules](#)
- C. [Data Flow Scenarios](#)
- D. [Execute Data Flows from Screen Scripts](#)
- E. [Case Studies](#)

A. Overview

Data flows is a process tool for transferring data between SoftOne modules, using a graphical interface and writing few lines of code. They are usually executed using flow scenarios, but there is also a function to run them through form scripts. Flow scenarios are context menu items that are displayed in browsers right click.

This very powerful tool allows you to create data-transfer batch jobs without the knowledge and use of SBSL scripts. Results log of the jobs is stored in the database table named **TRNFTRC**.

Figure A1 displays an example of data flow rule design, that creates customers from draft entries. The source entity contains the object SODRAFT (1) and the target entity the object CUSTOMER (2).

The target entity “Customers” (3) in which data will be imported, is set to retrieve data from the source table “Draft entries” (4). Datagrid displays the value assignments in the target fields.

The screenshot shows the 'Data Flow Rules' configuration window for a rule named '1025- Draft Entry to Customer'. The interface includes a toolbar with actions like List, Save, Delete, and a sidebar with navigation options like Jobs, Browsers, Screen Forms, and Tools.

Source entity: Name: SODRAFT, Description: 'Draft entries', View: (dropdown).

Data tables: A table with one entry: 'Draft entries - Draft entry link'. To the right are buttons for 'Grouping', 'Filters', and 'New table'.

Target entity: Name: CUSTOMER, Description: 'Customers', View: (dropdown), Tables: 1.

Customers: Data source: 'Draft entries - Draft entry link'. Below is a table for 'Assign field values':

Field	Expression	Aggregation
Code	{SODRAFT.CODE}	None
Name	{SODRAFT.NAME}	None
Primary Address	{SODRAFT.ADDRESS}	None
Zip	{SODRAFT.ZIP}	None
Location area	{SODRAFT.DISTRICT}	None
City	{SODRAFT.CITY}	None
Prefecture	{SODRAFT.DISTRICT1}	None
Primary Telephone	{SODRAFT.PHONE1}	None

At the bottom left is an 'Advanced' button.

Figure A1

The job is created and displayed in the draft entries browser through a flow scenario (Figure A2). Figure A3 shows the pop-up menu that appears upon right clicking on the draft entries browser.

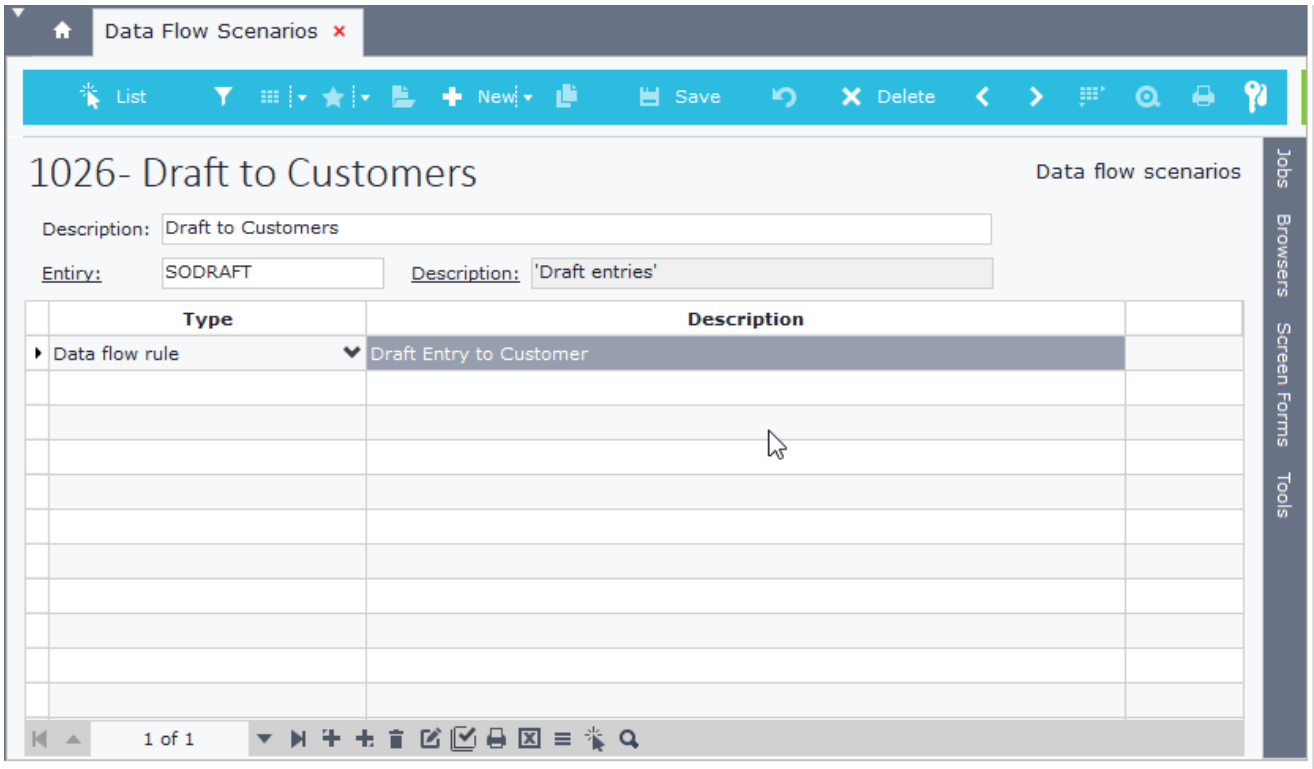


Figure A2

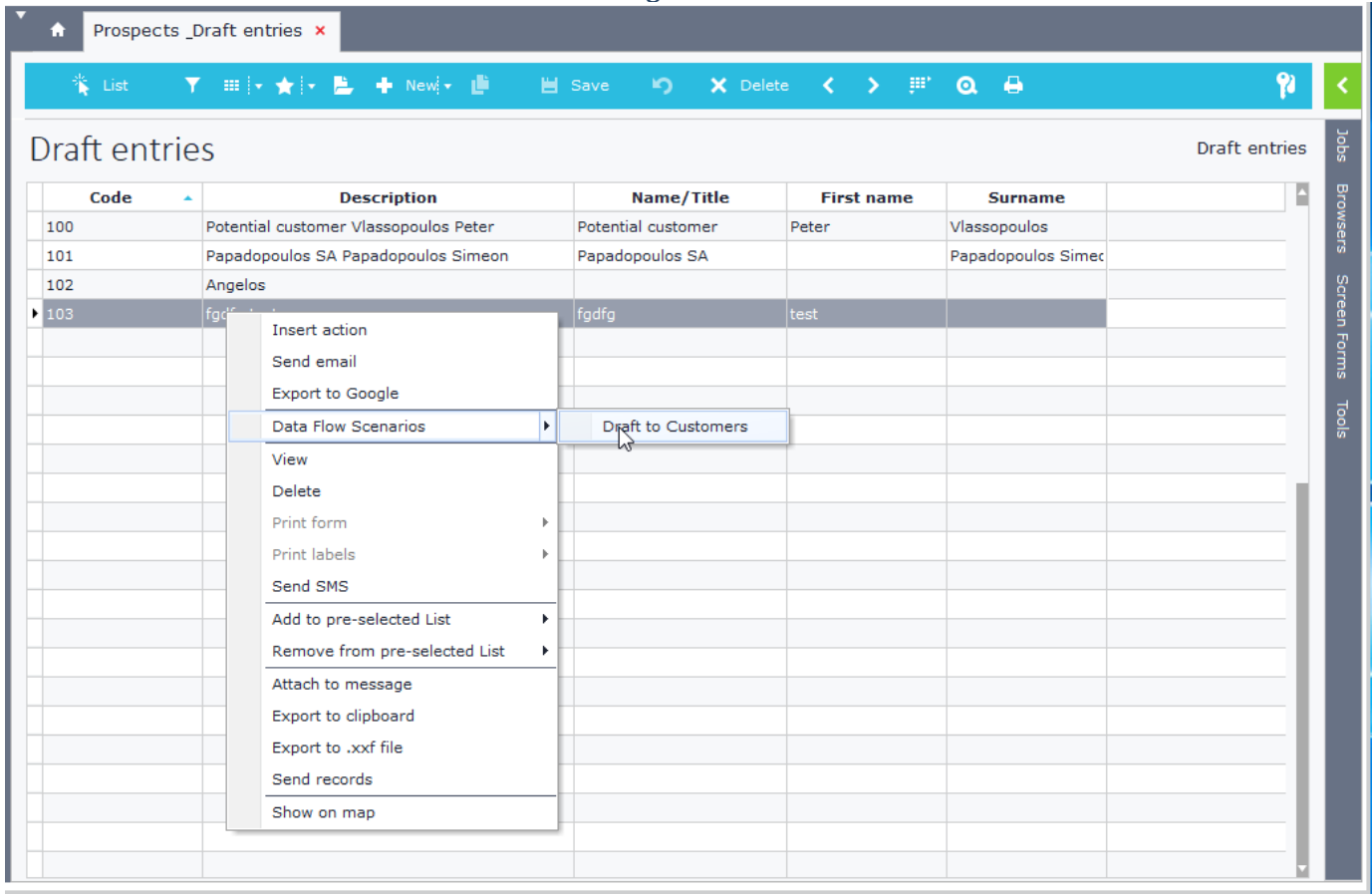


Figure A3

B. Data Flow Rules

Data flow rules is the tool where you define the jobs that will be executed when transferring data from one entity to another. Rules description is set through the "Description" textbox.

Data flow rules' design is saved in the blob field SODATA of the table CSTINFO (CSTTYPE=12) and can be easily transferred through different installations as ".cst" or ".auv" files using the "[Custom Administration](#)" tool.

B.1 Source entity

The source entity (object) is the module that contains the data to be transferred (e.g. SALDOC - sales documents). Inside the textbox "**Name**" you enter the name of the source object or you can select it by clicking on the link.

The "**View**" combo box lets you pick from the available views (forms) of the object. By selecting a specific form of the object, its tables, **including the custom ones** are displayed in the datagrid of the tab "Data tables". If no view is selected then the default one is used.

The datagrid of the tab "**Data**" lists all the tables you can use as "**Data source**" for your target tables. Fields from data source tables can be used to assign values to fields of the target tables.

- **Grouping** of source data can be done for any table of the source entity (**Figure B1.1 [1]**). Select the table from the datagrid of the tab "**Data tables**", click the button "**Grouping**" and then select a field of the source table.
- **Filters** can be applied to the records of any source table (**Figure B1.1 [2]**). Filters are added in the where clause of the SQL query that runs for the source entity. The accepted Values are:
 - **Numbers**: Adds the filter in the where clause using the SQL operator "=" (e.g. A.SODTYPE=51).
 - **Comma separated string**: Adds the filter in where clause using the operator "IN" (e.g. A.MTRMARK IN (1,2)).
 - **NULL, NOTNULL**: Adds the following in the where clause: **AND FIELD IS NULL, AND FIELD IS NOT NULL**.
 - **SQL statement**: Adds the SQL statement of the cell "Value" in the where clause of the main query. It requires manually entry of the keyword **SQLFILTER** inside the cell of the column "Field".

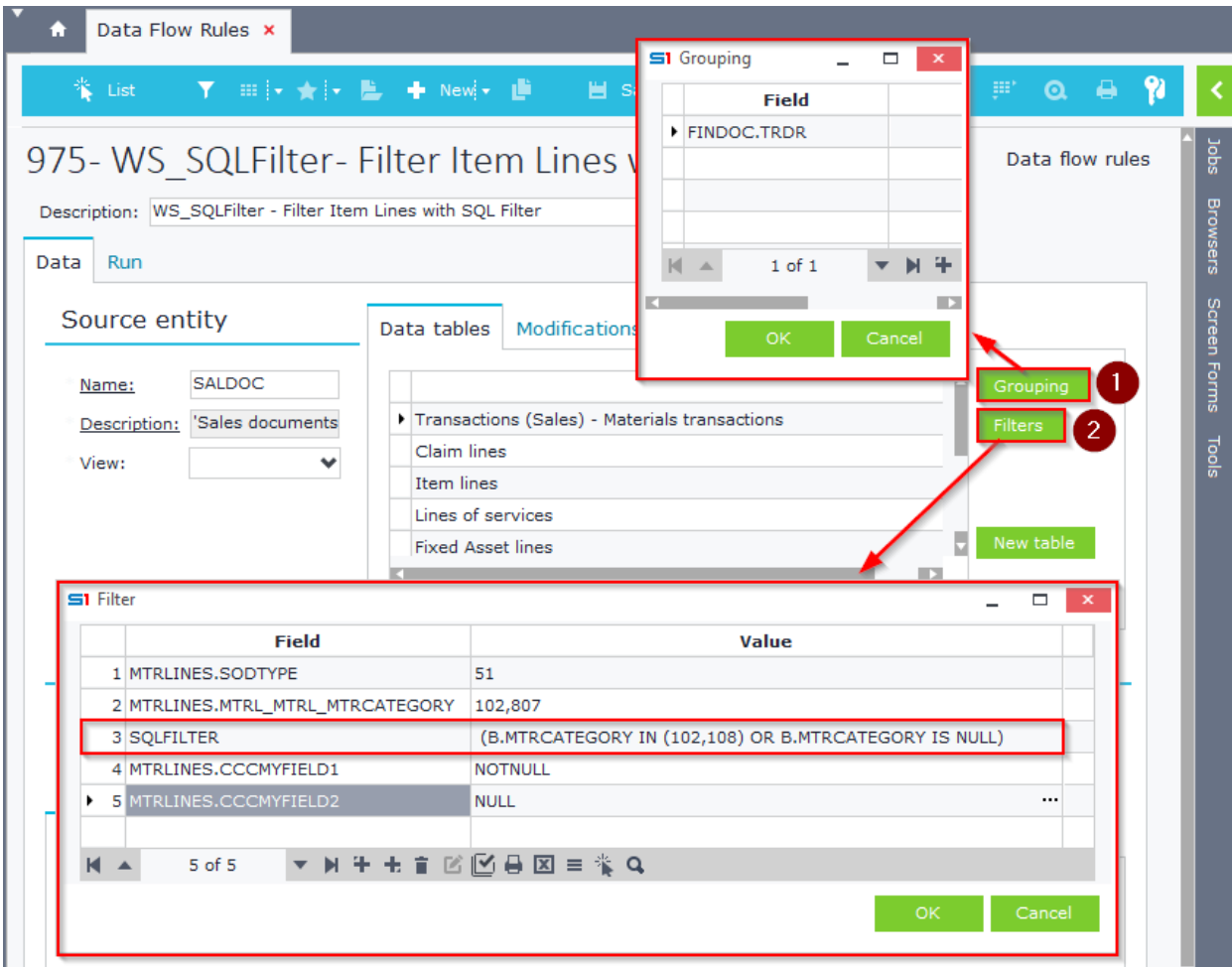


Figure B1.1

Inside the datagrid of the "**Modifications**" tab you can insert the fields of the source data tables that will be updated after the execution of the job (Figure B1.2).

If you enable the option "**Use Object**" (Figure B1.3) then the modification of the fields will be done through object (CreateObj, Locate, DBPost) using the form you have defined in the "**View**" combo box. This means all SoftOne business commands and validations will be executed. Apart from that, all the custom functions & commands you may have written in form script will also run.

When the option "Use Object" is not enabled then the updates are executed through SQL update statements directly to the database.

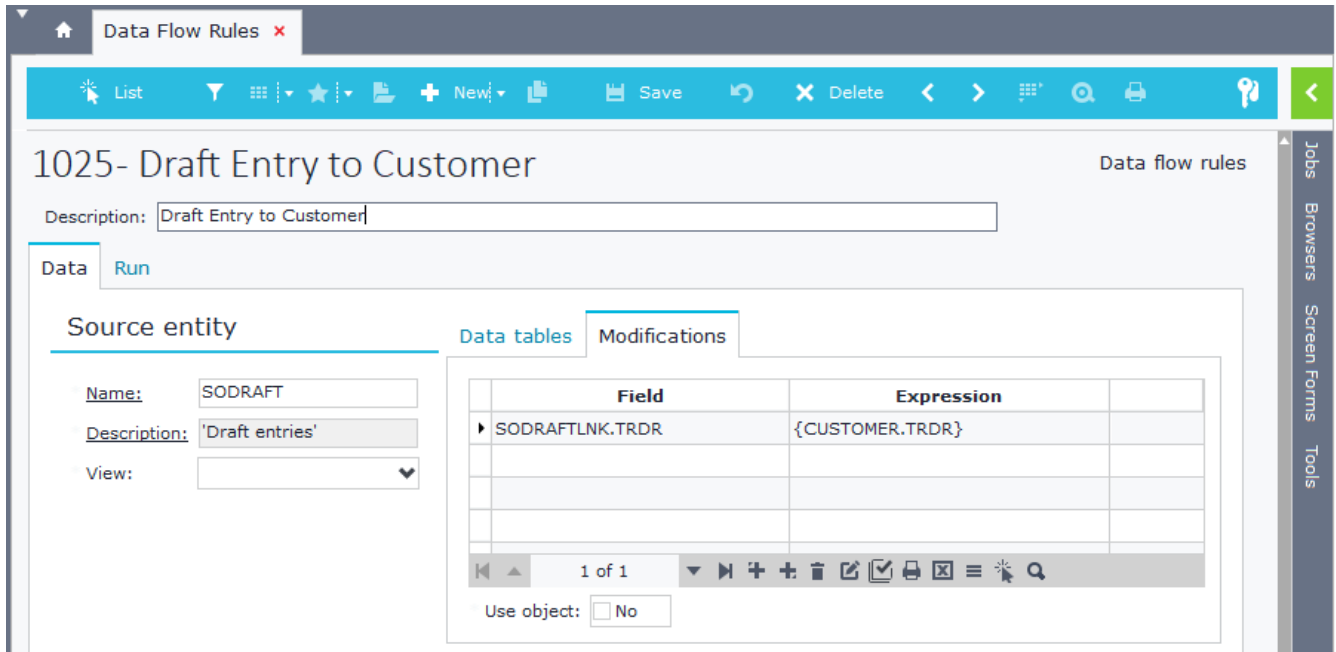


Figure B1.2

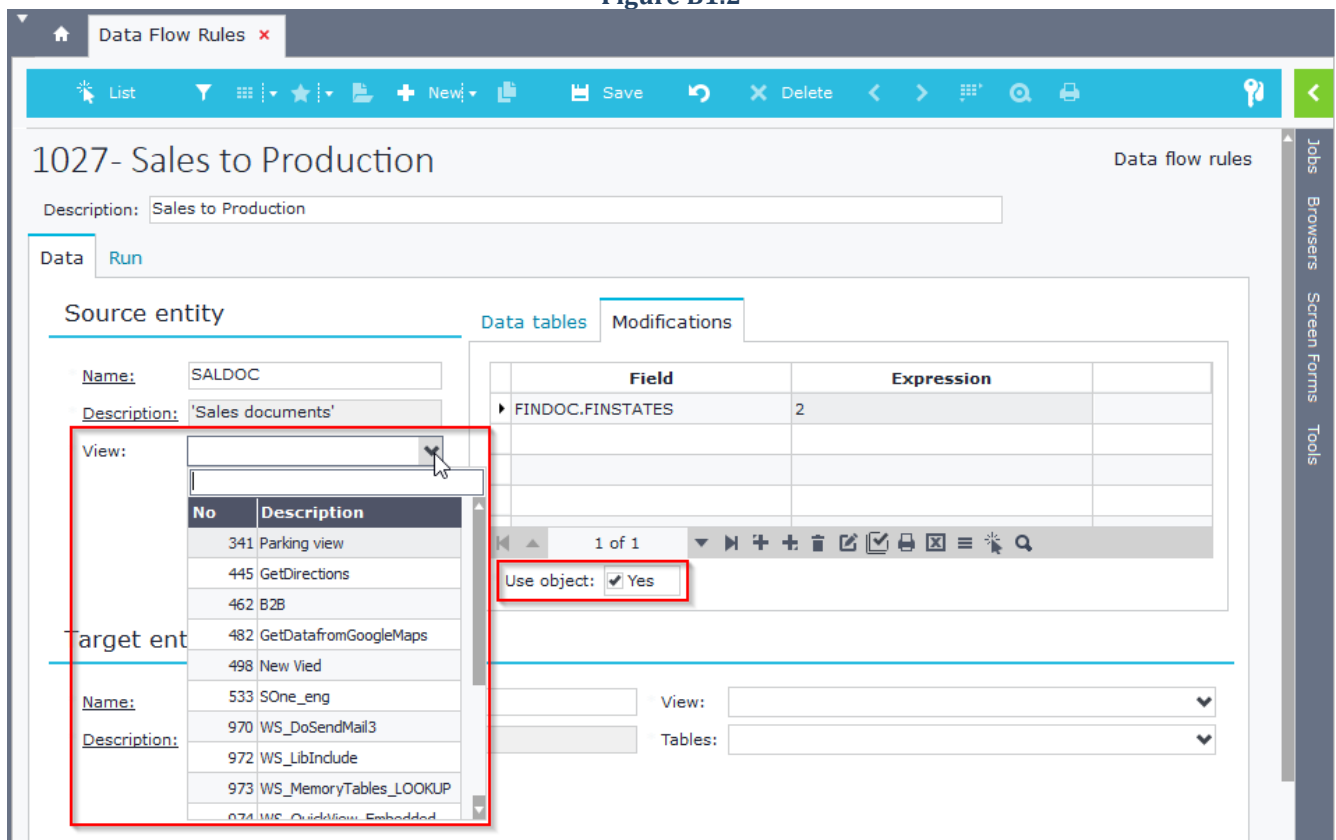


Figure B1.3

B.2 Target entity

The target entity contains the data of the target object, which is the one that will be created (e.g., PRDDOC - production documents). Inside the textbox "**Name**" type the name of the object or select it by clicking on the link.

If you select a specific form from the combo box "**View**" then the new record will be created using this form, meaning that the custom script and commands of this form will occur.

In "**Tables**" combo box select the tables that will be used for assigning values through the source tables. For every table selected, new tabs automatically open labeled with tables' name (**Figure B2.1**).

Target entity

Name: SALDOC View: 482 GetDatafromGoogleMaps

Description: 'Sales documents' Tables: 1,2,5

Transactions (Sales) Materials transactions Item lines

Data source: Transactions (Sales) - Materials transactions

Assign field values

Field	Value
Series	{FINDOC.SERIES}
Customer	{FINDOC.TRDR}

Advanced

	Password	Tables
<input checked="" type="checkbox"/>	1	Transactions (Sales)
<input checked="" type="checkbox"/>	2	Materials transactions
<input type="checkbox"/>	3	Entities to be Depreciated
<input type="checkbox"/>	4	Claim lines
<input checked="" type="checkbox"/>	5	Item lines
<input type="checkbox"/>	6	Lines of services
<input type="checkbox"/>	7	Fixed Asset lines
<input type="checkbox"/>	8	Serial number per material lines
<input type="checkbox"/>	9	Expense analysis
<input type="checkbox"/>	10	VAT analysis

☐ Select All

Figure B2.1

Each table of the target entity is displayed in a different tab. The "**Data source**" combo box contains all the available tables of the Source Entity (**Figure B2.2**). Pick the Data source table keeping in mind that its fields will be available when you assign values to the target table fields.

Target entity

Name: PURDOC View:

Description: 'Purchase documents' Tables: 1,7

Transactions (Purchases) Item lines

Data source: Items - Extra Data: Items - Item Photo

Assign field

Series	Calculated item cost/sales prices		
Supplier	Document pointers		
Date	Items - Extra Data: Items - Item Photo_1		

Figure B2.2

Inside the datagrid "**Assign field values**" you set the field values of the selected target table (**Figure B2.3**).

- Column "**Field**"(1): Target fields. Select the fields of the target entity that will be updated during the process.
- Column "**Expression**"(2): Enter the values that will be assigned to the target fields. You can use any internal function (e.g., IF, RUNSQL) or you can select any field from the source entity.

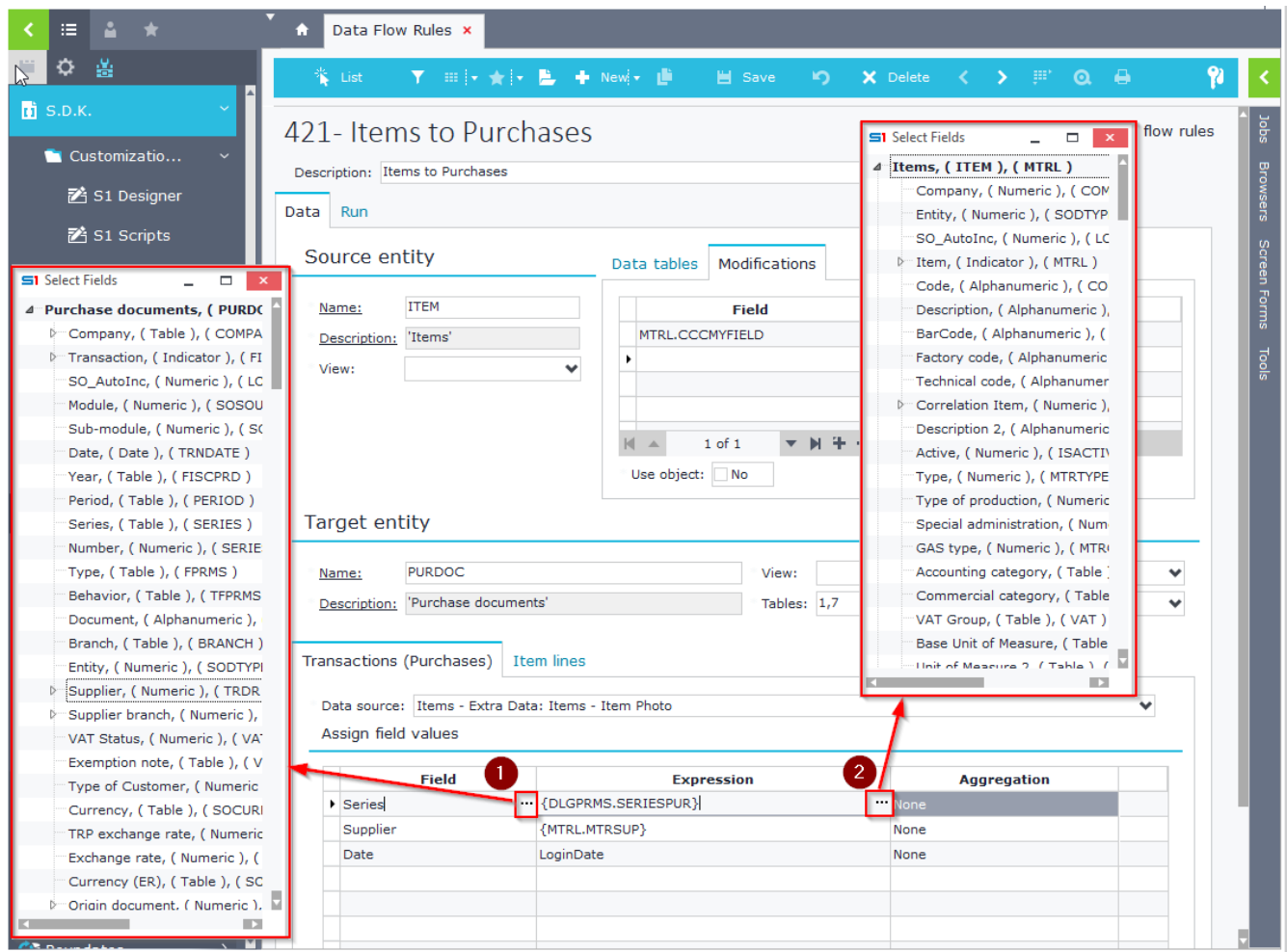


Figure B2.3

Double click on any line of the target entity datagrid zooms in the expressions (**Figure B2.4**).

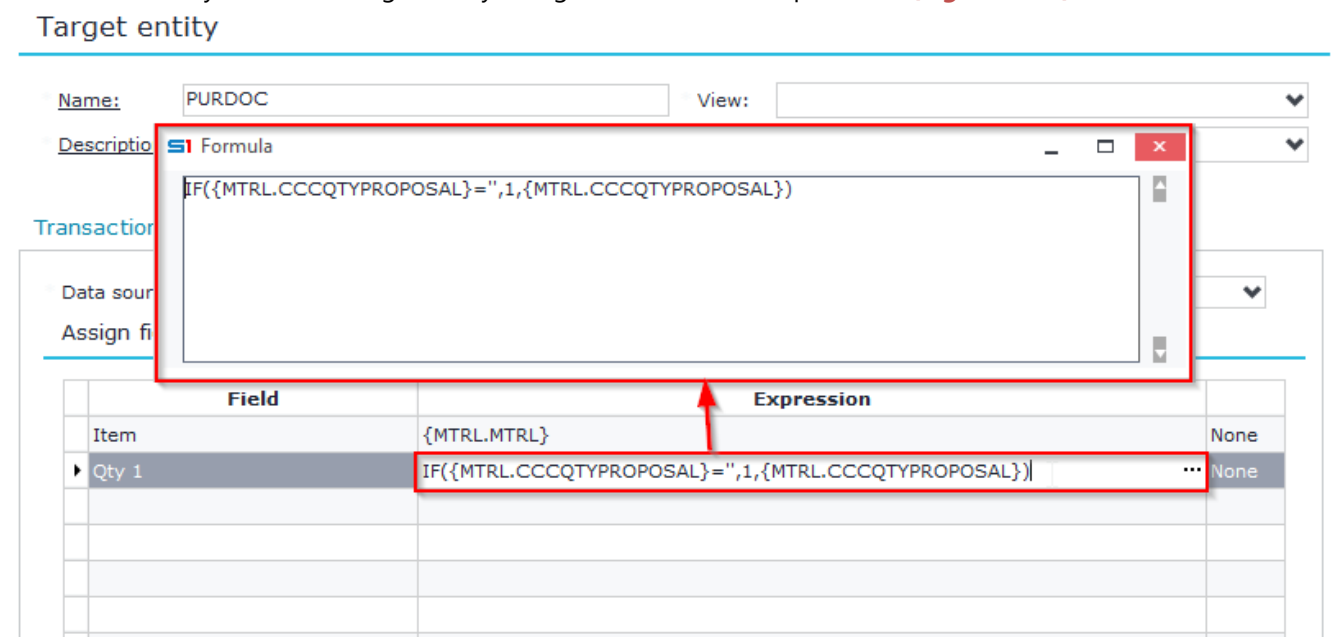


Figure B2.4

- Column "**Aggregation**": allows you to specify math operations (Sum, Greater than, etc.) in the fields of the datagrid. This can take place only if you have defined a source table that is grouped by certain values.
For example, if you want to create a Sales document from another Sales document and apply packing of items (in item lines), then you have to group the items by MTRL (in source table Item lines) and set their target quantity as the sum of the quantities per grouped item and their price as the max item price (**Figure B2.5**).

894- Convert Sales to Sales Data flow rules

Description:

Data Run

Source entity

Name:

Description:

View:

Data tables Modifications

Transactions (Sales) - Materials transactions
Claim lines
Item lines
Lines of service
Fixed Asset line

Target entity

Name:

Description:

Transactions (Sales) Item lines

Data source:

Assign field values

Field	Expression	Aggregation
Item	{MTRLINES.MTRL}	None
Qty 1	{ITELINES.QTY1}	Sum
Price	{MTRLINES.PRICE}	Greater than

Grouping

Field

MTRLINES.MTRL

OK Cancel

Figure B2.5

B.3 Runtime Settings

In the tab **"Run"** you set whether during running the job, the records created will be displayed on the screen and you can also define the dialog parameters, as well as the connection properties.

Enable the option **"Display screen"** when you wish to display the new records created on the screen and wait for the user to save them. If the option is not enabled then when the job is completed, the new records created will be displayed in log text under the link "Results" of the dialog.

The option **"Display dialog"** displays the dialog filters before running the job.

The dialog parameters are set through the datagrid **"Dialog parameters"** (Figure B3.1).

The records of the datagrid indicate the filter fields and the columns define their properties. These fields can be used either in the source entity for filtering the source tables or as values of the target table fields (Figure B9).

Enabling the option **"Exclude login company"** defines that the new records will not be created in the login company.

Inside the datagrid **"Other connections to apply"** you can set the companies that the new records will be created, as well as the branch and the credentials of the login user. If you do not apply the user credentials, then the entries will be created using the login user.

Inside the column **"Connection file XCO"** you can define a connection to a different database.

Data Flow Rules x

List Filter New Save Delete

421- Items to Purchases

Description: Items to Purchases

Data **Run**

Interface

Display screen: ☒ Yes

Display dialogue: ☒ Yes

Dialog parameters

Name	Descript	Data type	Displa...	Req...	Visible	Re...	Editor	Default...	For
▶ SERIESPUR	Series	Smallint	15	No	Yes	No	SERIES(F[SOSOURCE=1251])	2021	

1 of 1

Advanced Connection Properties

Exclude login company: ☐ No

Other connections to apply

Company	Branch	User	Password	Connection file(XCO)

Figure B3.1

C. Data Flow Scenarios

Inside data flow scenarios you can set the jobs (data flow rules or SBSL scripts) to be displayed and run when right clicking on the browsers of the specified objects. Data flow scenarios can be exported from the Custom Administration tool, where you can also set user rights.

Data flow scenarios' design is saved in the blob field SODATA of the table CSTINFO (CSTTYPE=13) and can be easily transferred through different installations as ".cst" or ".auv" files using the "[Custom Administration](#)" tool.

Figure C1 shows an example of a data flow scenario created to run through the "Items" browser (Entity ITEM). At runtime, if the dataflow rule is set to display a dialog, then the dialog will be displayed as in Figure C2.

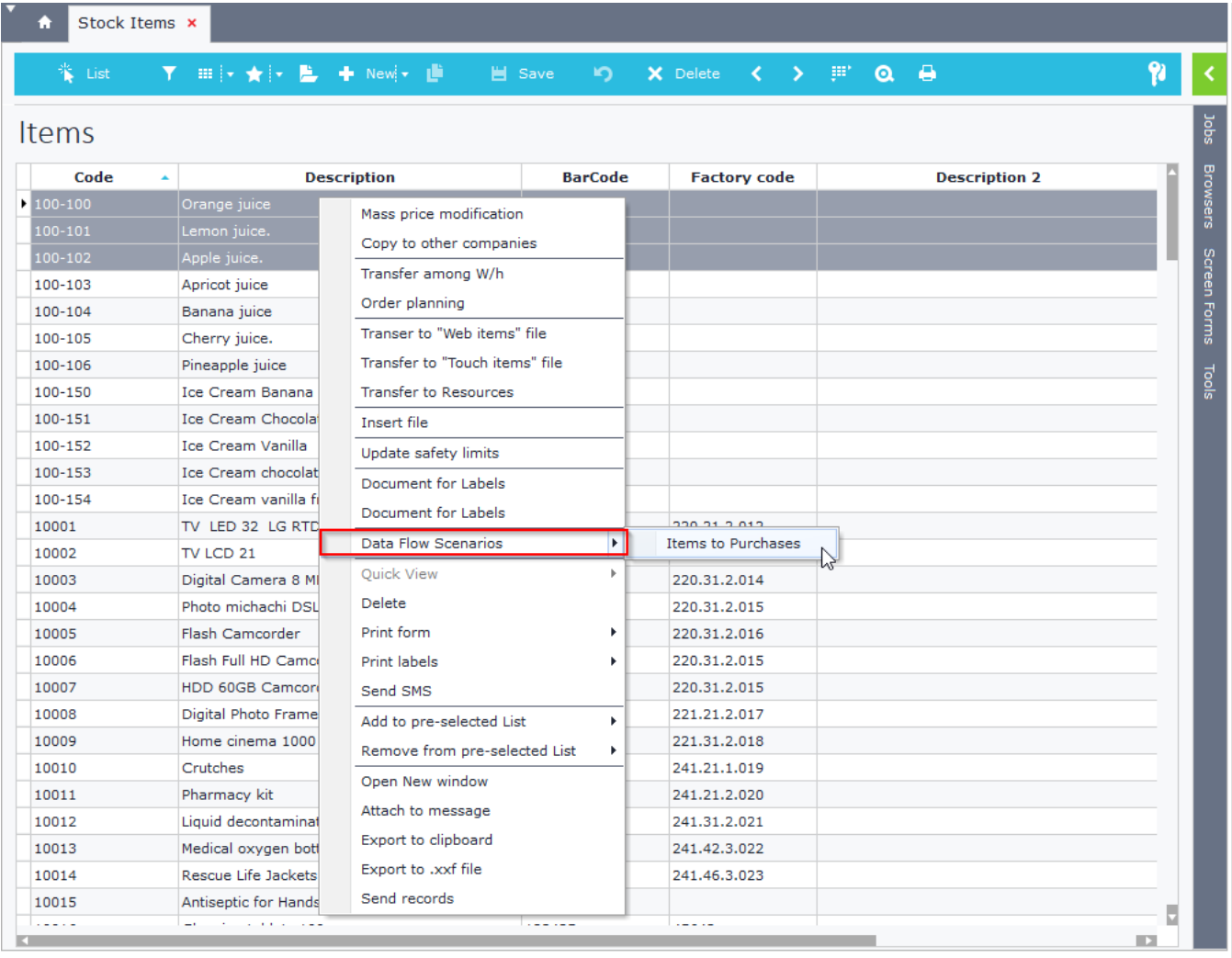


Figure C1

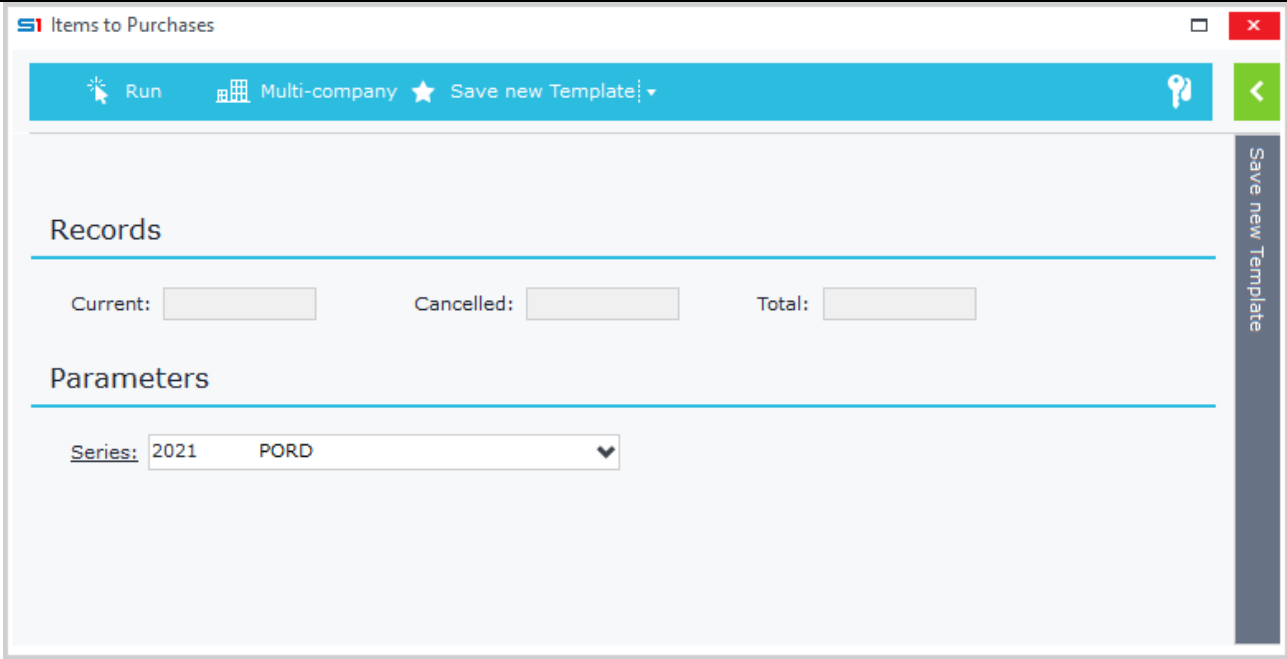


Figure C2

A data flow scenario is created through the menu Job "Tools – S.D.K. – Customization Tools – Data Flow Scenarios". Create a new one, enter the "Description", that will also be displayed in the right click pop up menu, and the "Entity" name of the object that this scenario will be applied.

Inside the datagrid set the data flow rules or SBSL scripts that will be executed when selecting the specific scenario from the pop-up menu of the browser (Figure C3).

A data flow scenario can run one or more data flow rules and SBSL scripts (Figure C4).

Note: Data flow rules are linked to data flow scenarios through their description. This means that if you change the description of a data flow rule then you have to link it again inside the dataflow scenario.

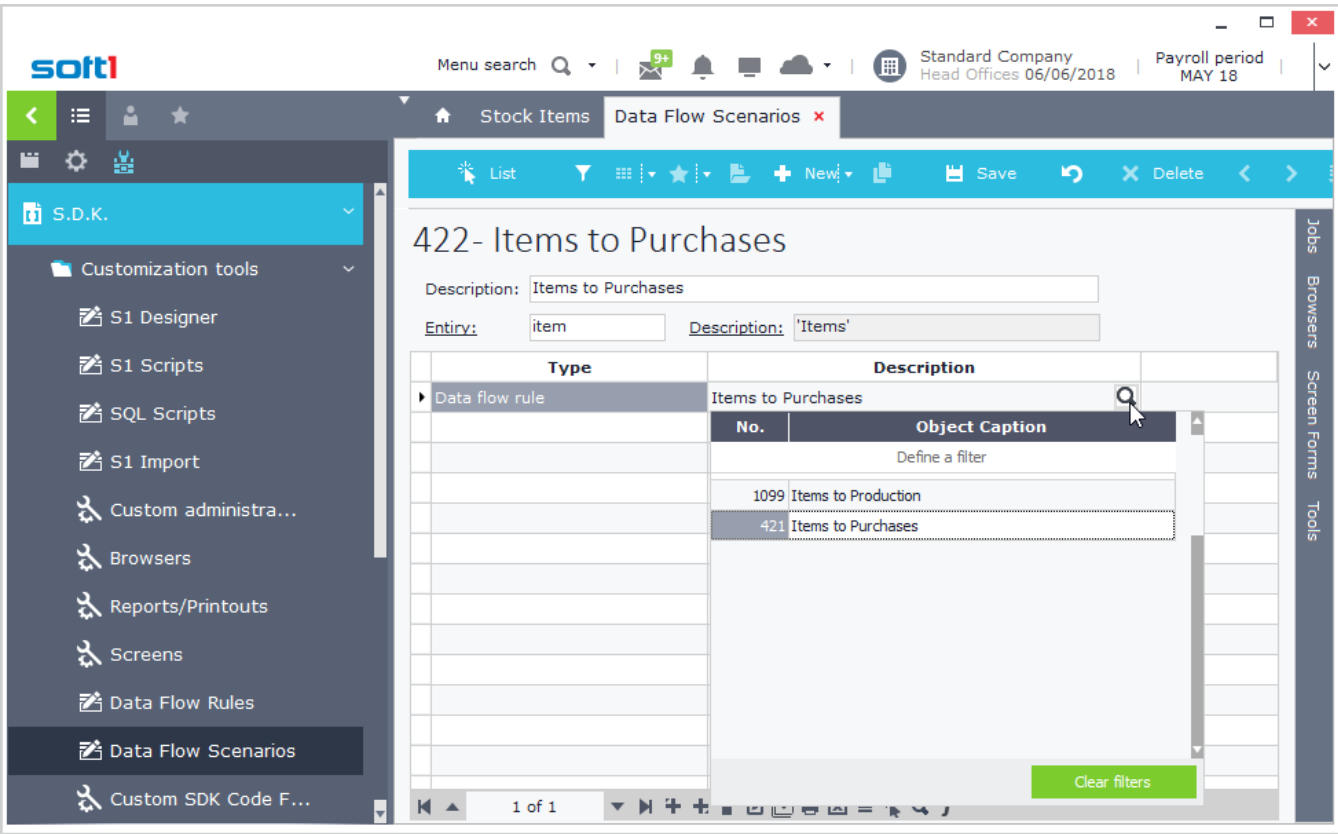


Figure C3

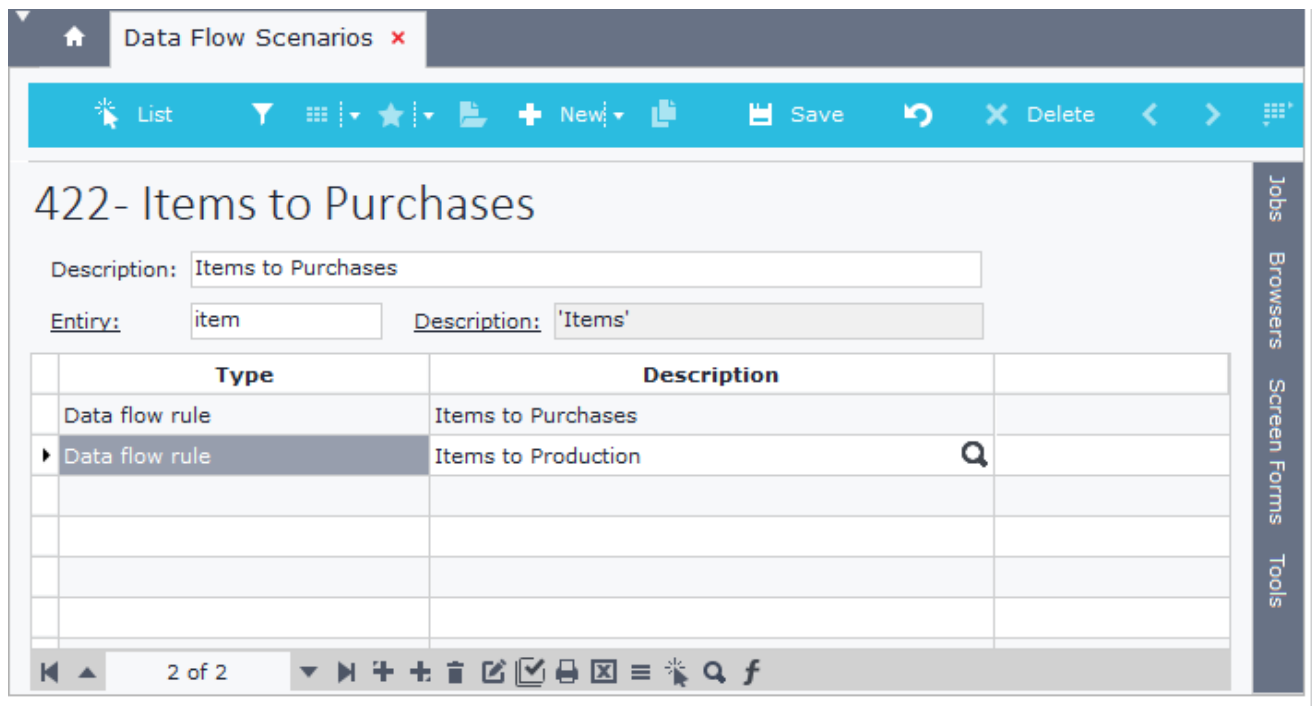


Figure C4

D. Execute Data Flows from Screen Scripts

Data flows can be called through form scripts of the screens, through function X.EVAL. You can call either the flow rules or scenarios using the following commands:

SOTRANSFORM.RULEBYID (CSTINFO: integer, IDs: string)

CSTINFO: Data flow rule ID

IDs: IDs of the entries, where the rule will be applied, separated by commas (,) or question marks (?)

SOTRANSFORM.RULEBYNAME (CSTNAME: string, IDs: string)

CSTNAME: Data flow rule Name

IDs: IDs of the entries where the rule will be applied, separated by commas (,) or question marks (?)

SOTRANSFORM.MODELBYID (CSTINFO: integer, IDs: string)

CSTINFO: Data flow scenario ID

IDs: IDs of the entries, where the scenario will be applied, separated by commas (,) or question marks (?)

SOTRANSFORM.MODELBYNAME (CSTNAME: string, IDs: string)

CSTNAME: Data flow scenario Name

IDs: IDs of the entries, where the scenario will be applied, separated by commas (,) or question marks (?)

Example

Run the data flow rule with code 1026, that converts a draft entry to a customer through the screen script of the draft entries object. The rule is executed after clicking a button (cmd = 20180607) (Figure D1) and displays the dialog window that is set within the rule (Figure D2).

```
function EXECCOMMAND(cmd) {
    if (cmd == 20180607)
        Transfertocustomer();
}
function Transfertocustomer() {
    if (SODRAFT.SODRAFT > 0)
        A = X.EVAL('SOTRANSFORM.RULEBYID(1026, ' + SODRAFT.SODRAFT + ')');
}
```

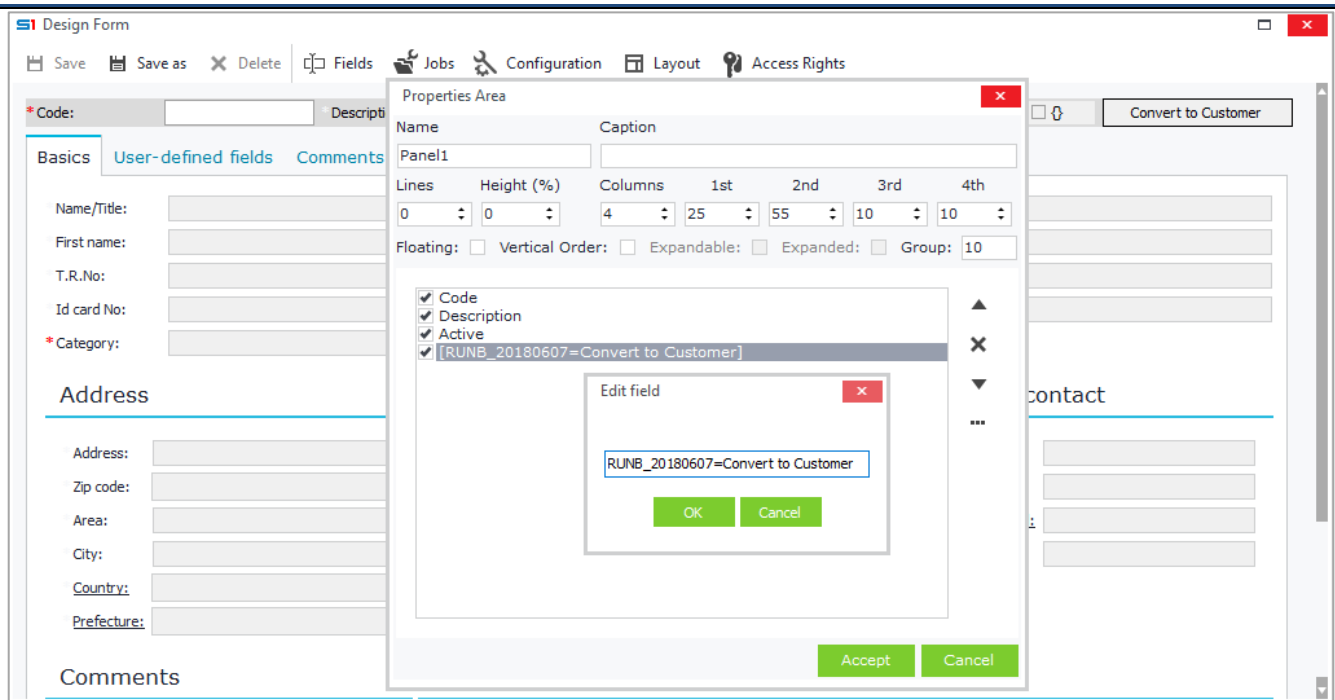


Figure C4

E. Case Studies

1. Sales to Purchases Conversion

Batch job created with data flow rule that transfers **sales documents to purchase documents** with series and supplier selection in dialog filters.

Create a new data flow rule and select the object SALDOC as source entity. The “Data tables” tab displays the available data tables of this object (Figure E.1.1).

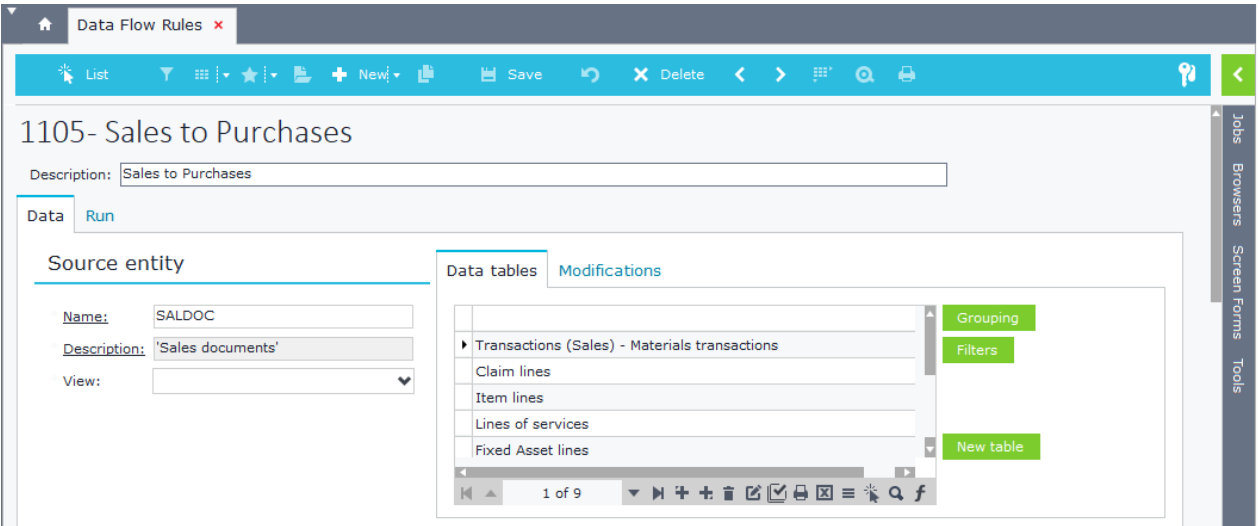


Figure E.1.1

The dialog filters are created through the “Run” Tab. For each field, enter the description, the data type, Display size, Editor and any other property required (Figure E.1.2).

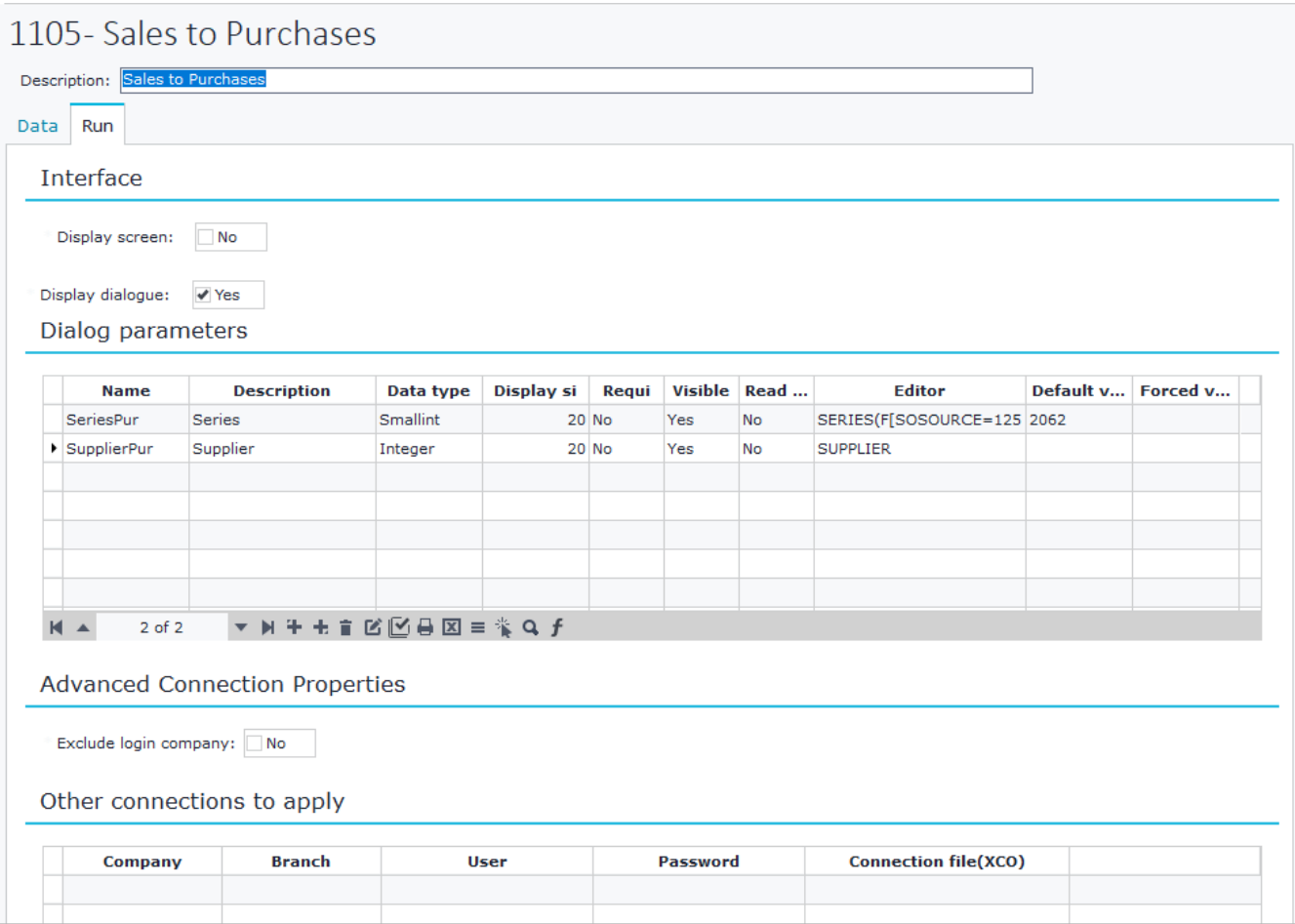


Figure E.1.2

The data will be transferred to “purchase documents”, thus the object of the target entity must be PURDOC. Select the target tables (e.g. Transactions and Item lines) and then for each table (Tab) select the source table from which fields values will be assigned. Inside the datagrid select the fields and add their values in the “Expression” column (pick from source table fields or filter parameters) (Figure E.1.3).

Target entity

Name: PURDOC

View:

Description: 'Purchase documents'

Tables: 1,7

Transactions (Purchases)

Item lines

Data source: Transactions (Sales) - Materials transactions

Assign field values

Field	Expression	Aggregation
Series	{DLGPRMS.SeriesPur}	None
Supplier	{DLGPRMS.SupplierPur}	None
Document	'Pur'+{FINDOC.FINCODE}	None

Figure E1.3

Conversion (Originated from) is done through the fields FINDOCS and MTRLINESS of Item Lines (Figure E1.4).

Target entity

Name: PURDOC

View:

Description: 'Purchase documents'

Tables: 1,7

Transactions (Purchases)

Item lines

Data source: Item lines

Assign field values

Field	Expression	Aggregation
Item	{MTRLINES.MTRL}	
Qty 1	{MTRLINES.QTY1}	
Origin	{MTRLINES.FINDOC}	
Pointer to transaction lines	{MTRLINES.MTRLINES}	

Select Fields

Account, (Alphanumeric), (ACNMSK)

Account 2, (Alphanumeric), (ACNMSK1)

Origin, (Numeric), (FINDOCS)

Pointer to transaction lines, (Numeric), (MTRLINESS)

Transaction Link, (Numeric), (FINDOCL)

Pointer to transaction lines Link, (Numeric), (MTRLIN

Analysis by Variation, (Text), (SOANAL)

Import from gross value, (Numeric), (SXMSKK)

Gross value, (Numeric), (SXPERC)

S.C.T., (Numeric), (BGLEXCISE)

Create semi-finished product document, (Numeric), (A

Revenue account, (Alphanumeric), (ACNMSKS)

Figure E1.4

The last step is to create a data flow scenario using this data flow rule in order to display the job upon right-clicking in the “sales documents” browser (Figure E1.5).

Data Flow Scenarios

List

Filter

Grid

Star

New

Save

Delete

Back

Forward

Search

Print

1106- Sales to Purchases

Description: Sales to Purchases

Entry: SALDOC

Description: 'Sales documents'

Type	Description
Data flow rule	Sales to Purchases

Jobs

Browsers

Screen

Figure E1.5

2. Sales to Sales (Company)

Batch job created by data flow rule for transferring **sales documents from the login company to sales documents of another company** and branch. It also applies customers grouping.

The source entity is created as in the previous example using SALDOC. Grouping with customers table is executed by selecting the record (table) "Sales Transactions" in the datagrid of the "Data tables" tab, clicking on the button "Grouping" and selecting the field "Customer" (FINDOC.TRDR) of the sales documents (Figure E2.1).

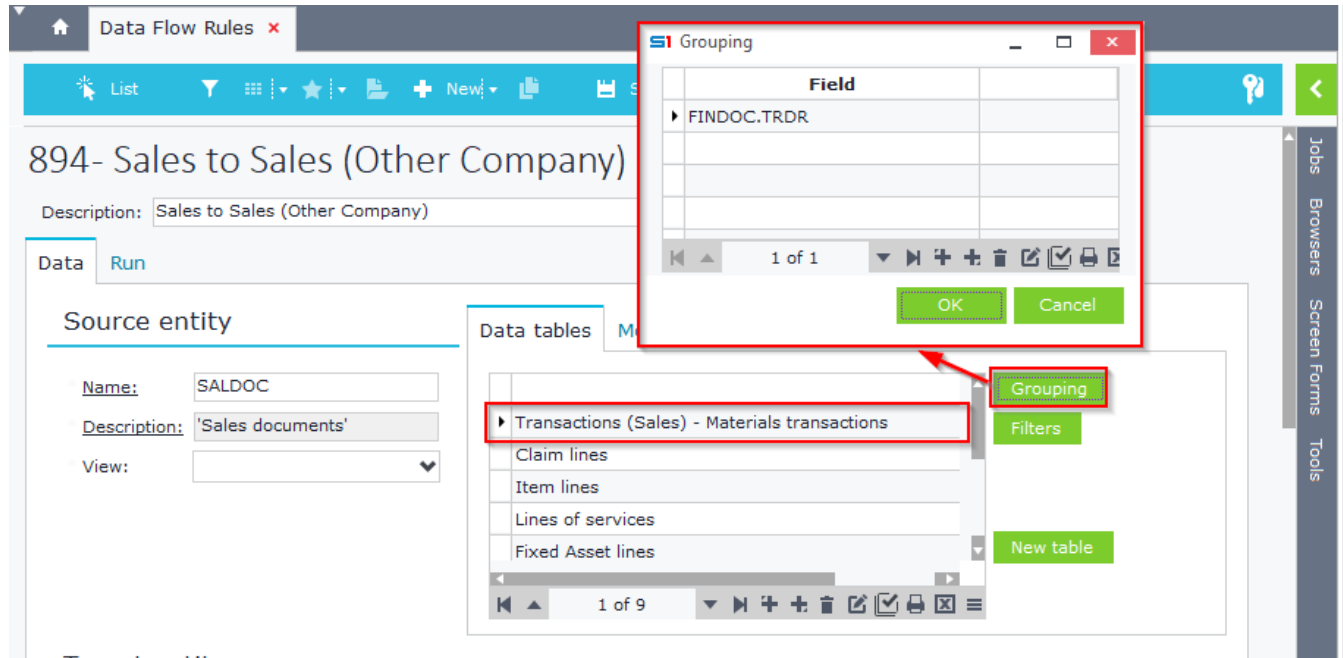


Figure E2.1

Customers (TRDR) and items (MTRL) have different ids in each company, so the matching should be done by their "Code" field. This can be done by using the function "ID" that returns the id of a table for a given code (Figures E2.2 and E2.3). Alternatively you can use a SQL statement using the internal function RUNSQL. Of course, you have to make sure that in order for the data flow to work correctly you must have the same customer codes and item codes in both companies.

Target entity

Name:	SALDOC	View:	
Description:	'Sales documents'	Tables:	1,5

Transactions (Sales)

Item lines

Data source: Transactions (Sales) - Materials transactions

Assign field values

Field	Expression	
Series	{FINDOC.SERIES}	None
Customer	ID('CUSTOMER',{FINDOC.TRDR_TRDR_CODE})	None

Figure E2.2

Transactions (Sales) Item lines

Data source: Item lines

Assign field values

Field	Expression	Aggregation
Item	ID('ITEM',{MTRLINES.MTRL_MTRL_CODE})	None
Qty 1	{ITELINES.QTY1}	Sum
Price	{MTRLINES.PRICE}	Greater than

Figure E2.3

In the Run Tab, in the "Advanced connection properties" select to exclude the login company and then in the datagrid below add a new record with the target company and branch (Figure E2.4).

Advanced Connection Properties

Exclude login company: ☒ Yes

Other connections to apply

Company	Branch	User	Password	Connection file(XCO)
1002	1002			

Figure E2.4

After running the data flow scenario by right clicking the browser of sales documents, the results that indicate whether transfer was executed correctly will be displayed as in Figure E2.5.

Sales to Sales (Other Company)

Run Multi-company Save new Template

Records

Current: 2 Cancelled: Total: 2 Results

Save new Template

Figure E2.5

3. Items to Purchases

Batch job created by data flow rule that will run through the inventory **items** browser and then **create purchase documents** with grouping based on the ordinary supplier of the items. For the items that don't have a main supplier, the supplier indicated in the dialog filters will be inserted. If no purchase quantity is entered in items, line items shall have quantity 1.

In the source entity select the object ITEM. Grouping items per ordinary supplier is executed using the field MTRL.MTRSUP (Figure E3.1).

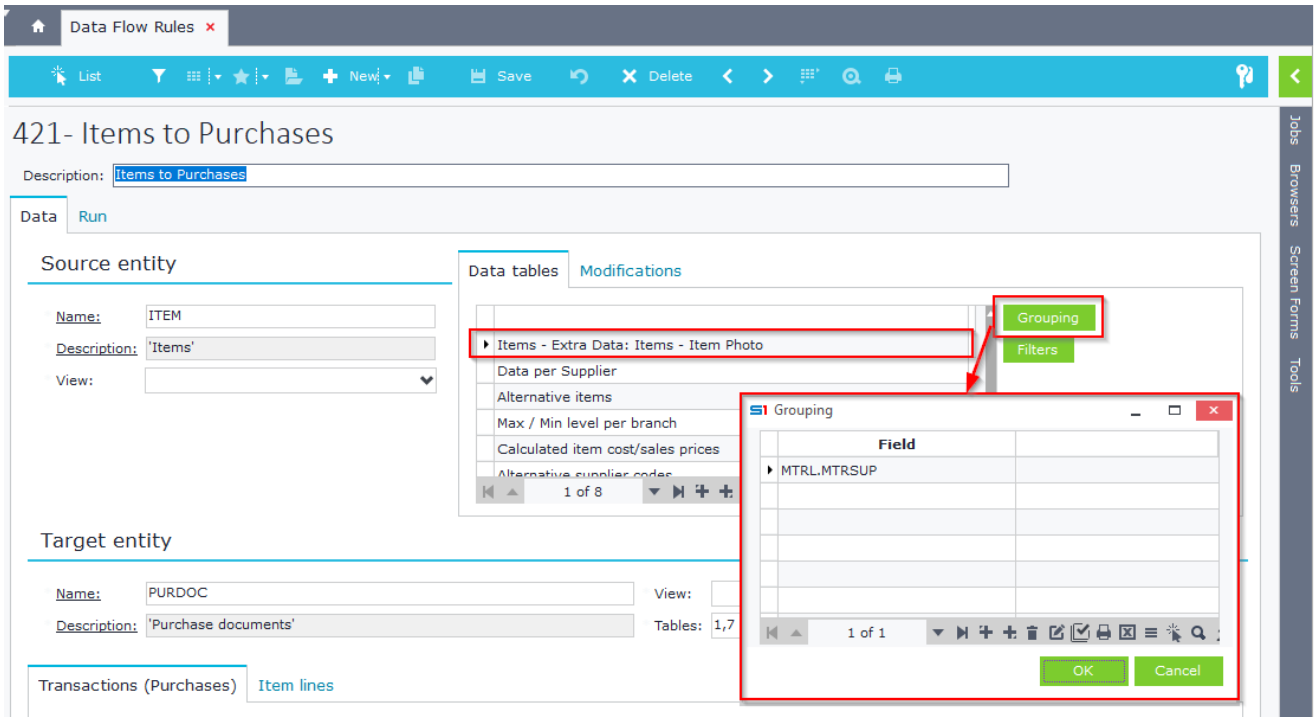


Figure E3.1

The dialog filters display the purchase series and the supplier that will be used only for items that do not have an main supplier (Figure E3.2).

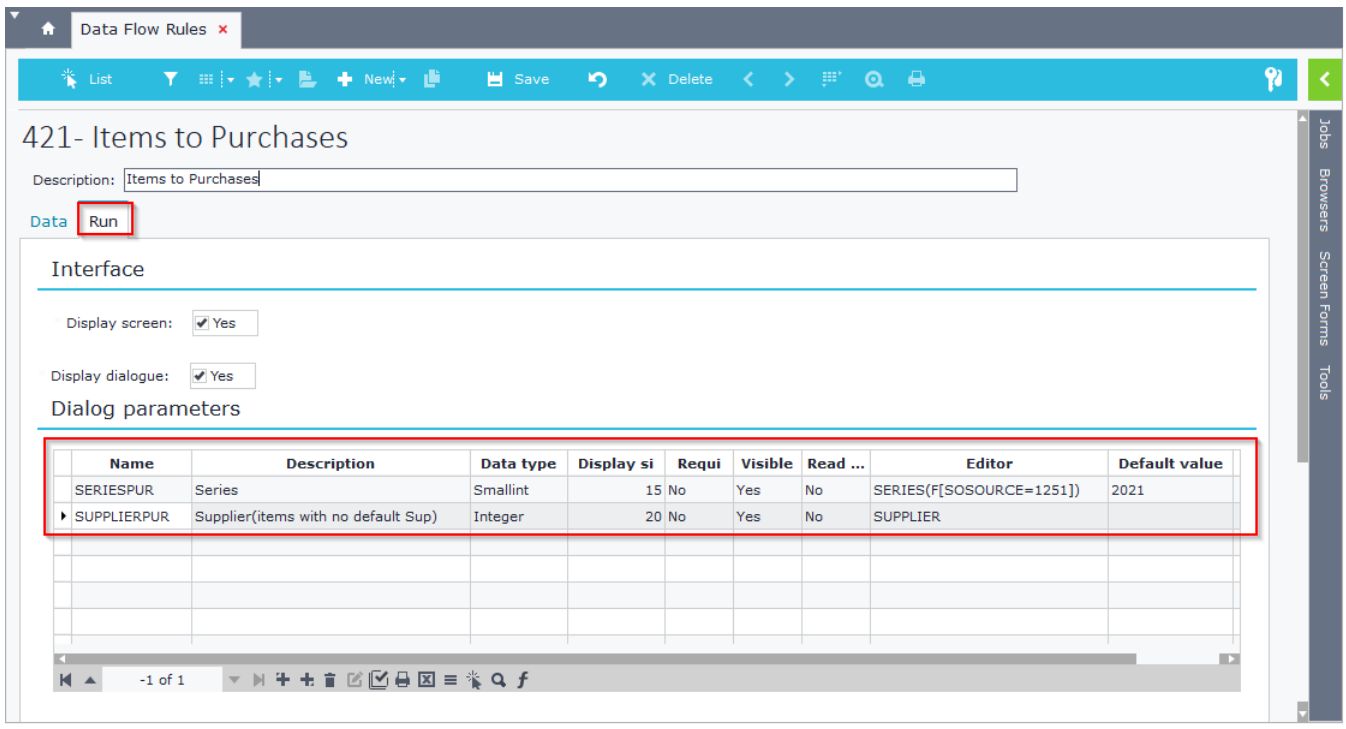


Figure E3.2

In the target entity enter the object PURDOC (purchase documents) and select "Transactions" and "Item lines" as target tables (Tables 1,7).

The header of the new document (Transactions (Purchases)) will have as a datasource the *Items* table, because of the grouping per items ordinary supplier. The assignment of field values is executed as in Figure E3.3.

Supplier is filled using the expression: `IF ({MTRL.MTRSUP}=' ', {DLGPRMS.SUPPLIERPUR}, {MTRL.MTRSUP})`

Target entity

Name: PURDOC View:
 Description: 'Purchase documents' Tables: 1,7

Transactions (Purchases) Item lines

Data source: Items - Extra Data: Items - Item Photo

Assign field values

Field	Expression	Aggregation
Series	{DLGPRMS.SERIESPUR}	None
Supplier	IF({MTRL.MTRSUP}=' ', {DLGPRMS.SUPPLIERPUR}, {MTRL.MTRSUP})	None
Date	LoginDate	None

Figure E3.3

Item lines cannot be updated from the items table of the source entity because this particular table has been grouped per item main supplier, thus it contains only the distinct supplier records. So, you need to add again the item table by creating a new table in the source entity, which will contain all the item records (Figure E3.4).

Then you can use it as the data source table for the item lines (Figure E3.5).

Data Run

Source entity

Name: ITEM View:
 Description: 'Items'
 View:

Grouping

Field

MTRL.MTRL

1 of 1

OK Cancel

tables Modifications

Calculated item cost/sales prices

Alternative supplier codes

Document pointers

Items - Extra Data: Items - Item Photo 1

Grouping

Filters

New table

8 of 8

Figure E3.4

Target entity

Name: PURDOC View:
 Description: 'Purchase documents' Tables: 1,7

Transactions (Purchases) Item lines

Data source: Items - Extra Data: Items - Item Photo_1

Assign field values

Field	Expression	Aggregation
Item	{MTRL.MTRL}	None
Qty 1	IF({MTRL.PURQTY}=' ', 1, {MTRL.PURQTY})	None

Figure E3.5

Chapter 10 – Data Flows

In the example of Figure E3.6, seven items with four different suppliers are selected. After running the job, four new documents will be created (three of them are based on item suppliers and one is based on the supplier provided from the dialog) (Figure E3.7).

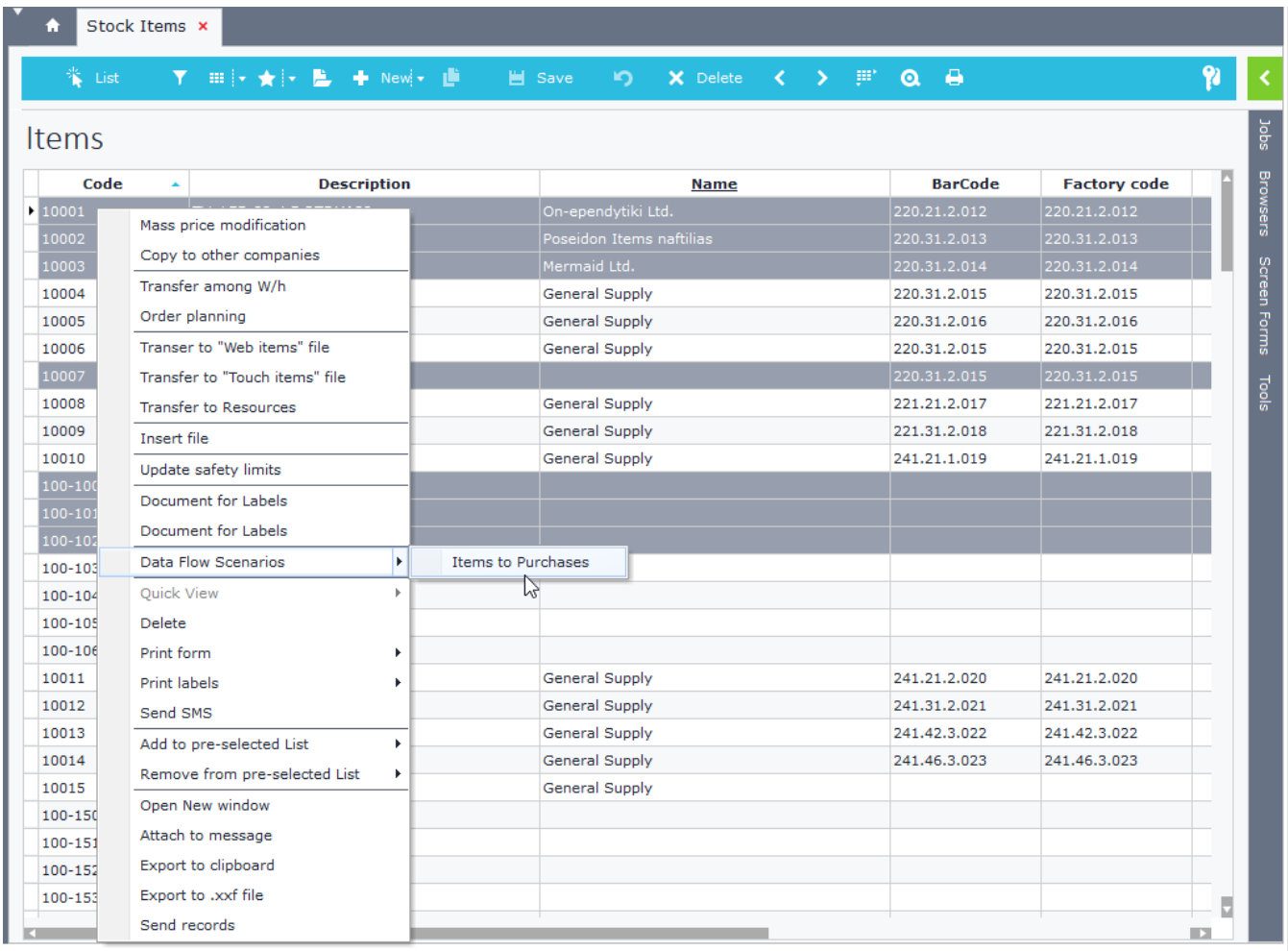


Figure E3.6

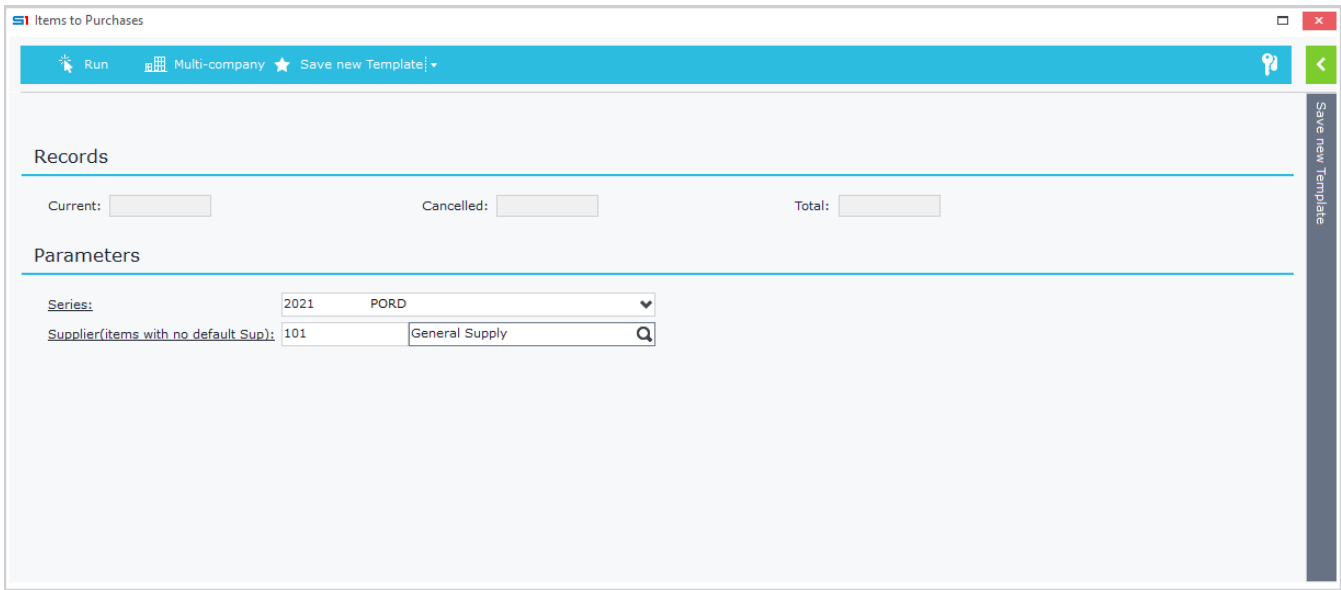


Figure E3.7

4. Production to Purchases

Batch job created by data flow rule that **creates purchase documents from production documents**. The items will be inserted based on the raw materials lines. The quantity will be entered through calculation of the lines' quantity and waste.

In the source entity select the object PRDDOC (production documents) and in the target entity the object PURDOC (Figure E4.1).

1108- Production to Purchases

Description: Production to Purchases

Data **Run**

Source entity

Name: PRDDOC
Description: 'Production documents'
View: [v]

Data tables **Modifications**

Production documents - Materials transactions
Line of product
Lines of co-products
Lines of by-products
Lines of Raw/Auxiliary materials

Grouping
Filters
New table

Target entity

Name: PURDOC
Description: 'Purchase documents'
View: [v]
Tables: [v]

Figure E4.1

The dialog filters will have the purchases series and the supplier (Figure E4.2)

Data **Run**

Interface

Display screen: ☒ Yes
Display dialogue: ☒ Yes

Dialog parameters

Name	Description	Data type	Display si	Requi	Visible	Read ...	Editor	Default v...	F
SERIESPUR	Purchase Series	Smallint	25	No	Yes	No	SERIES(F[SOSOURCE=1251])	2061	
SUPPLIERPUR	Supplier	Integer	25	No	Yes	No	SUPPLIER		

Figure E4.2

The assignment of the values of the final entity will be executed for tables "Transactions" and "Item lines". In the transactions table enter the fields you want to be updated based on the source table of the production documents (Figure E4.3).

Target entity

Name:

PURDOC

View:

Description:

Purchase documents'

Tables:

1

Transactions (Purchases)

Data source:

Production documents - Materials transactions

Assign field values

Field	Expression	Aggregation
Series	{DLGPRMS.SERIESPUR}	None
Supplier	{DLGPRMS.SUPPLIERPUR}	None

Figure E4.3

In item lines table enter the fields from the corresponding source table "Lines of Raw/Aux materials" (Figure E4.4).

Transactions (Purchases)

Item lines

Data source:

Lines of Raw/Auxiliary materials

▼

Assign field values

	Field	Expression	Aggregation
▶	Item	{MTRLINES.MTRL}	None
	Qty 1	{MTRLINES.QTY1}-{MTRLINES.QTY1}*{MTRLINES.WASTE}/100	None

Figure E4.4

Create a new data flow scenario using the data flow rule above and run it from the production documents browser. The results will appear as in Figure E4.5.

Data Flow Rules **Production Documents x**

List Filter Grid Star New Save Delete Navigation icons

Documents

Date ▾	Document	Branch	Series	Materials	Costing data	Production
11/12/2018	PROD000004	Head Offices	PROD	13.221,00	0,00	13.221,00
▶ 10/12/2018	PROD000002	Head Offices	PROD	3.900,18	2.657,00	6.557,18

Production to Purchases

Run Multi-company ★ Save new Template ▾

Records

Current: Cancelled: Total: [Results](#)

Parameters

Purchase Series: PINV ▾
Supplier: General Supply 🔍

Save new Template

Figure E4.5

5. Marketing Campaigns to Tasks

Batch job created from data flow rule that **creates Tasks from Marketing Campaigns** with a specific form that has default values for Tasks fields.

In the source entity select the object PRJC (sales opportunities) and filter PRJC as in figure E5.1 (PRJC.PRJCRM = 1).

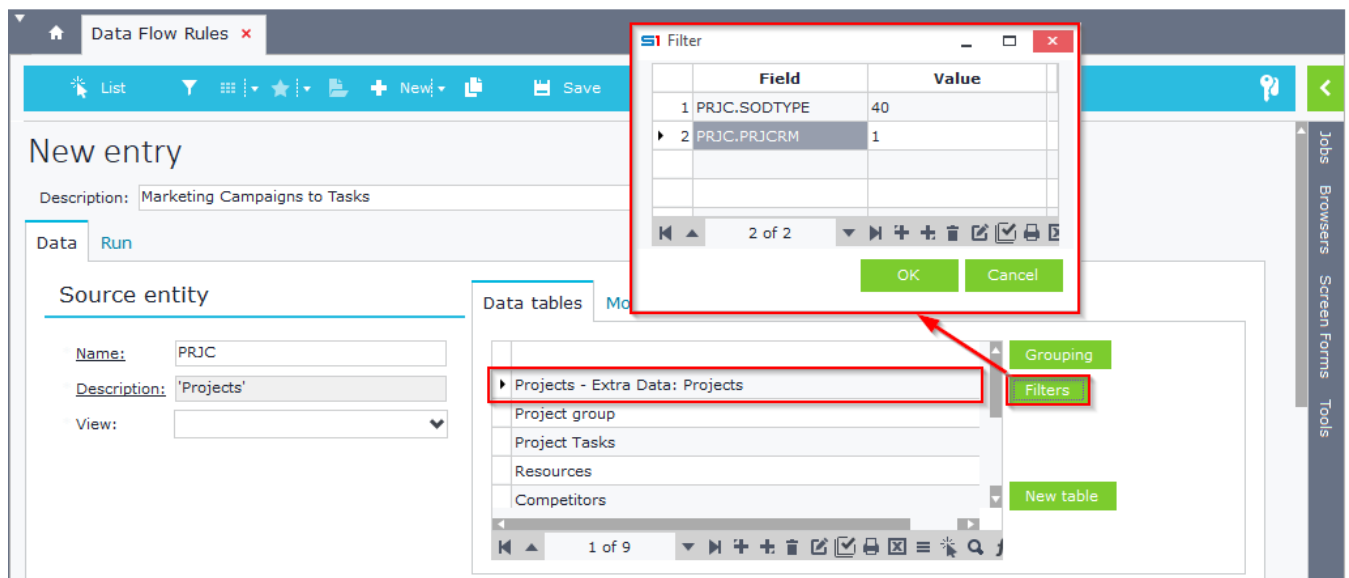


Figure E5.1

In the target entity enter the object SOTASK and select the desirable form, that has default values for specific fields. The assignment of values for the fields in the Tasks table is done as in Figure E5.2. Select the option "Display dialogue" in order to display the filters dialog when running the data flow scenario.

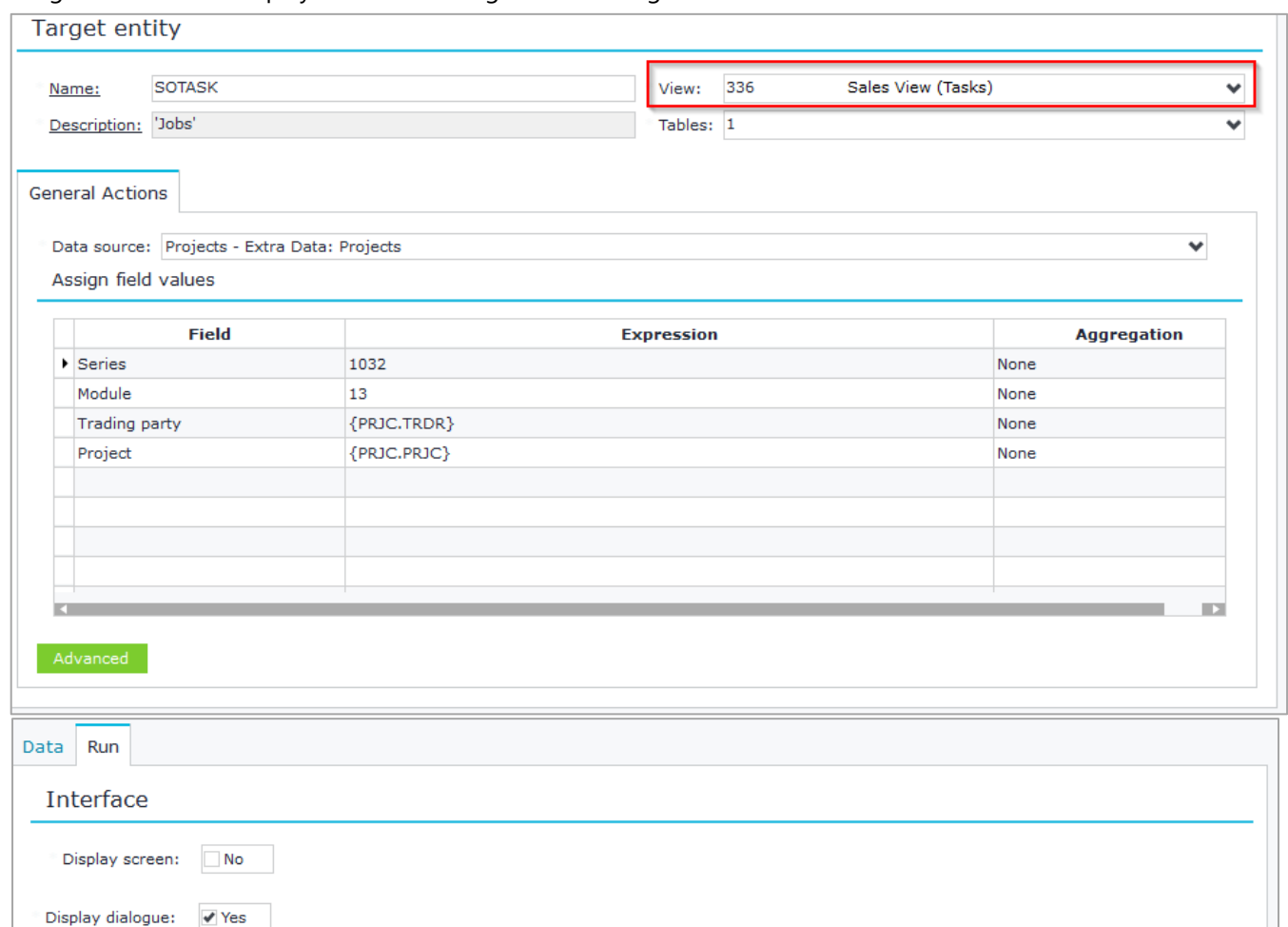


Figure E5.2

Create a new data flow scenario as in Figure E5.3. Run the data flow scenario through the marketing campaigns browser (Figure E5.4) the entries in the Tasks will be created as in Figure E5.5.

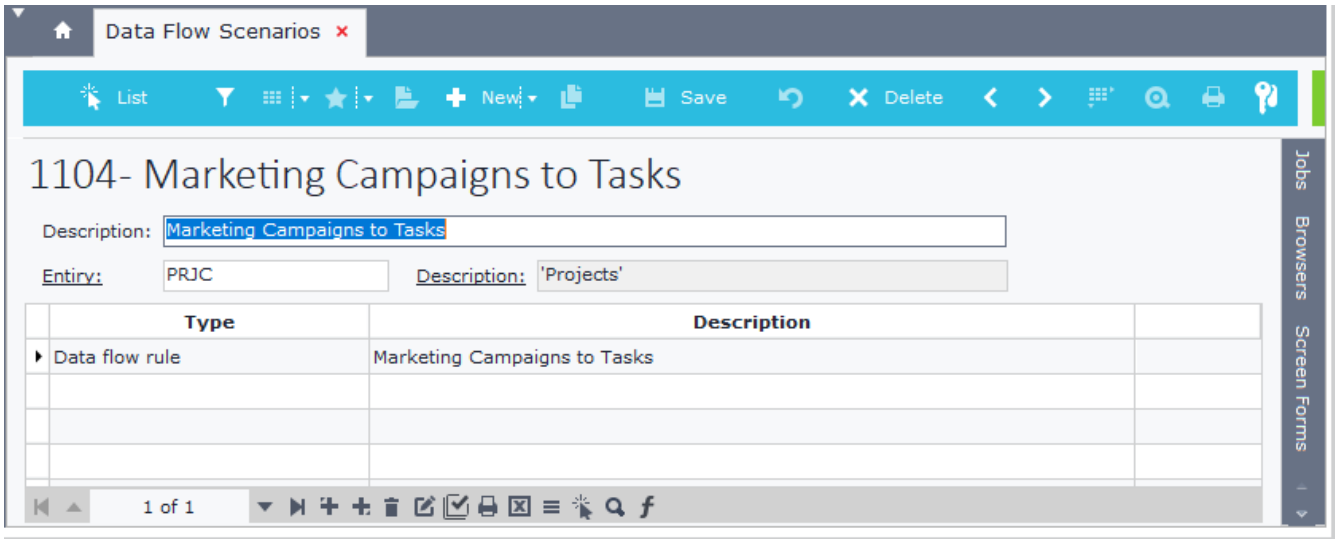


Figure E5.3

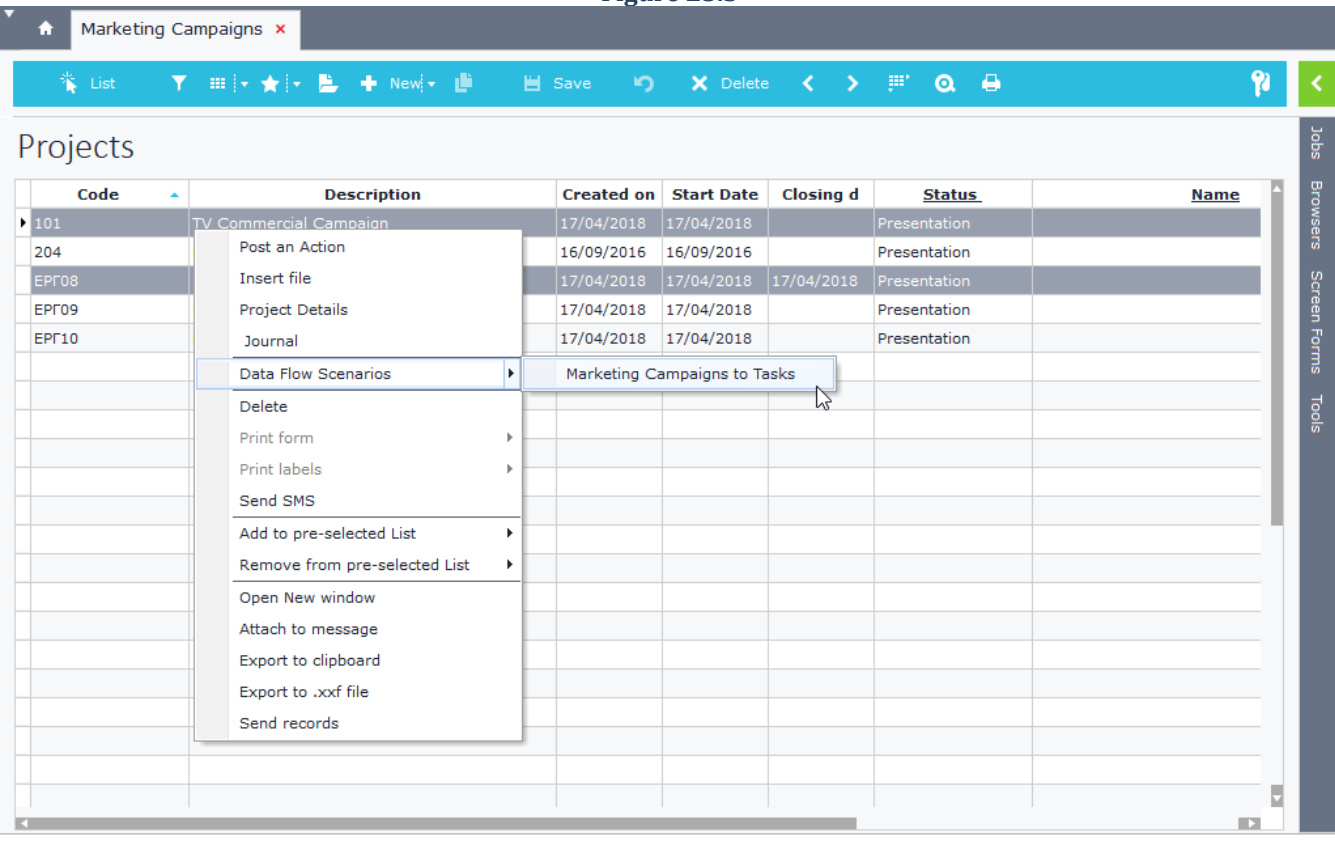


Figure E5.4

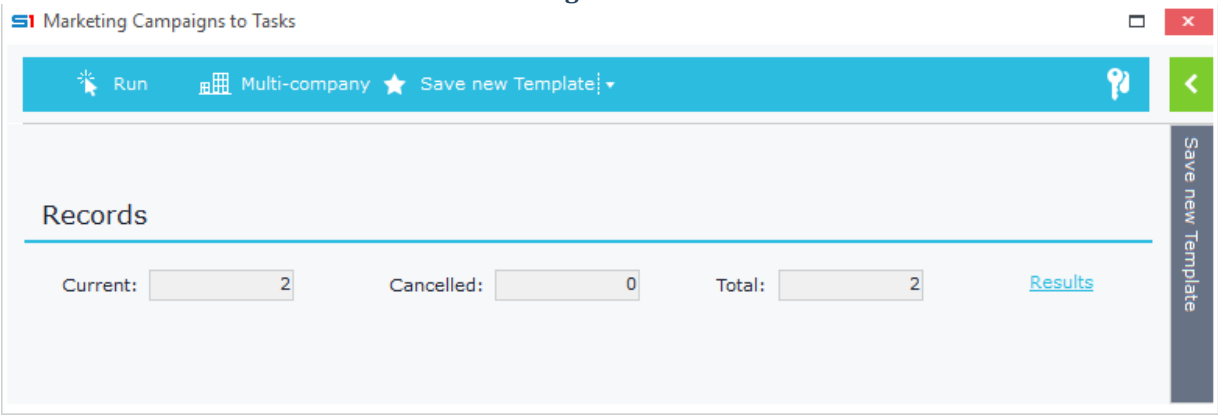


Figure E5.5

11. SOFTONE BATCH SCRIPT LANGUAGE (SBSL)

- A. [Overview](#)
- B. [Syntax Basics](#)
- C. [Dialog screen design \(Interface\)](#)
- D. [Main Code](#)
- E. [SBSL Script Execution](#)
- F. [Built-in Functions](#)
- G. [SoftOne Libraries](#)
- H. [Case Studies](#)

A. Overview

S1 Batch Script Language (SBSL) is a programming language developed initially by SoftOne for data import purposes from other applications. This language quickly proved to be a very powerful tool that can be used for process design through batch jobs.

One of the numerous advantages of SBSL compared to SQL scripts is that it follows the rules and logic of the application - something that cannot be done directly with SQL statements. This means for example that it inserts - updates or deletes records in SoftOne tables, in the same way that a user could do throughout the objects interface.

Scripts in SBSL produce objects similar to batch jobs whose interface (dialog parameters) is designed by code. SBSL scripts can be incorporated inside the section "S1 scripts" or can be saved in files with ".imp" extension.

With the use of SBSL scripts you can develop numerous batch jobs such as:

- Import / modify data in any SoftOne object
- Export data from SoftOne objects to ASCII files
- Transfer data to external SQL, Oracle, Access tables
- Create XXF file from any object of SoftOne
- General Batch job execution (e.g. cost price update)

SBSL scripts can be executed using:

- Right-click on selected browser records
- Menu jobs
- Functions in form screen scripts

Figure A1 shows an example of a batch job dialog screen.

SI Import Script

Run Multi-company Save new Template

Import Script

Current: Canceled: Total:

Messages:

Jobs

Job: Refresh Data SodType: Invoice Account Debit/Credit:

Entries of Doses (Filters)

From Date: From Supplier Code: To Supplier Code: From Invoice Code: To Invoice Code:

Entry (Mandatory select series)

Entry Series: Entry Date: 07/06/2018 ... or Date of Dose: Yes

Entry Comments:

Entries

Invoice Code	Invoice Value	Supplier Code	Supplier Name	Date	Value	Invoice Account	Account 2	Payment
								Yes

-1 of 0

Figure A1

B. Syntax Basics

The following sections include information about the different elements that comprise the SBSL language.

B.1 Case Insensitive

SBSL is case insensitive which means that all commands, variables and functions can be written using either small or capital letters.

B.2 Semicolons

SBSL statements must be terminated with semicolons (;).

B.3 Comments

Comments syntax is the same as in JavaScript, C# etc. using //.

Example:

```
// my comments
```

B.4 Variables

Variable are declared with a var statement. Their type is variant; when used their type is recognized automatically.

Example:

```
var x, y, temp;
```

B.5 Libraries References

References to built-in SoftOne libraries or to custom "code libraries" is made by using the keyword "include".

Examples

```
include 'PILib';
include 'ModuleIntf';
include 'SysRequest';
include 'myscript';
```

B.6 Functions

Functions are written as code blocks (inside curly {} brackets) preceded by the keyword "function".

```
function PrintOnScreen(fMess)
{
    x=SendResponse(ReplaceStr(fMess,'...',''), 'ImpTable.vMess');
}
```

The code inside functions will be executed when it is called from the SBSL script.

```
x = PrintOnScreen('Complete job execution');
```

Functions can also return values back to where the call was made. This can be done by using the "return" statement.

```
function myfunctionname(var1,var2) {
    //testcode
    return var1*var2;
}
x = myfunctionname(10,5); //assigns the value 50 to the x variable
```

B.7 Error Messages

Custom error messages can be created using the RaiseException function.

```
x = RAISEEXCEPTION('Process Stopped!');
```

B.8 Comparison symbols

Comparison is done using the following symbols

=	Equal
<>	Not equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

B.9 Logical operators

SBSL provides the following logical operators

AND	Conjunction
OR	Disjunction
NOT	Negation

C. Dialog screen design (Interface)

The first section of the SBSL script consists of the commands that design the dialog screen.

The design primitives are the same as in screen forms. Fields, tables, tabs, panels and datagrids can be displayed in the dialog screen as long as they are inserted inside block Form. Form block may include the sections [TABLES], [CACHETABLES], [PANELS] and [STRINGS].

```
Form {
  [TABLES]
  ...
  [CACHETABLES]
  ...
  [PANELS]
  ...
  [STRINGS]
  ...
}
```

C.1 Section TABLES

Inside this section you define the virtual tables (Datasets) that will be used in the dialog window.

A table is defined by stating its name (which shall be used in order to refer to the table within the script) as well as its caption, using the following syntax.

TableName = ;;;TableCaption;3;0

Example (Ex C.1)

```
[TABLES]
HeaderTable1 = ;;;MyHeader;3;0
GridTable1 = ;;;MyGrid;3;0
CacheTable1 = ;;;MyCache;3;0
```

C.1.1 Table / Dialog Fields

The declaration of the fields inside the tables is done with the creation of sections for each TableName entered in Tables section. Fields are imported using the following syntax.

FieldName = DataType; DisplaySize; Required; Visible; ReadOnly; Caption; Editor; TblEdit; Forced; Default; Decimals

PROPERTY	DESCRIPTION
DataType	Field type: 1 =String, 2 =Smallint, 3 =Integer, 4 =Word, 6 =Float, 7 =Currency, 9 =Date, 11 =DateTime, 16 =Memo
DisplaySize	Field width
Required	Defines if the field is mandatory or not. 0 = not required, 1 = required
Visible	0 = not displayed, 1 = displayed
ReadOnly	0 =allows data modification in the field, 1 =does not allow data modification
Caption	Label of the field
Editor	Field editor (See C.7 Selector List , G. Editor Attributes)
TblEdit	Internal use only
Forced	Forced field value
Default	Default field value
Decimals	Decimal points (for numeric fields). If used in memo fields then it shows the scrollbars (1=Horizontal, 2=Vertical, 3=Both).

Examples

1. Small integer field displayed as Boolean control (Yes/No) with caption "Job confirmation" and default value "Yes".

```
[HeaderTable1]  
vImpOk=2;15;1;1;0;Job confirmation;$Y;;;1
```

2. Small integer field displayed as combobox control populated with the sales series (using editor), with default value "7021".

```
vSalSeries=2;30;1;1;0;Sales series;SERIES (F[SOSOURCE=1351]);;;7021
```

3. Date field with default value the login date.

```
vDate=11;25;1;1;0;From insertion date;;;X.SYS.LOGINDATE
```

4. Text field that can be used in memo textbox for displaying job log messages.

```
vImpMess=16;64000;0;1;1;Job messages...;;;
```

5. Small integer field displayed as combobox control populated with the commercial categories of items (using editor).

```
vItemCategory=2;25;1;1;0;Commercial category;ITECATEGORY;;;
```

6. String field displayed as text picker control populated with items (using editor) that returns the value of the field "code".

```
vItemCode=1;25;0;1;0;From item code;ITEM (M,R[CODE]);;;
```

7. String field displaying a file picker dialog box. Returns file path.

```
vFile=1;256;1;1;0;Pick File;$Filename;;;
```

8. Small integer field displayed as hyperlink button that can be used to execute a job. The button of this particular example displays the object "CUSTOMER" when clicked.

```
vButton=2;15;0;1;0;Button;XCMD:CUSTOMER;;;
```

9. Small integer field displayed as combobox control populated with warehouses (WHOUSE editor) and filtered from other dialog filters (vBranch and vFilter).

```
vWhouse=1;15;0;1;0;W.h..;WHOUSE (W[EXISTS (SELECT 1 FROM BRANCH B WHERE B.COMPANY=  
A.COMPANY AND B.BRANCH=:ImpTable.vBranch AND  
CHARINDEX (:ImpTable.vFilter+CAST (A.WHOUSE AS  
VARCHAR)+:ImpTable.vFilter,:ImpTable.vFilter+B.WHOUSE+:ImpTable.vFilter)>0) ],Z,M,R[W  
HOUSE]);
```

10. Small integer field displayed as combobox control populated with MTRUNIT editor that auto updates (U) another dialog field (vMtrUnit1).

```
vMtrunit1Init=1;40;0;1;1;UOM of initial code;MTRUNIT (U[vMtrunit1=MTRUNIT@]);;  
vMtrunit1=1;40;0;1;1;U.O.M.1 of New code;MTRUNIT;;
```


C.1.2 Browser Selected Records

SBSL scripts executed through right click in browsers return the selected browser records using the command **&SELRECS**, which must be used inside a field of the section [TABLES]. **&SELRECS** returns the primary keys of the selected records either whether they are header tables or lines analysis tables.

The command **&SELRECS** must be used in a text dialog variable and the syntax is as follows:

vWhere=16;80000;0;1;0;Records selected from Browser;;;&SELRECS;&SELRECS

Example

Browser selected records (SQL where clause)

```
Form {
  [TABLES]
    ImpTable=;;;3;0
  [ImpTable]
    vWhere=16;80000;0;1;0;Selection of documents from Browser;;;&SELRECS;&SELRECS
}

Connect Xplorer SoftoneCon {
  connect();
  sDoc = SELECT FINDOC.FINCODE, FINDOC.TRDR
          FROM FINDOC
          WHERE ($REMOVEQUOTES(:$ImpTable.vWhere));
}

Var x;
{
  //fetch sDoc{}
  x = ShowWarning(VarToStr(:ImpTable.vWhere));
}
```

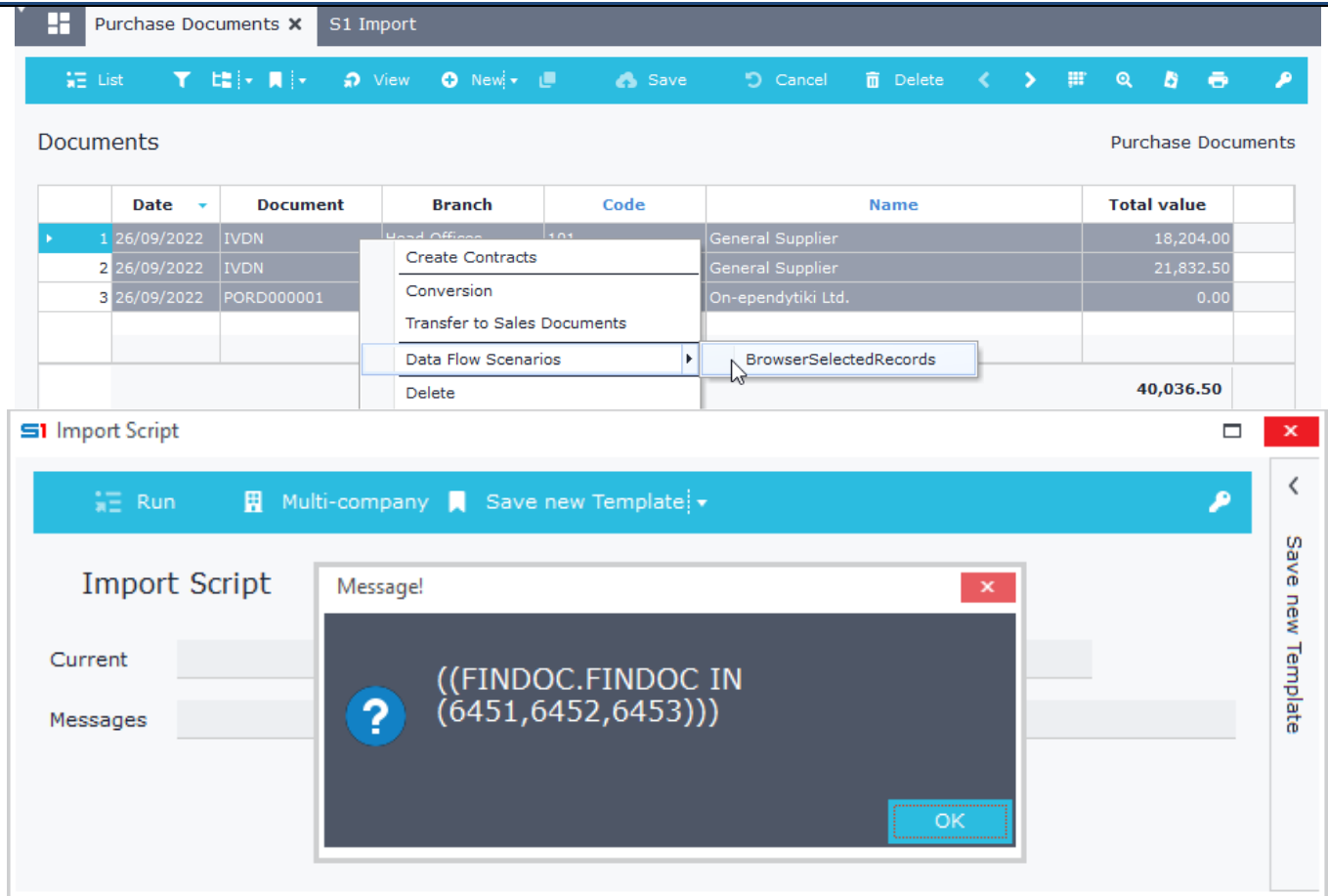


Figure C1.1

C.2 Section CACHETABLES

This section contains datasets (cache tables) that are populated from SQL statements and **run during script initialization** and is usually used for initialization of dialog filter fields.

These datasets can be used either as editors in fields or anywhere else inside the main SBSL script.

Example 1

Field vPayment uses a dynamic editor (TblPayment) which is filtered by a user-selected field (vSodtype).

```
Form {
  [TABLES]
    ImpTable=;;;3;0
    TblPAYMENT=;;;TblPAYMENT;3;0

  [ImpTable]
    vSodtype=2;25;1;1;0;Type SODTYPE;$SODTYPE(W[12,13]);;
    vPayment=2;25;1;1;0;Payment;TblPAYMENT(W[SODTYPE=:ImpTable.vSodtype AND
COMPANY=:X.SYS.COMPANY]) ;;;

  [TblPAYMENT]
    PAYMENT=2;5;0;1;0;
    NAME=1;30;0;1;0;
    COMPANY=2;5;0;0;0;
    SODTYPE=2;5;0;0;0;

  [CACHETABLES]
    TblPAYMENT=SELECT PAYMENT,NAME,COMPANY,SODTYPE AS SODTYPE FROM PAYMENT

  [PANELS]
    PANEL11=0;;0;50

  [PANEL11]
    ImpTable.vSodtype
    ImpTable.vPayment
}
```

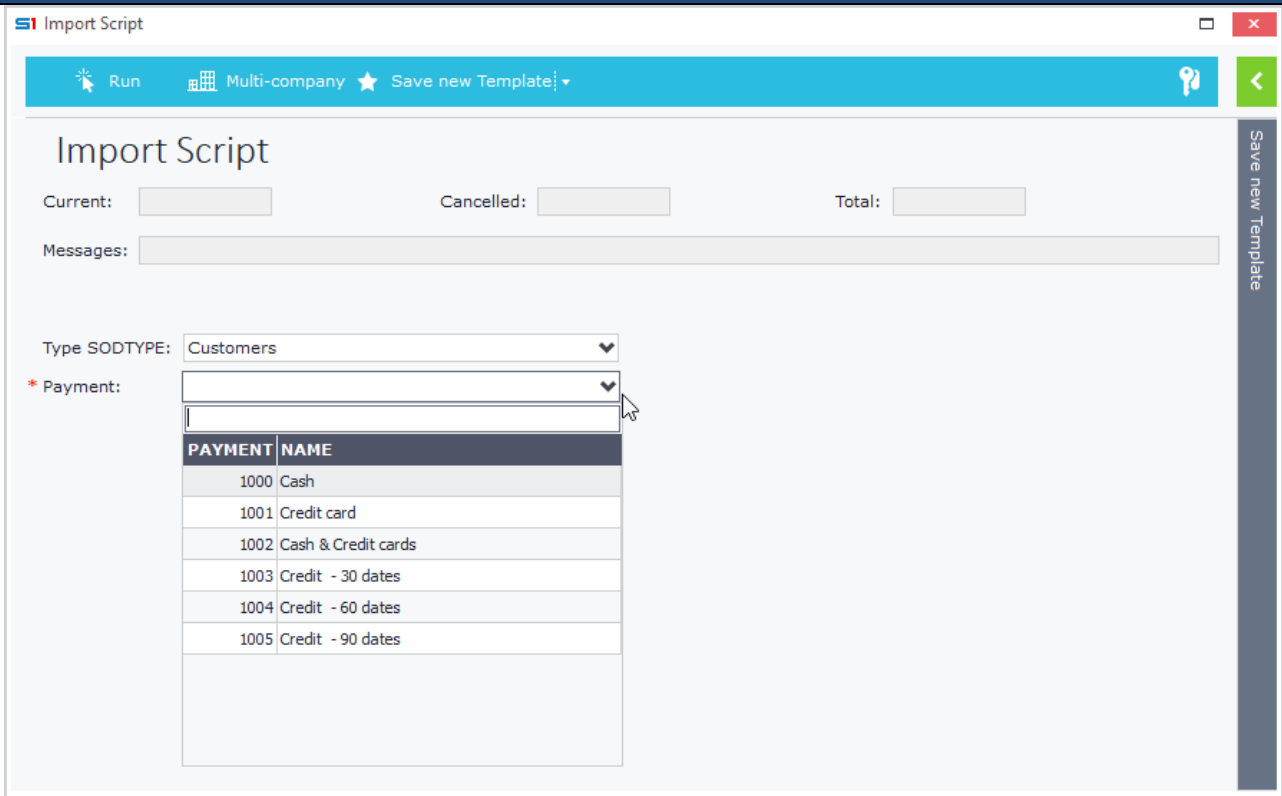


Figure C2.1

Example 2

Field vPayment uses a dynamic editor (TblPayment) which is filtered by browser selected records (Sales).

```
include 'SysRequest';
Form {
[TABLES]
ImpTable = ;;;;Master;3;0
TblPAYMENT=;;;TblPAYMENT;3;0
[ImpTable]
vPayment=2;25;1;1;0;Payment;TblPAYMENT(W[SODTYPE=13 AND COMPANY=:X.SYS.COMPANY]);;;
vWhere=16;80000;0;1;0;Browser records selection;;;&SELRECS;&SELRECS
vMess =16;64000;0;1;1;;;;;

[TblPAYMENT]
PAYMENT=2;5;0;1;0;;;;;
NAME=1;30;0;1;0;;;;;
COMPANY=2;5;0;0;0;0;;;;;
SODTYPE=2;5;0;0;0;0;;;;;

[PANELS]
PANEL1 = 0;;0;30,30,30,0,N,H25
PANEL2=4;Messages;0;H100

[PANEL1]
ImpTable.vPayment
[PANEL2]
ImpTable.vMess

[CACHETABLES]
TblPAYMENT=SELECT PAYMENT,NAME,COMPANY,SODTYPE AS SODTYPE FROM PAYMENT WHERE SODTYPE=13 AND
COMPANY=:X.SYS.COMPANY AND PAYMENT IN (SELECT FINDOC.PAYMENT FROM FINDOC WHERE
$RemoveQuotes(:ImpTable.vWhere) );
}

var x,UserResp,icnt,vSelRecs,vSelected,vMess,vImpTable,Count,DsFinDoc,module;

Connect Xplorer SoftOne {
connect();

vSelected=SELECT FINDOC,PAYMENT
FROM FINDOC
WHERE SOSOURCE=1351 AND COMPANY=:X.SYS.COMPANY AND ($REMOVEQUOTES(:$ImpTable.vWhere));
}

function PrintMessage(fMess){
    vMess = vMess+fMess+#13+#10;
    x = SendResponse(vMess, 'ImpTable.vMess');
}

{
Count = 0;
vSelRecs = VarToStr(GetParamValue(XModule,'SELRECS'));

Module = CallPublished('ModuleIntf.CreateModule','SALDOC,WARNINGS:OFF,NOMESSAGES:1');
DsFinDoc = CallPublished('ModuleIntf.GetDataset',VarArray(Module,'FINDOC',2));

Count = 0;
fetch vSelected {
    Count = Count + 1;
}
x = SendResponse(Count, 'RESULTS.TOTREC');

if (vSelRecs='') {
    x = RaiseException('No records are selected');
}

Count = 0;
fetch vSelected {
    x = PrintMessage('Payment: '+VarToStr(vSelected.PAYMENT)+' Findoc:
'+VarToStr(vSelected.FINDOC)+#13+#10);
    x = CallPublished('ModuleIntf.LocateModule',VarArray(Module,vSelected.FINDOC,2));
    Count = Count + 1;
    x = SendResponse(Count, 'RESULTS.CURREC');
    x = CallPublished('ModuleIntf.DatasetEdit',DsFinDoc);
    x = CallPublished('ModuleIntf.SetFieldValue',VarArray(DsFinDoc, 'PAYMENT', :ImpTable.vPayment, 3));
    x = CallPublished('ModuleIntf.PostModule', Module );
}
}
```

The screenshot shows the 'Dynamic Filter Editor' window for 'Sales Documents'. The 'Description' is 'Dynamic Filter Editor' and the 'Entity' is 'SALDOC'. Below this is a table with columns 'Type' and 'Description'. The 'Type' is 'Import script' and the 'Description' is 'CLIENTIMPORT,SCRIPTNAME:DynamicFilterEditor'.

Below the table is a list of documents with columns: Date, Document, Payment, Branch, Code, Name, and Total. The 'Payment' column is highlighted with a red box, and a context menu is open over it. The menu options are: Conversion, Transfer to, Remove pending, Restore pending, Mass Approve/Reject Documents, Documents processing, Discount credit notes calculation, Order planning, Recalculate, Create payment code, Update Accounting, Update myDATA Doc Code, Recalculate myDATA, Data Flow Scenarios, and Delete. The 'Data Flow Scenarios' option is selected, and a sub-menu is open showing 'Dynamic Filter Editor' and 'Sales to Purchases'.

	Date	Document	Payment	Branch	Code	Name	Total
1	10/06/2021	SISN000006	Credit - 30 dates	Head Offices	101	Salesconsul GmbH	254.86
2	09/06/2021	ORDW000004		Head Offices	923	073586841	6,150.00
3	09/06/2021	ORDW000005		Head Offices	923	073586841	6,150.00
4	09/06/2021	ORDR000006	Cash	Head Offices	109	Patrick Theodore	123.88
5	09/06/2021	SHNO000002				073586841	12,300.00
6	08/04/2021	ORDW000003				073586841	6,150.00
7	08/04/2021	SISN000002	Credit			Alexis Chalkidis	469.68
8	08/04/2021	SISN000004	Credit			Nick Panterben	2,618.35
9	08/04/2021	SHNO000001	Cash			Patrick Theodore	152.37
10	08/04/2021	SISN000005				073586841	6,150.00
11	07/04/2021	SISN000001	Credit			Nick Panterben	3,198.00
12	07/04/2021	ORDR000001	Cash			Patrick Theodore	123.88
13	06/04/2021	ORDR000002	Credit			Kouskousis George	22.95
14	05/04/2021	SISN000003	Cash			Gohemburger Ltd	4,688.07
15	20/11/2020	ORDR000006	Cash			Patrick Theodore	123.88
16	09/07/2020	SISN000004	Credit			Nick Panterben	3,198.00
17	08/07/2020	ORDR000002	Cash			Patrick Theodore	123.88
18	07/07/2020	ORDR000001	Credit			Kouskousis George	22.95
19	17/06/2020	PICK000003	Credit				1.35
20	13/06/2020	ORDW000005				Sales to Purchases	6,150.00

Figure C2.2 – Browser Selection (Distinct Payments are only 2)

The screenshot shows the 'Import Script' window. The 'Current' field is empty, the 'Cancelled' field is empty, and the 'Total' field is empty. The 'Messages' field is empty. The 'Payment' field is highlighted with a red box, and a context menu is open over it. The menu options are: 1000 Cash and 1003 Credit - 30 dates. The '1003 Credit - 30 dates' option is selected.

PAYMENT	NAME
1000	Cash
1003	Credit - 30 dates

Figure C2.3 – Field vPayment displays only the selected payments

C.3 Section PANELS

This section contains all the information needed to design the interface of the dialog screen. The available controls are stated through the following syntax.

ControlName = Control; Caption; Level; Info

PROPERTY	DESCRIPTION
Control	Control Type Definition: 0 =Panel, 1 =Tab, 2 =Datagrid, 4 =Memo textbox
Caption	Text that defines the caption of the control
Level	Number that defines the level of the control. Used for nested controls (panel inside tab)
Info	Properties of the Control (Comma Separated). Use any of the following properties: Columns width (i.e. Panel with 3 columns is defined using 50,30,20), G =Group Name (i.e. G10), N =Bottom Line (displayed only in old UI), L =Lines (i.e. 5 lines are defined using L5), H =Height (i.e. 20% is defined using H20), E0 =Expandable (not Expanded) E1 =Expandable (Expanded), F =Floating (usage only with other panels that have L – not H) Note: Old UI Page controls need to be floating – property F . (e.g. PANEL11 = 1;'Tab1';0;F)

Example

Design a dialog interface with 3 tabs. The 1st tab will have 2 panels and a memo textbox, the 2nd tab will have one panel with 2 horizontal split panels (50-50) and the 3rd tab will have a datagrid.

```
Form
{
  [TABLES]
    ImpTable=;;;Master;3;0
  [ImpTable]
    vImpOk      = 2;15;1;1;0;Job confirmation;$Y;;0;
    vSeries     = 3;30;1;1;0;Sales series;SERIES(F[SOSOURCE=1351]);;;7021
    vJOB        = 2;15;1;1;0;Job;JOB;;;&JOB
    vDate       = 11;25;1;1;0;Date;;;;;X.SYS.LOGINDATE
    vFile       = 1;4000;1;1;0;File;$Filename;;;C:\MyFiles
    vIteCat     = 2;25;1;1;0;Commercial category;ITECATEGORY;;;
    vMess       = 16;64000;0;1;0;Messages;;;
  [PANELS]
    PANEL11     = 1;'Page1-Areas';0;F
    PANEL112    = 0;'Area1';1;50;50;0,0,N,H10
    PANEL113    = 0;'Area2';1; 40;0,0,0,H10
    PANEL114    = 4;'Memo';1;H80
    PANEL12     = 1;'Page2-AreaSplit';0;F
    PANEL121    = 0;'AreaSplit';1;50;50;0,0
    PANEL1211   = 0;'AreaSplit 1';2;100;0,0,0
    PANEL1212   = 0;'AreaSplit 2';2;100;0,0,0
    PANEL13     = 1;'Page3-Grid';0;F
    PANEL131    = 2;'Grid';1;
  [PANEL112]
    ImpTable.vImpOk
    ImpTable.vFile
  [PANEL113]
    ImpTable.vSeries
  [PANEL114]
    ImpTable.vMess
  [PANEL1211]
    ImpTable.vIteCat
  [PANEL1212]
    ImpTable.vDate
}
```

```

[STRINGS]
    JOB=JOB
[JOB]
    1=Items import
    2=Documents import
}

```

When executing the above code, the dialog window will be displayed as in Figures C3.1, C3.2 and C3.3.

Figure C3.1

Figure C3.2

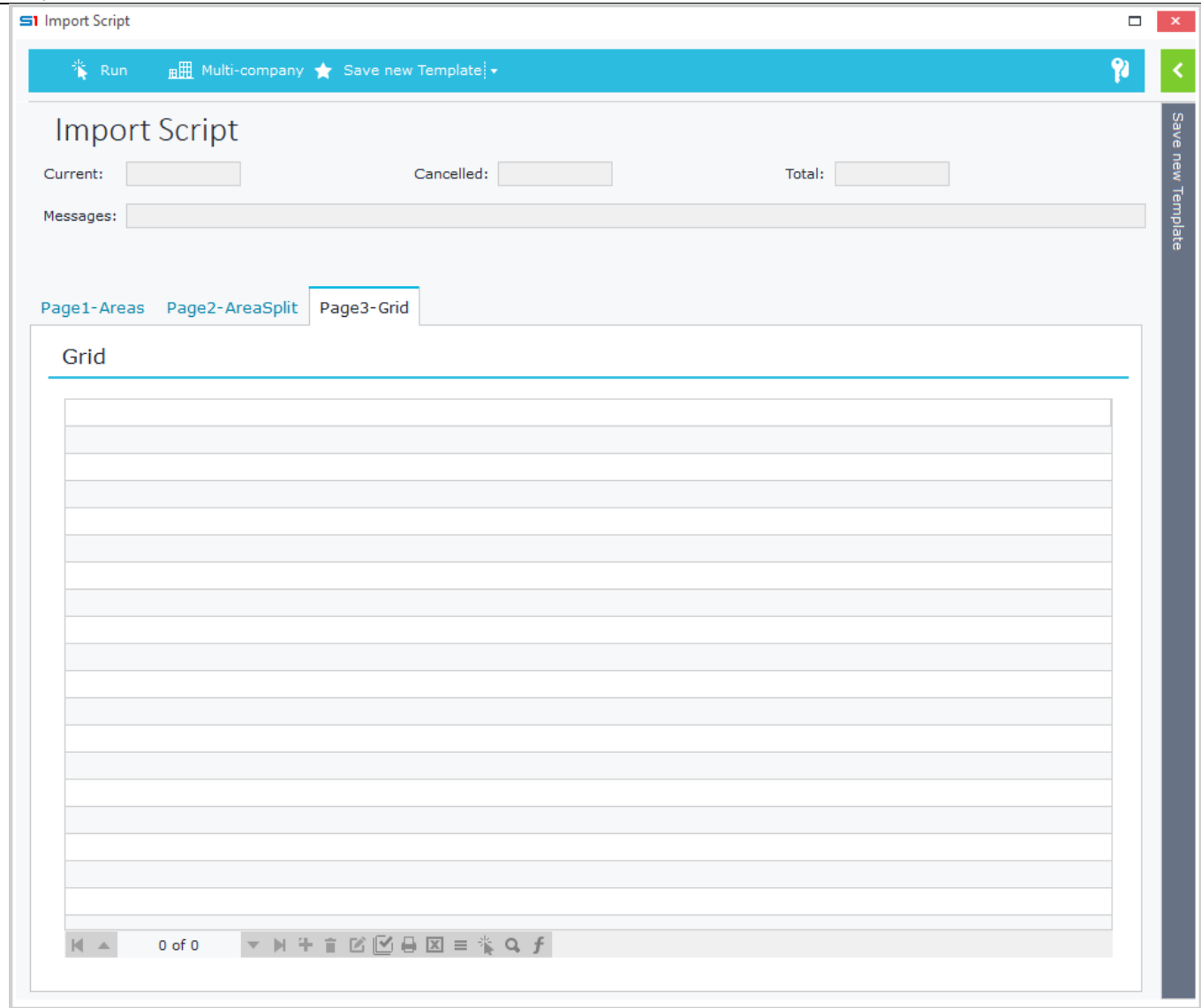


Figure C3.3

C.4 Section STRINGS

In this section you define the string lists to be used and displayed as editors in combo box fields in the same way as you state them in database designer or form designer. Syntax is the following:

StringListName = StringListName

Example (Figure C4.1)

*Design a dialog window with 3 combo boxes populated from string lists. This example uses also a string list named **SQLOrder** within the main code for sorting a SQL statement based on user selection. It also uses the string lists as parameters for the execution of the batch job (&MyString1, &Job, &SQLOrder).*

```
Form {
  [TABLES]
    ImpTable = ;;;;Master;3;0
  [ImpTable]
    vMyString1 = 2;15;1;1;0;StringList1;MyString1;;;&MyString1
    vJob = 2;15;1;1;0;StringList2;Job;;;&Job
    vSQLOrder = 1;15;1;1;0;StringList3;SQLOrder;;;&SQLOrder
  [STRINGS]
    MyString1 = MyString1
    Job = Job
    SQLOrder = SQLOrder
  [MyString1]
    1=Display New Codes
    2=Update Table
    3=Create New Password
    4=Create Composition Documents
  [Job]
    1=Documents export (One time)
    2=Repeat documents exports
  [SQLOrder]
    T.MTRMANFCTR = Manufacturer
    ML.LINENUM = Initial order
  [PANELS]
    PANEL1 = 0;'Area1';0;30,30,30,0,N,H100
  [PANEL1]
    ImpTable.vMyString1
    ImpTable.vJob
    ImpTable.vSQLOrder
}
Connect Xplorer SoftoneCon {
  connect();
  sSiteLines = SELECT ML.MTRL,ISNULL(ML.QTY1,0), ML.PRICE
                FROM MTRLINES ML, MTRL MT
                WHERE ML.FINDOC = :$sDoc.FINDOC AND ML.MTRL = MT.MTRL
                ORDER BY $REMOVEQUOTES(:$ImpTable.vSQLOrder);
}
```

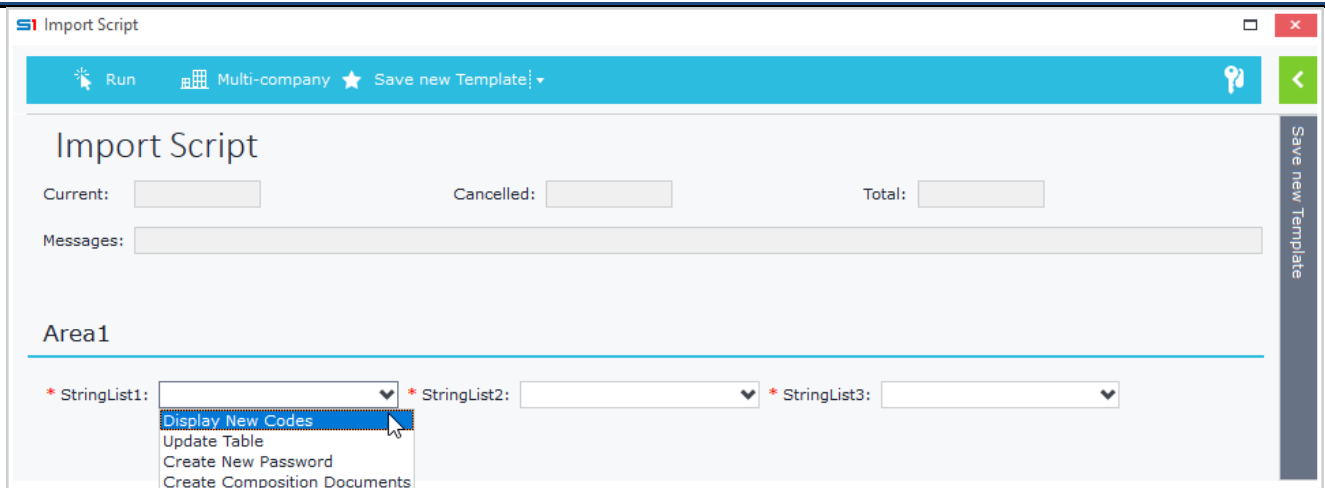


Figure C4.1

C.5 Section FIELDEXEC

Fields added in this section trigger the execution of the main script code upon each value modification performed by the user. This means that if for example you have added a string list field in this section, then when the user selects a value, the "Run" button will be activated and the script will be executed.

Using the appropriate "if" statement inside your code you can enable different processes to be executed depending on the selected values, such as filling a datagrid of the dialog filter or hiding panels.

Example

Design a dialog window that changes the visible property of panels and the caption of a field based on user selections. Note: In order to run the script, use the ClientImport execution method.

```
Form
{
  [TABLES]
    ImpTable =;;; Master; 3; 0

  [ImpTable]
    vImpOk    = 2;15 ;0;1;0;Job confirmation;$Y;;0;
    vJob      = 2;10 ;0;1;0;Job;JOB;;;
    vSeries   = 1;1024;0;1;0;Purchases series;#SERIES(W[SOSOURCE=1251 AND
(TFPRMS=102 OR TFPRMS=103 OR TFPRMS=151 OR TFPRMS=152) ] );;;
    vItemCode = 1;25;0;1;0;From item code;ITEM(M,R[CODE]);;;
    vFile     = 1;256;0;1;0;Pick File;$Filename;;;
  [FIELDEXEC]
    ImpTable.vJob

  [PANELS]
    PANEL1 = 0;Job;0;100,G10
    PANEL2 = 0;Series;0;100,G10
    PANEL3 = 0;Item;0;100,G10
    PANEL4 = 0;Select file;0;100,G10
    PANEL5 = 0;Job confirmation;0;100,G10

  [PANEL1]
    ImpTable.vJob
  [PANEL2]
    ImpTable.vSeries
  [PANEL3]
    ImpTable.vItemCode
  [PANEL4]
    ImpTable.vFile
  [PANEL5]
    ImpTable.vImpOk

  [STRINGS]
    JOB=JOB

  [JOB]
    1=Data import
    2=Data Export
}

include 'ModuleIntf';

Connect Xplorer Export
{
  connect();
}

{ //Main code
  Var x, vImpTable, vJob, vImpOk;

  vImpTable=GetDataset(xModule,'ImpTable');
  vJob =GetFieldValue(vImpTable,'vJob');
```

```

vImpOk =GetFieldValue(vImpTable,'vImpOk');

if (vJob=1)
{
    x=SetProperty(xModule, 'PANEL', 'USR_PANEL2', 'VISIBLE', 'TRUE'); // Panel
visible
    x=SetProperty(xModule, 'PANEL', 'USR_PANEL3', 'VISIBLE', 'FALSE'); // Panel
hidden
    x=SetProperty(xModule, 'PANEL', 'USR_PANEL4', 'VISIBLE', 'TRUE');
    x=SetProperty(xModule, 'FIELD', 'ImpTable.vFile', 'CAPTION', 'Import file');
}
else if (vJob=2)
{
    x=SetProperty(xModule, 'PANEL', 'USR_PANEL3', 'VISIBLE', 'TRUE');
    x=SetProperty(xModule, 'PANEL', 'USR_PANEL2', 'VISIBLE', 'FALSE');
    x=SetProperty(xModule, 'PANEL', 'USR_PANEL4', 'VISIBLE', 'TRUE');
    x=SetProperty(xModule, 'FIELD', 'ImpTable.vFile', 'CAPTION', 'Export file');
}
if (vImpOk=1) //The code below is executed only when the user clicks "Yes" in
field "Job confirmation"
{
    if (vJob=1)
    {
        //Add Code Here that will be executed when vJob=1
    }
    if (vJob=2)
    {
        //Add Code Here that will be executed when vJob=2
    }
}
}

```

The following figures show the different panels and field caption displayed according to the selection of the field "Job".

The screenshot shows the 'Import Script' window with the following elements:

- Header:** 'Import Script' title bar with 'Run', 'Multi-company', and 'Save new Template' buttons.
- Form Fields:** 'Current:', 'Cancelled:', and 'Total:' input fields.
- Messages:** A text area for messages.
- Table:** A table with columns 'A/A', 'Stage', 'Starts on', 'End Date', and 'Duration'.
- Job:** A dropdown menu showing 'Data import' (selected), 'Data import', and 'Data Export'.
- Series:** A dropdown menu for 'Purchases series'.
- Select file:** A section with an 'Import file' input field and a file selection button.
- Job confirmation:** A section with a 'Job confirmation' checkbox and a 'No' button.

Figure C5.1

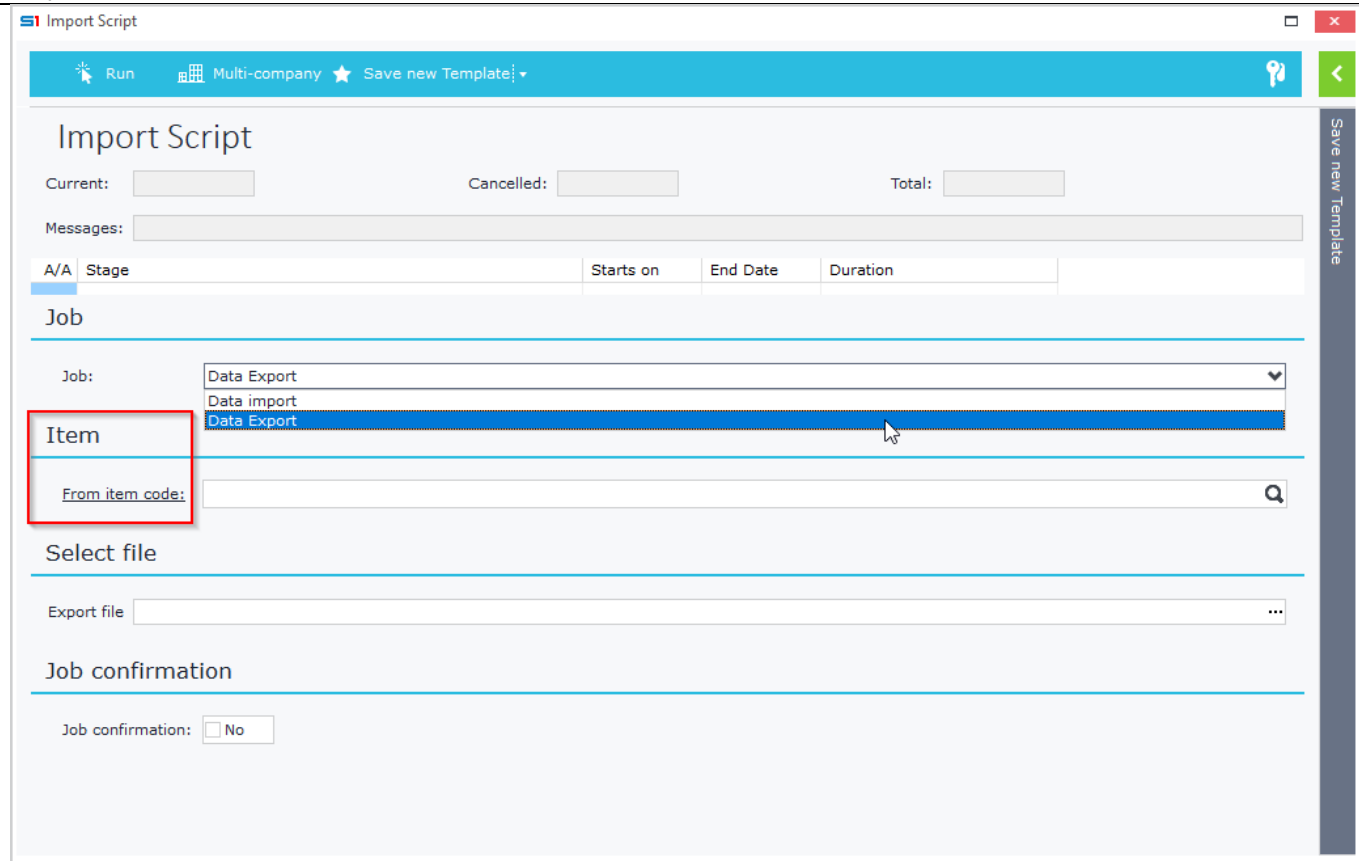


Figure C5.2

C.6 Section SCRIPT

This section uses JavaScript code to interact with dialog object and field events.

One way to use this section is when you need to add a calculated value as default filter value. The calculations may be entered inside the event "ON CREATE". You can also use it to populate a grid of dialog filters.

Another way to use this section is in cases of displaying different dialog filters (panels) based on field value changes. This means that you can trigger the on-change event of fields and show/hide panels according to field values.

Example

```
form {

  [TABLES]
  ImpTable=;;;Master;3;0

  [ImpTable]
  vFDateFrom  = 11;15;0;1;0;Date From;;;;;
  vFDateTo    = 11;15;0;1;0;Date To;;;;;
  vTxtFile    = 1;255;0;1;0;Choose File;$Filename;;;
  vOrderNo    = 1;32;0;1;0;New Order No;;;
  vImpMess=16;84000;0;1;1;Job Messages...;;;;;

  [PANELS]
  PANEL1=0;Filters;0;035,000,000,000,G10
  PANEL15=4;Job Messages...;0;100,L5

  [PANEL1]
  ImpTable.vFDateFrom
  ImpTable.vFDateTo
  ImpTable.vTxtFile
  ImpTable.vOrderNo

  [PANEL15]
  ImpTable.vImpMess

  [Script]
  //X.WARNING("test"); //Runs on init

  function ON_CREATE () {
    //X.WARNING("OnCreate Event");
    GetNewOrderTimestamp();
  }

  function ON_ImpTable_vFDateFrom(){
    X.WARNING("ImpTable.vFDateFrom changed");
    debugger;
    if (ImpTable.ISNULL("vFDateFrom") == 0) {
      ImpTable.vFDateTo =
X.FORMATDATE("dd/MM/yyyy",X.EVAL(' INCYEAR(ImpTable.vFDateFrom,1) '));
      ImpTable.vImpMess = "Data from OnChange event: " + ImpTable.vFDateTo;
    }
  }

  function ON_POST() { //On run button
    X.WARNING("POST Event");
  }
}
```

```

function ON_ImpTable_vTxtFile() {
    GetNewOrderTimestamp();
}

function GetNewOrderTimestamp() {
    var myDate = new Date();
    var year = "" + myDate.getFullYear();
    var month = "" + (myDate.getMonth() + 1); if (month.length == 1) { month = "0"
+ month; }
    var day = "" + myDate.getDate(); if (day.length == 1) { day = "0" + day; }
    var hour = "" + myDate.getHours(); if (hour.length == 1) { hour = "0" + hour; }
    var minute = "" + myDate.getMinutes(); if (minute.length == 1) { minute = "0" +
minute; }
    var second = "" + myDate.getSeconds(); if (second.length == 1) { second = "0" +
second; }

    ImpTable.vOrderNo = year + month + day + "_" + hour + minute + second;
    //X.FORMATDATE("yyyymmdd_hhMMss", X.SYS.CLIENTDATE);
}
}

```

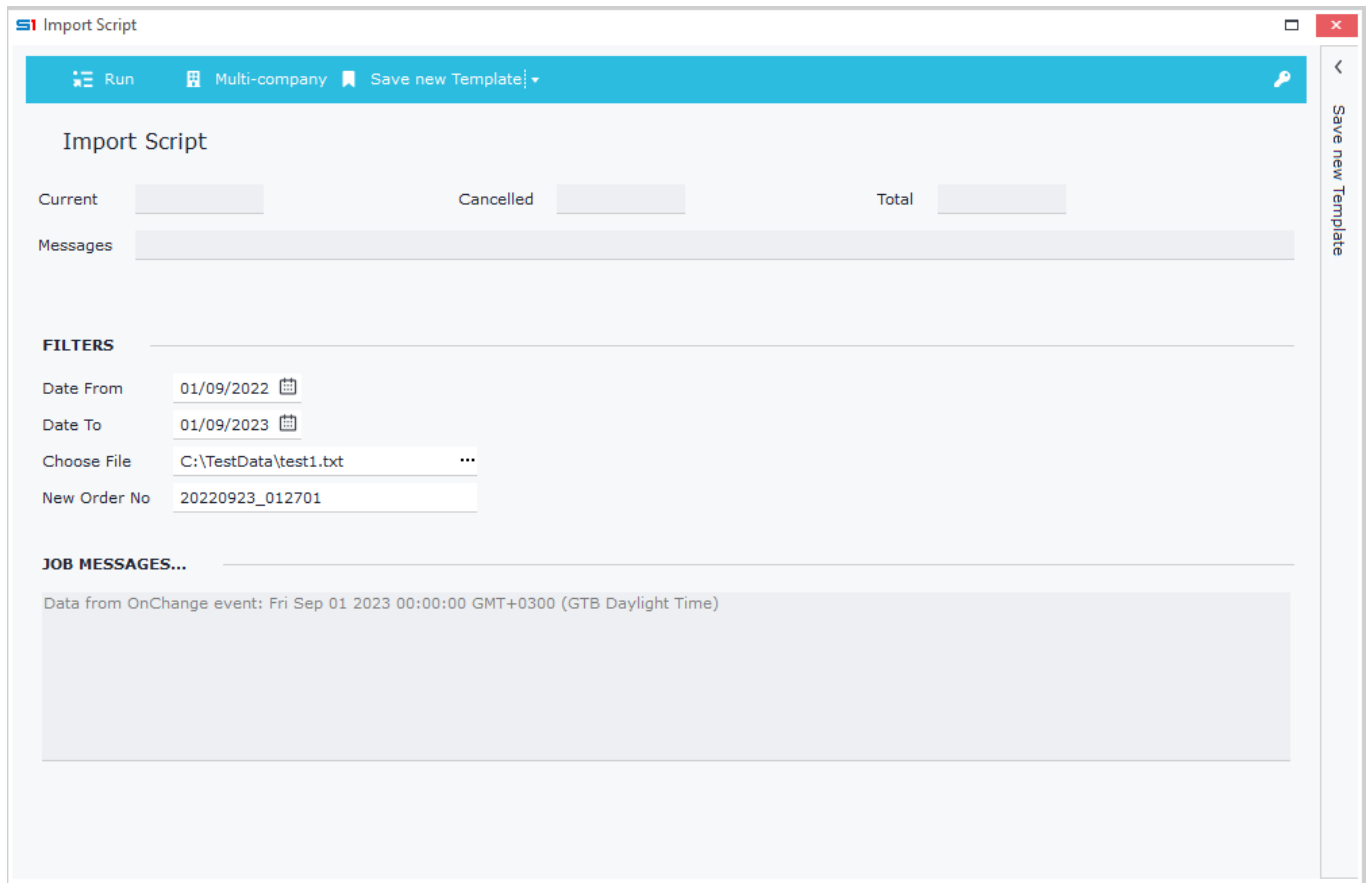


Figure C6.1

D. Main Code

D.1 Converters

In any place of the code, you can add a Converter that runs a SQL statement based on the parameters and when called returns the value of the return field defined. The syntax of Converters is the following:

Converter funcname (TableName:String, 'ParameterVariables:String', ReturnFieldName:String)

Examples

1. Converter that returns the company name based on company code given as parameter

```
Converter GetCompanyName (COMPANY, 'COMPANY', NAME);
GetCompanyName(1);
```

2. Converter that returns item id based on the company code and item code given as parameters

```
Converter ConvItem (MTRL, 'COMPANY;SODTYPE=51;CODE', MTRL);
MTRL = ConvItem(:X.SYS.COMPANY,'100-01');
```

3. Converter that returns customer id based on the company code and customer code given as parameters

```
Converter ConvCustomer (TRDR, 'COMPANY;SODTYPE=13;CODE', TRDR);
TRDR = ConvCustomer(:X.SYS.COMPANY,sSalDoc.CUSTOMERCODE);
```

D.2 Functions

Same as in converters you can define your own functions which can be called from anywhere inside the script and may be used to return values.

Examples

1. Function that displays a warning message

```
function ShowMessage(fMess)
{
    x = CallPublished('SysRequest.ExecuteXScript', VarArray(XModule, 1, 'function
RUN() { X.WARNING('+#39+VarToStr(fMess)+#39+');}', 'RUN', 4));
}
//...
x = ShowMessage('This is a warning message');
```

2. Function that adds text message in a memo textbox field (vMess)

```
function PrintOnScreen(fMess)
{
    x = SendResponse(ReplaceStr(fMess,'...',''), 'ImpTable.vMess');
}
//...
x = PrintOnScreen('Completed!'+#13+#10);
```

3. Function that formats the output of a string field by adding spaces until it meets the specified length given as parameter. This string field can be later used for the export of an ASCII delimited file.

```
function AddStrLine(mystr, icount)
{
    if (mystr = null)
    {
        mystr = '';
    }
    mystr = Copy(mystr,1, icount);
    return (mystr + Space(icount - Len(mystr)));
}
//...
vTextWrite = CallPublished('PILib.CreateText', :ImpTable.vPath3);
vLine = '';
vLine = vLine + AddStrLine('',1);
vLine = vLine + AddStrLine('VAT', 3);
vLine = vLine + AddStrLine('23,00', 5);
vLine = vLine + AddStrLine('VAT', 3);
vLine = vLine + AddStrLine(FormatFloat('0.000',HEADER.VAT3), 5);
vLine = vLine + AddStrLine(VarToStr(HEADER.VATVAL3), 14);
x = CallPublished('PILib.WriteLine', VarArray(vTextWrite, vLine, 2));
```

4. Function that returns the date in sql format

```
function getSQLDate(v1,conName)
{
    var sql,x, newd;
    newd = FormatSqlText(conName, ':1',v1);
    sql = 'Select convert(VARCHAR(8), '+newd+ ' , 112)';
    x = GetQueryResults(conName, sql, '');
    return x;
}
//...
newvMesDate = VarToStr(getSQLDate(:ImpTable.vMesDate,'S1Con'));
```

D.3 Connection Types

The connection types you can use inside SBSL scripts are SQL Server, Oracle and ODBC. For every connection you must define a specific name (i.e. ConS1), which can be later used as a reference inside the script. Notice though that you must always define a connection to SoftOne.

SoftOne

Connection to SoftOne login database (**must always be defined**)

Syntax

```
Connect Xplorer ConS1
{
    connect();
    dsfin = select findoc, trdr, ... from findoc where sosource=1351 and ... ;
}
```

SQL SERVER

Connection to SQL Server database

Syntax

```
Connect DBDriver DocData
{
    //Driver, DBase, ServerName, User, Password, DataBaseName
    connect('XADODrv.bpl', 'MSSQL', 'SERVER\MYSRV', 'sa', '123456', 'DBNAME');
}
```

ORACLE

Connection to ORACLE database

Syntax

```
Connect DBDriver SRSDData
{
    //Driver, DBase, ServerName, User, Password, DataBaseName
    connect('XODAC.BPL', 'ORACLE', 'SERVER', 'srvn1', 'srvn1', 'srvn1');
    //OldVersion connect('XADODrv.bpl', 'ORACLE', 'SERVER', 'srvn1', 'srvn1', 'srvn1');
    dsl = select fld1, fld2, ... from MYTABLE where myfld=1;
}
```

Syntax for Oracle 19 (add parameter USEOCI=1 and use Service Name instead of ServerName and Database)

```
Connect DBDriver SRSDData
{
    //Driver, DBase, ServiceName, User, Password, ServiceName
    connect('XOdac.bpl', 'oracle', '', 'usr', 'pass', '', 'USEOCI=1');
    dsl = select fld1, fld2, ... from MYTABLE where myfld=1;
}
```

ODBC

Connection through ODBC (Excel, Access, etc.) based on the connection string

Syntax

```
Connect DBDriver XlsData
{
    connect('XADODrv.bpl', 'CUSTOM', 'Provider=MSDASQL.1;Persist Security
Info=False;Data Source=XlsMtrl' );
    XlsMtrl = select fld1, fld2, fld3, ... from [mtrl$];
}
```


D.4 Variables

Script variables can be stated at any point of the code before being called. They are declared using the word “var” and their type is variant; when used their type is recognized automatically.

```
var vTot, vLabelText, vRow, vRowCancel, UserResp, x;
```

D.5 Use Libraries

In the main script code you can call functions that exist either in internal SoftOne libraries, or in libraries that you have created by using SBSL scripts. In order to use libraries, you have to add the “include” statement. Analysis of SoftOne libraries can be found in the following modules.

Examples

```
include 'PILib';
include 'ModuleIntf';
include 'SysRequest';
```

In order to create and use your own functions as libraries you have to add them in a SBSL script (i.e. MyLib) and then call them from any other SBSL script by using include statement.

Example:

```
include 'MyLib';
```

D.6 Execute Javascript from SBSL

Javascript can be executed inside SBSL scripts using the following steps.

Step 1. Create Javascript functions

Add a new SBSL script, enter your JavaScript Functions and then Save it (e.g. MyJavFunctions).

Example

```
function MyWarning() {
    X.WARNING('Using JavaScript functions');
}
```

Step 2. Call Javascript functions from SBSL

Add the following lines inside your main SBSL script, replacing MyWarning with the function you want to execute.

```
var jScript;
jScript = VarToStr(GetQueryResults('Softone', 'SELECT SOIMPORT FROM SOIMPORT WHERE
CODE='+#39+'MyJavFunctions'+#39, Null));
x = CallPublished('SysRequest.ExecuteXScript', VarArray(XModule, 1, jScript,
MyWarning, 4));
```

Pass parameters from SBSL to Javascript

a) Create your parameters inside the main SBSL script using the function SetParamValue

```
x = CallPublished('ModuleIntf.SetParamValue', VarArray(sModule,
'MyMessage', :ImpTable.vParameter1, 3));
var jScript;
jScript = VarToStr(GetQueryResults('Softone', 'SELECT SOIMPORT FROM SOIMPORT WHERE
CODE='+#39+'MyJavFunctions'+#39, Null));
x = CallPublished('SysRequest.ExecuteXScript', VarArray(XModule, 1, jScript,
MyCustomWarning, 4));
```

b) Get parameters from the script that contains the Javascript functions using GETPARAM

```
function MyCustomWarning() {
    var strMessage = X.GETPARAM('MyMessage');
    X.WARNING(strMessage);
}
```

E. SBSL Script Execution

The scripts created using the SBSL can be executed from the menu using the following ways:

FORMIMPORT : Runs script on the server and transfer results to the client

CLIENTIMPORT : Runs script from the client

The first method (run on server) is clearly faster than the second one (client / server environment).

The call action using clientimport is used when the script must have access to local resources (screen, keyboard, disks and file system). If for example the user must interact with the screen displayed after running the script (i.e. document conversion including the display of the document screen before posting) then clientimport must be used. Another example of use of clientimport is when importing or exporting files to disk.

A third method to run a script is **CLIENTIMPORT2** command; this method does not differ in the mode of data transfer to the client (CLIENTIMPORT), it simply changes the appearance of the dialog window (first line with dialog parameters is not displayed).

E.1 Menu Execution

SBSL scripts that are built to run as menu jobs must be created as follows:

Job type: Batch Job

Command (Server): FORMIMPORT,SCRIPTNAME:NameOfScript

Command (Client): CLIENTIMPORT,SCRIPTNAME:NameOfScript (Figure E1.1).

If the script is stored in a file, then you enter the entire file path (Figure E1.2).

The screenshot shows a window titled "S1 Job parameters". It has two tabs: "General Data" and "Properties". The "General Data" tab is selected. Inside this tab, there are three main input areas: "Job type:" with a dropdown menu showing "Batch job"; "Command/File:" with a text box containing "FORMIMPORT,SCRIPTNAME:Example1"; and "Job title:" with a text box containing "Example 1". To the right of the "Command/File:" text box is a green button labeled "Available jobs". At the bottom right of the window are two green buttons labeled "OK" and "Cancel".

Figure E1.1

This screenshot is similar to Figure E1.1, showing the "S1 Job parameters" window with the "General Data" tab active. The "Job type:" dropdown is still "Batch job". The "Command/File:" text box now contains "CLIENTIMPORT,SCRIPTNAME:C:\S1Scripts\Example2.imp". The "Job title:" text box contains "Example 2". The "Available jobs" button and "OK"/"Cancel" buttons are also present at the bottom.

Figure E1.2

E.2 Browser Execution

Execution of SBSL scripts through right click from browsers can be performed in two ways, either using the file OBJINFOS.CFG, or the SoftOne tool [Data Flow Scenarios](#) (recommended).

E.2.1 Run using file OBJINFOS.CFG

If you choose this method, then you need to create a file named OBJINFOS.CFG in the application folder of each client. The scripts called this way, are displayed in the bottom of the pop-up menu appearing upon right clicking on browser records (Figure E2.1).

Within the file OBJINFOS.CFG (Figure E2.2) you define the objects which will run the SBSL scripts and the titles of the jobs to be displayed on right click, using the following syntax.

```
[OBJECTNAME_BRMENU]
1;Job title = FORMIMPORT,SCRIPTNAME:nameofscript
1;Job title = CLIENTIMPORT,SCRIPTNAME:nameofscript
```

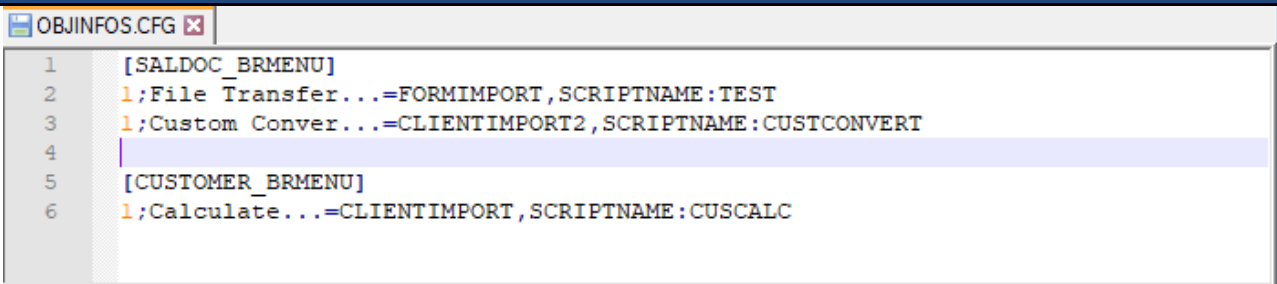


Figure E2.1

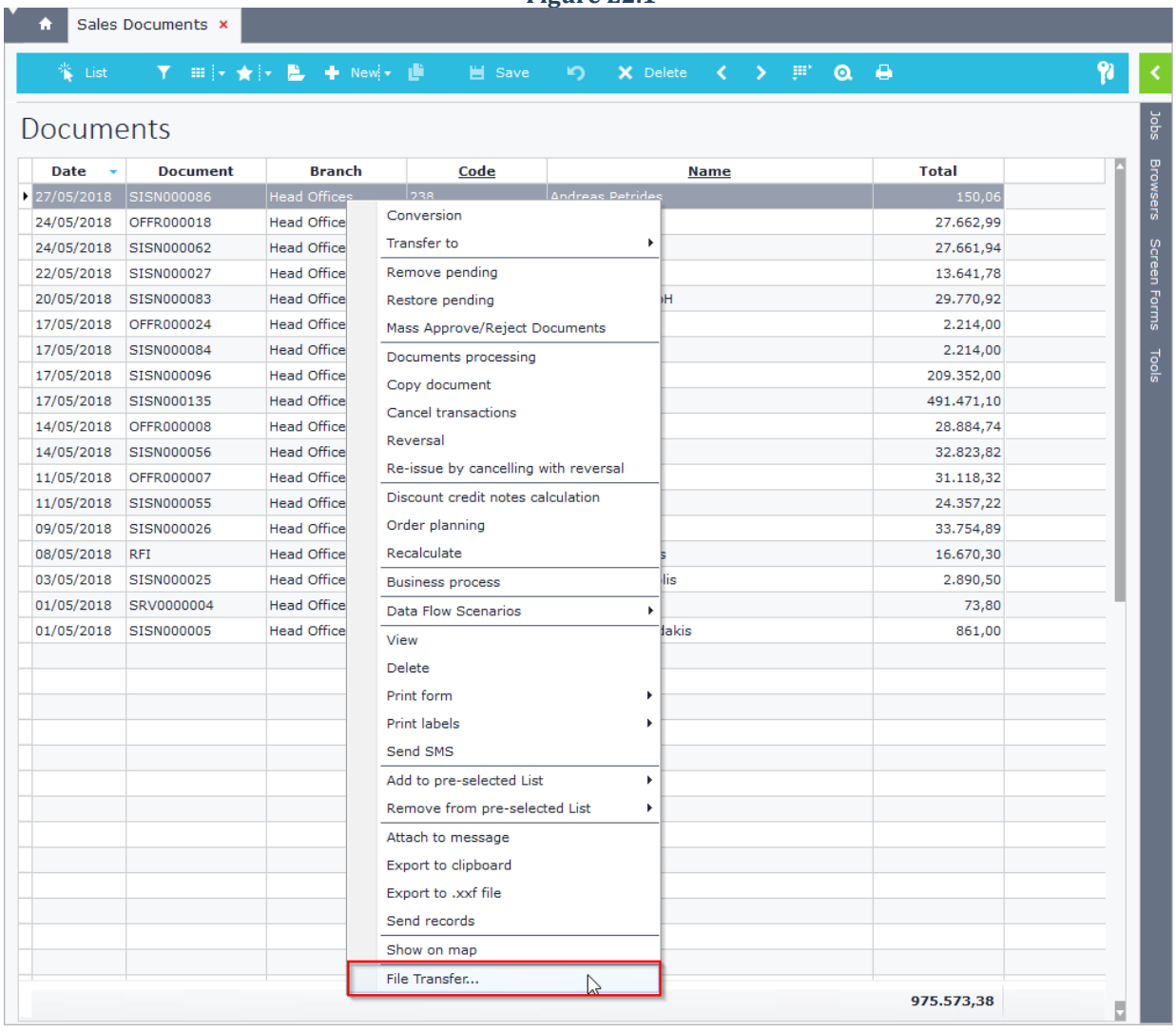


Figure E2.2 – Sales documents pop up menu displaying custom job “File Transfer...”

E.2.2 Run using Flow Scenarios

Execution of SBSL scripts through flow scenarios (Figure E2.3) gives you the option of defining user rights for the scenario (and thus for the SBSL script) using the same way as the definition of rights in any other module.

Another difference between calling SBSL script from a flow scenario and calling it from the file OBJINFOS.CFG is the location of the job displayed under pop-up menus. The jobs are displayed and executed from a submenu of the right click menu, which is named "Flow scenarios" (Figure E2.4).

Example (Ex E.1)

Display the script named "test1" in the pop up menu of sales documents using a flow scenario.

Create a new flow scenario and enter the "Description", that indicates the title of the job that will appear upon right clicking.

Inside "Entity" enter the SoftOne object (SALDOC) where the script will be displayed and executed from.

Inside the datagrid insert a new line, select "Import script" type and in description enter the command for the SBSL script named "test1", which is: FORMIMPORT, SCRIPTNAME: test1 (Figure E2.3)

Restart the application, open the SoftOne object "Sales Documents", select any records and then by right click notice the new job, under the submenu "Data flow scenarios" (Figure E2.4).

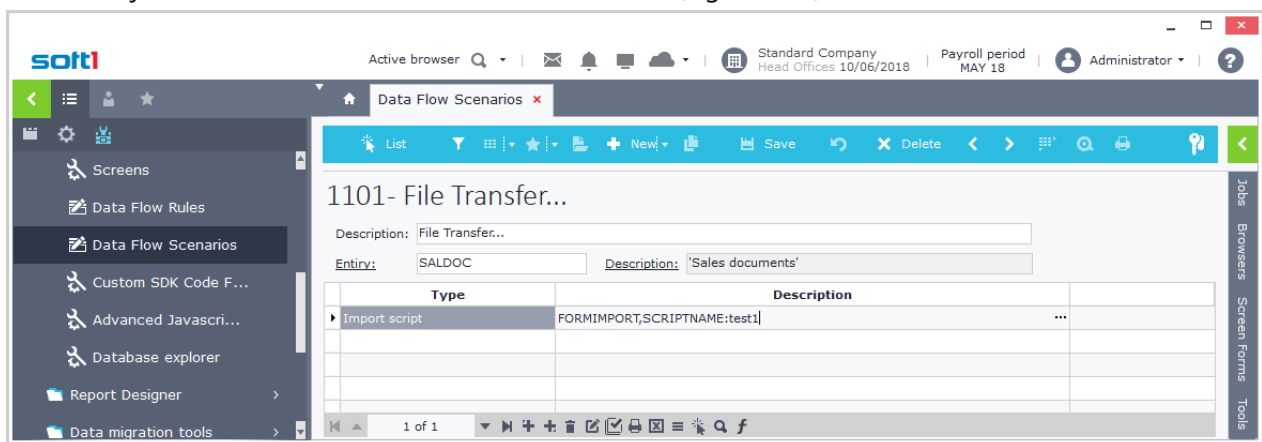


Figure E2.3

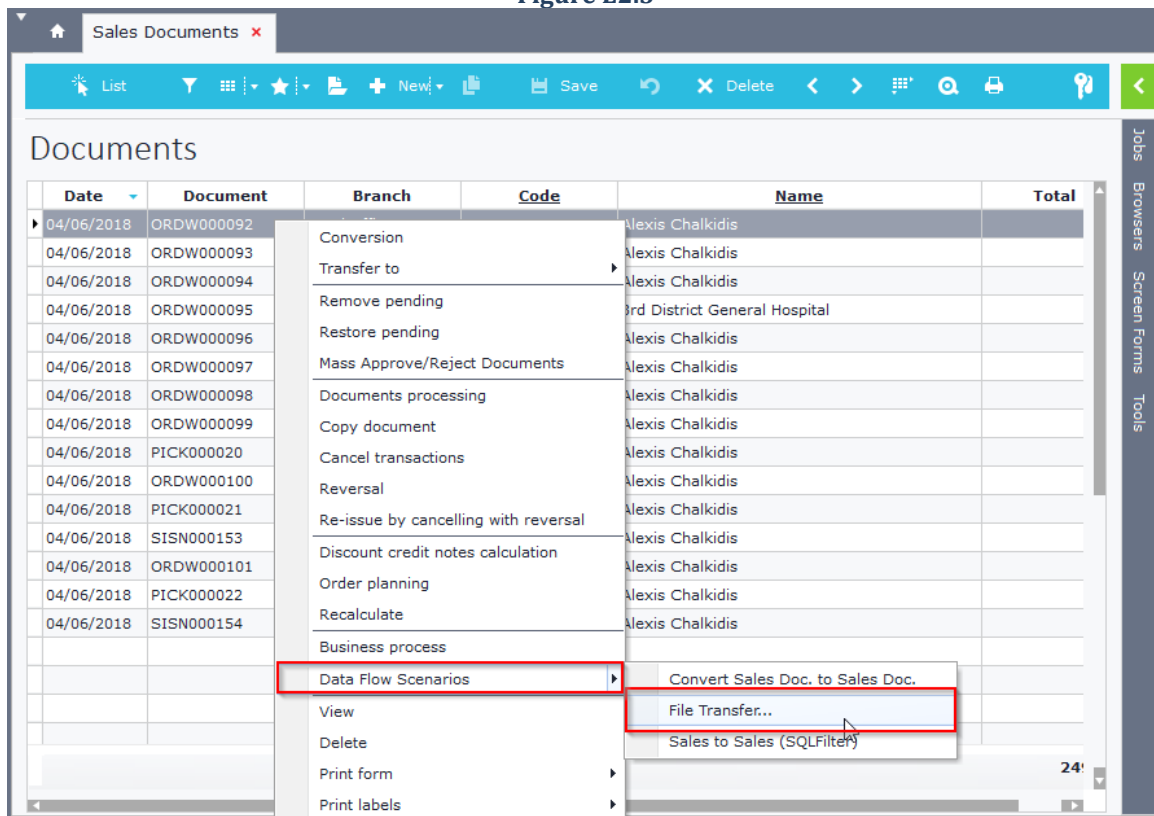


Figure E2.4

E.3 Form Execution

SBSL jobs can be executed through object form screen scripts using the function EXEC with the following syntax.

```
X.EXEC('XCMD: FormImport, ScriptName: NameOfScript')
```

Example (Ex E.2)

Execute the SBSL script named "UpdateItem" using a button in items view for the located item id.

Create a new button (cmd = 20180101) in items view (Figure E3.1).

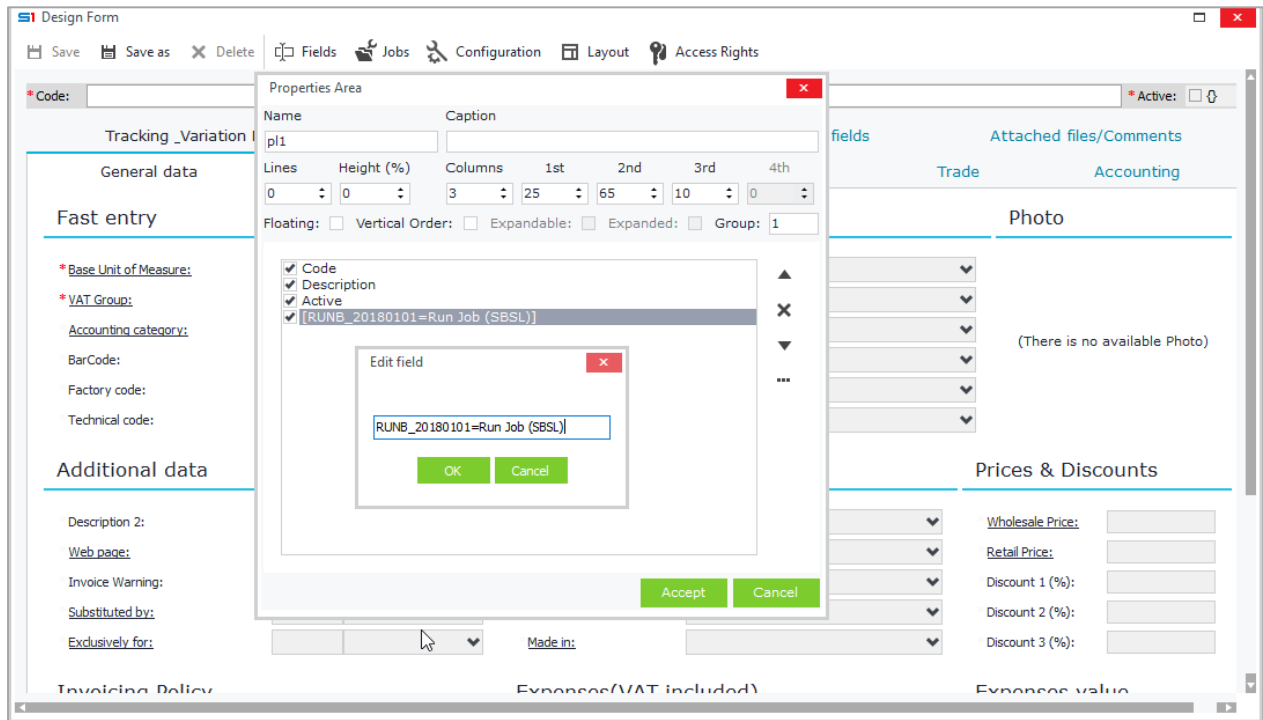


Figure E3.1

Add the following form script in "Configuration" window.

```
function EXECCOMMAND(cmd) {
    if (cmd == 20180101) {
        RunSBSLJob();
    }
}

function RunSBSLJob() {
    X.EXEC('XCMD:FormImport, ScriptName:ItemsUpdate');
}
```

Create the SBSL script "ItemsUpdate" as follows. The id of the located item is retrieved through the dialog parameter **vItemId**, as long as you set its default value to be **:ITEM.MTRL** (Figure E3.2). This field should not necessarily be visible in the job dialog filters.

```
form {
    [TABLES]
    ImpTable=;;;Master;3;0

    [ImpTable]
    vImpOk=2;15;1;1;0;Approve Job;$Y;;;0
    vItemID=3;15;1;1;0;Item ID;ITEM;;;ITEM.MTRL

    [PANELS]
    PANEL1=0;;0;100

    [PANEL1]
    ImpTable.vImpOk
    ImpTable.vItemID
}
```

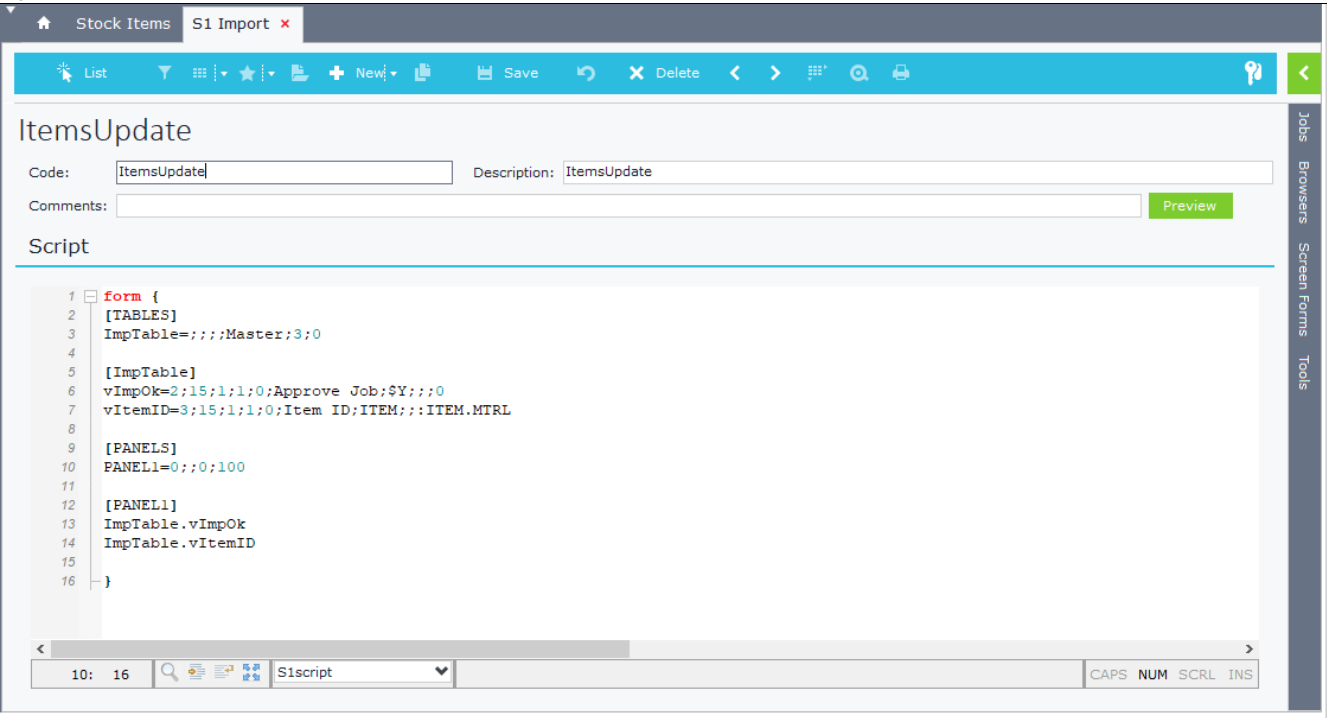


Figure E3.2

When you click on the button "Run Job (SBSL)" inside an item, notice that the filter "vItemID" is populated with the id of the located item (Figure E3.3).

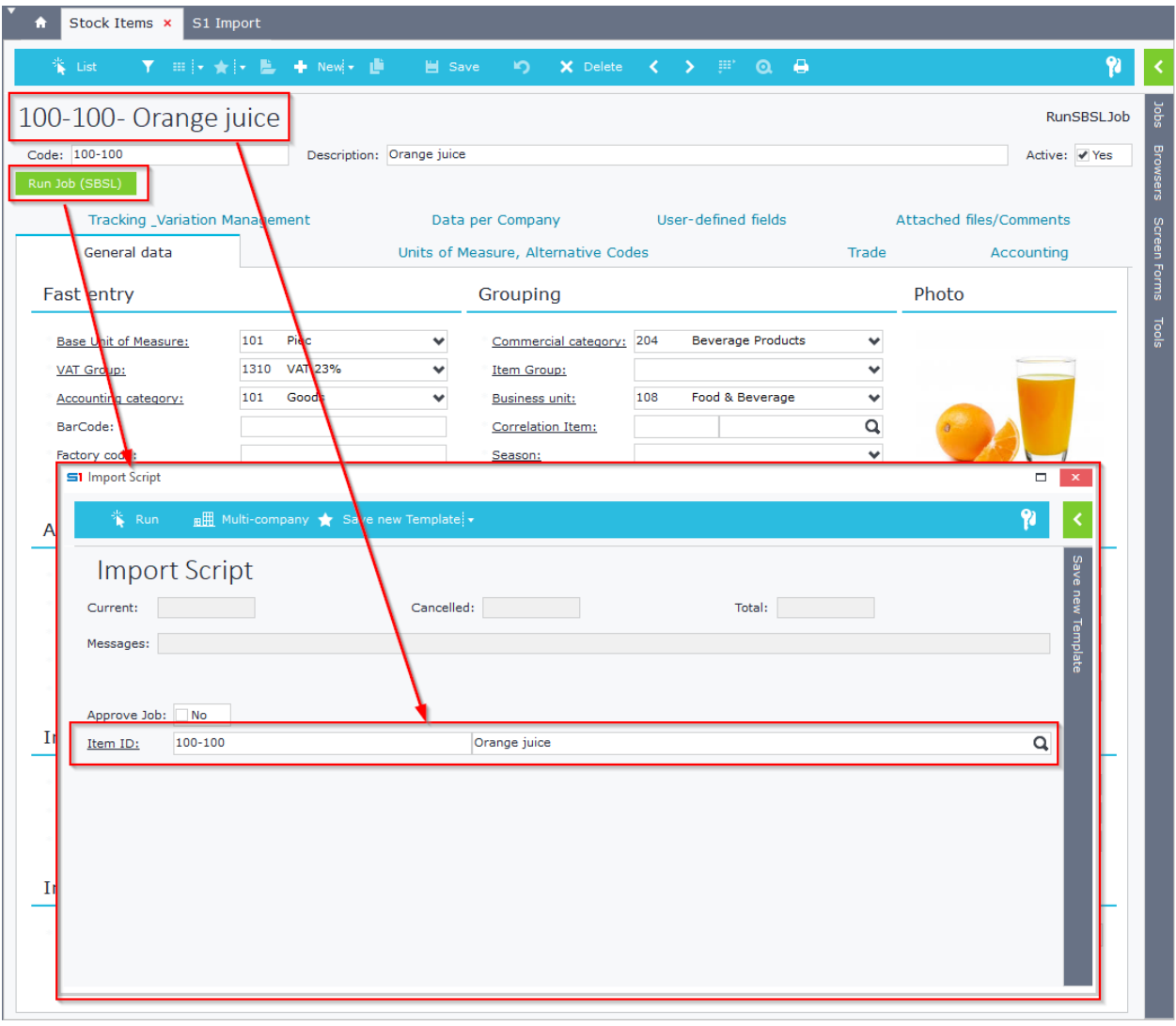


Figure E3.3

E.4 Run using Parameters

SBSL provides the ability to set parameters within the code which can be activated when calling the script. Parameters can be inserted using the ampersand symbol, &, before a variable definition (e.g. **&JOB**).

Script calling with a specific parameter is performed by adding the parameter name and its value at the end of the execution call (e.g. FORMIMPORT, SCRIPTNAME:myscript,**JOB:1**).

Example (Ex E.3)

Run the script of the example “[Ex. C.3](#)” (Section CACHETABLES) for a given SODTYPE.

The name of the script is “cache1”.

The only change that must be done in the script of example Ex C.3 is inside the statement of the dialog parameter vSodtype. You must indicate a value (i.e. &mySodtype) as the default value, which is the parameter that has to be defined when calling the script.

```
Form {
  [TABLES]
    ImpTable=;;;3;0
    TblPAYMENT=;;;TblPAYMENT;3;0
  [ImpTable]
    vSodtype=2;25;1;1;0;Type SODTYPE;$SODTYPE(W[12,13]); &mySodtype
    vPayment=2;25;1;1;0;Payment;TblPAYMENT(W[SODTYPE=:ImpTable.vSodtype AND
COMPANY=:X.SYS.COMPANY]) ;;;

  [TblPAYMENT]
    PAYMENT=2;5;0;1;0;
    NAME=1;30;0;1;0;
    COMPANY=2;5;0;0;0;
    SODTYPE=2;5;0;0;0;

  [CACHETABLES]
    TblPAYMENT=SELECT PAYMENT,NAME,COMPANY,SODTYPE AS SODTYPE FROM PAYMENT

  [PANELS]
    PANEL11=0;;0;50

  [PANEL11]
    ImpTable.vSodtype
    ImpTable.vPayment
}
```

Adding the parameter &mySodtype allows you to call the script through form screens using specific Sodtype (e.g. 13) with the following syntax:

```
X.EXEC ('XCMD:FormImport,ScriptName:cache1,mySodtype:13')
```

You can also execute the SBSL script through a menu job for a specific SODTYPE(e.g. 12) using the syntax:

```
FormImport,ScriptName:cache1,mySodtype:12
```

The screenshot shows a 'Job parameters' dialog box with the following details:

- Job type:** Batch job (selected from a dropdown menu)
- Command/File:** FormImport,ScriptName:cache1,mySodtype:12
- Job title:** Run Job ...
- Buttons:** Available jobs (green), OK (green), Cancel (green)

F. Built-in Functions

This section lists all the internal functions you can use in SBSL scripts.

Abs(Number: numeric): numeric;

Returns the absolute value of the given number.

AddMessage(Text: string);

Adds the given text in the window that is displayed when user clicks the “Results” hyperlink after the execution of a job.

Example

Add a text in Results window that redirects user to a specific sales document record.

```
vObject='SALDOC';
vMess='Document converted <' + sDoc.FINCODE + '$' + vObject + ';' +
VarToStr(sDoc.FINDOC)
+ '> to document <' + vFincode + '$' + vObject + ';' + VarToStr(resultnum) + '>
successfully!' + #10;
x=AddMessage(vMess);
```

CallPublished(FunctionName: string, Params: VarArray);

Executes the function named “FuntionName” using the parameters defined in “Params”.
Use it when you want to call functions from SoftOne Libraries.

Examples

1. Reupdate sales documents (based on a specific query) by changing the value of the field PAYMENT.

See [Case Study 1](#)

2. Import Excel file in a SQL table named “CCCIMPORTEXCEL” (using the function Evaluate from SoftOne Library SysRequest and the function ExcellImport from built-in functions).

```
x = CallPublished('SysRequest.Evaluate', VarArray(ImportModule('Imp'),
'ExcellImport('+#39+:ImpTable.vExcelFile+#39+', '+#39+'CCCIMPORTEXCEL'+#39+',1,2)', 2));
```

CharToOem (var:string);

Converts the given string to OEM.

Example

```
x = CallPublished('PILib.WriteLine', VarArray(vFileWrite, CharToOEM(vLine), 2));
```

Commit(Connectionstring: string);

Ends the transaction by committing the data from the given connection string. Works only on FORMIMPORT calls.

Example

```
x = Commit('SoftOne');
```

Copy(Source: string, Start: integer, Count: integer);

Returns the Source text from “Start” character, for the number of “Count” characters.

DateOfDateTime(DateTime: TDateTime);

Returns the date for a specific “TDATETIME”.

Example

```
TRNDATE = DateOfDateTime(sSalDoc.TRNDATE);
```


DayOfDate(DateTime: TDateTime): integer;

Returns an integer that represents the day of the given date.

DayOfWeek(): integer;

Returns an integer that represents the day of the week. (1=Sunday, 2=Monday, ..., 7=Saturday)

DaysinMonth(DateTime: TDateTime): integer;

Returns the number of days of the month of the given date.

Delay(Time: Integer);

Pauses execution for the given time calculated in msec.

Example

Delay execution for 2 seconds

```
x = Delay(2*1000);
```

DeQuotedStr (text: string);

Returns text without quotation marks.

EncodeTime(Hours: integer, Minutes: integer, sec: integer, ms: integer): TDateTime;

Returns "TDateTime" for specific Hours, minutes, seconds and milliseconds.

ExecPrg(File: string, Params: string);

Runs the file with given parameters

Example

```
vParams = '/c';
x = ExecPrg('c:\windows\system32\cmd.exe', vParams);
```

ExecSQL(Connection: string, Query: string, Params: VarArray);

Executes the SQL "Query" statement based on the "Connection" string.

Example

```
x = ExecSql('SoftOne', 'Update TEMP Set Updated = 1 Where CUSCODE=' +
QuotedStr(sCustomer.CUSTOMERCODE) , Null);
```

Fetch SQLCursorName

Retrieves the rows of a SQL statement defined in "SQLCursorName".

Example

See [Case Study 1](#)

FormatSQLText(Connection: string, text: string, Params: VarArray);

Returns the "text" in SQL format according to the given parameters.

Example

Returns the vDateL date in SQL format

```
FormatSqlText('SoftOne', ':1', :ImpTable.vDateL)
```

GetIndexVar (Array: VarArray, index: integer);

Returns the value of the index element of a zero-based array.

Example

```
if (VarArrayDimCount(fVar) > 0)
    fVar = GetIndexVar(fVar, 0);
```

GetTimeHour(DateTime: TDateTime): integer;

Returns the hour of a specific DateTime.

GetTimeMin(DateTime: TDateTime): integer;

Returns the minutes of a specific DateTime.

GetTimeSec(DateTime: TDateTime): integer;

Returns the seconds of a specific DateTime.

GetTimeMs(DateTime: TDateTime): integer;

Returns the ms of a specific DateTime.

GetQueryDataset(Datasetname: string, Connection: string, Query:string, Params:VarArray);

Populates a Dataset (e.g. datagrid in dialog window) with the data returned from the SQL "Query" statement. Fields of the Dataset must be declared in the same order that they appear in the SQL statement.

Works only in CLIENTIMPORT calls.

Example

```
x=GetQueryDataset('TblSID', 'MyConnection', 'SELECT FIELD1, FIELD2 FROM CCCMYTABLE',
null);
```

GetQueryResults(Connection: string, Query: string, Params: VarArray);

Executes the sql "Query" statement and returns the result set.

Example

```
fVat = GetQueryResults('SoftOne', 'SELECT TOP 1 VAT FROM VAT WHERE ISACTIVE=1',
null);
```

GetWhere(XModule, ParamWhere: string, Param3(0=and,1=where));

Creates the SQL where clause defined in "ParamWhere" string using and or where depending on Param3.

Example 1

```
vFilters = GetWhere(XModule, 'F.TRNDATE=:ImpTable.vDateL;:ImpTable.vDateH'
+#13+#10+'F.SERIES=:ImpTable.vSeries', 0);
```

Example 2

```
vFilters = GetWhere(XModule, 'F.TRNDATE=:ImpTable.vDateL;:ImpTable.vDateH'
+#13+#10+'F.SERIES=:ImpTable.vSeries', 0);
```

GrConvert(text: string): string;

Returns the text converted to Greek characters.

Example

```
vFINCODE = GrConvert(Trim(sSalDoc.FINCODE));
```

ImportModule(Module: string);

Returns the pointer of the import Module.

Example

Import Excel file from the path defined in the variable vExcelFile in SQL table named "CCIMPORTExcel"

```
x=CallPublished('SysRequest.Evaluate', VarArray(ImportModule('Imp'),
'ExcelImport('+#39+:ImpTable.vExcelFile+#39+', '#39+'CCIMPORTExcel'+#39+',1,2)',
2));
//ExcelImport(FileName:string, TableName:string, Param1(1=Excel file with Headers,
2=no Headers), Param2 (1=Keep table records, 2=Delete all table records)
```

Len(text: string): integer;

Returns length of the text.

ModuleCommand (Module, Soft1Command, Params: string);

Executes the "Soft1Command" (form button/job) using the given "Params" parameters.

Example

```
x=CallPublished('ProgLibIntf.ModuleCommand',VarArray(vModule,1032,VartoStr(resultnum),3));
```

MonthOfDate(DateTime: TDateTime): integer;

Returns the month of the given date.

Now(): TDateTime;

Returns the current date and time.

OEMToChar (oem: var);

Converts OEM key code to characters.

QuotedStr (text: string);

Returns text in single quotes.

Example

```
x = GetQueryResults('SoftOne','SELECT COUNT(*) FROM TRDR WHERE
COMPANY='+VartoStr(:X.SYS.COMPANY)+' AND SODTYPE=13 AND
CODE='+QuotedStr(VartoStr(fAdd)+VartoStr(fCounter)), null);
```

Pos(Text: string,Source: string): integer;

Returns an integer specifying the position of the first occurrence of "Text" in "Source" string.

Example

```
x = Pos('c','abcdefg') //Returns 3
```

RaiseException(Message: string);

Terminates the process of the script and displays an exception message on the screen.

Example

Display an exception message if variable vImpOK is not "Yes"

```
if (:ImpTable.vImpOk=0)
x = RaiseException('Run confirmation not selected!!!');
```

RefreshMemoryTable(MemoryTable: string);

Refreshes memory Table and returns the pointer.

Example

```
ds = RefreshMemoryTable('SERIES');
```

Resultnum;

Returns the id of the record that has already been posted.

Example

See [Case Study 2](#)

ReplaceStr (text: string, String1: string, String2: string);

Replaces text in "String1" with the "String2" text.

Example

```
vWhere = ReplaceStr(vWhere, 'SOFT1', 'SOFTONE');
```

RollBack(Connectionstring: string);

Rolls back data defined from connentionstring. Only works on FORMIMPORT calls.

Example

```
x = RollBack('SoftOne');
```

SafeCallPublished(FunctionName: string, Params: VarArray);

Executes the function "FunctionName" from the libraries of the application. If an error is displayed, the exception does not appear on the screen. Moreover, it updates the internal variable of ImportError system.

Example

Import Excel file from the path defined in the variable vExcelFile in sql table named "CCIMPORTEXCEL"

```
x = SafeCallPublished('SysRequest.Evaluate', VarArray(ImportModule('Imp'),
'ExcelImport('+#39+:ImpTable.vExcelFile+#39+', '+#39+'CCIMPORTEXCEL'+#39+',1,2)',
2));
//ExcelImport(FileName:string, TableName:string, Param1(1=Excel file with Headers,
2=no Headers), Param2 (1=Keep table records, 2=Delete all table records))
```

SafeExecSQL(Connection: string, Query: string, Params: VarArray);

Executes the sql "Query" statement. In case of error it does not display the exception and continues the flow of the code.

Example

Deletes data of SID table from the base (whether created or not).

```
x = SafeExecSQL('Export', 'TRUNCATE TABLE SID', Null); //if table does not exist then
no error is displayed
```

SendResponse(Values, Fields: string);

Updates the default interface dialog fields of the batch job.

Example

Update the following interface fields:

Current(CurRec), Canceled(CanRec), Total(TotRec), Messages(LabelText)

```
x = SendResponse(vRow, vRowCancel, vTot, vLabelText,
'RESULTS.CURREC;RESULTS.CANREC;RESULTS.TOTREC;RESULTS.LABELTEXT');
```

ShowWarning(Message: string);

Displays a warning message on the screen.

Example

```
x = ShowWarning('Warning Message');
```

Space(num: integer);

Inserts the defined number of spaces.

Example

```
vLine= VarToStr(Exp.TRNDATE) + Space(1);
```

StartTrans(Connectionstring: string);

Starts a new transaction for the given connection string. Works only on FORMIMPORT calls.

Example

```
x = StartTrans('SoftOne');
```

StrToDate(text: string, format: string): TDateTime;

Converts string to Date format.

Example 1

```
x = StrToDate('20210414', 'dd/mm/yyyy');
```

Example 2

```
[ImpTable]
vSalSeries=1;512;0;1;0;Series;#SERIES(W[SOSOURCE=1351]);;
vDateL=11;20;0;1;0;Date;$DATERANGE;;131
vDateH=11;20;0;1;0;

Var vDateFrom, vDateTo, vSeries;

{
vSeries = ModuleIntf.GetFieldValue(vImpTable, 'vSalSeries');
vDateFrom = ModuleIntf.GetFieldValue(vImpTable, 'vDateL');
vDateTo = ModuleIntf.GetFieldValue(vImpTable, 'vDateH');
vSQLWhere = 'AND F.SOSOURCE=1351'
+GetWhere(XModule, 'F.TRNDATE='+StrToDate(vDateL, '') + ';' + VarToStr(vDateH), 0)
+GetWhere(XModule, 'F.SERIES IN ('+vSeries+')', 0);
}
```

Example 3

```
vSQL = FormatSqlText('Xplorer',';1',StrToDate(vCol2,'dd/mm/yyyy'));
```

StrToFloat(var: string);

Converts float to string.

Example

```
Module = CallPublished('ModuleIntf.CreateModule','SALDOC,WARNINGS:OFF,NOMESSAGES:1');
DsMTRLINES = CallPublished('ModuleIntf.GetDataSet',VarArray(Module,'MTRLINES',2));
LinesQTY = StrToFloat(CallPublished('ModuleIntf.GetFieldValue', VarArray(DsMtrlines,
'QTY1', 2)));
x = CallPublished('ModuleIntf.SetFieldValue',VarArray(DsMtrlines,'QTY',LinesQTY,3));
```

StrToInt (var: string);

Converts string to integer.

TableExists(Connection: string, TableName: string);

Checks if a table named "TableName" exists and returns True or False.

Example

Check whether SID table exists and creation thereof in case it doesn't

```
if (TableExists('Export', 'SID') = 'False')  
    x = ExecSQL('Export', 'CREATE TABLE SID (FR INT, FV INT)', Null);
```

Trim(text: string): string;

Removes spaces and returns the text.

Time():TDateTime;

Returns the current time.

VarArray(Var1, Var2, ..., Varn, n);

Declares an array of variants, where the last element is the number of variants.

VarArrayDimCount (Array: VarArray);

Returns the number of dimensions of the Array.

Example

```
if (VarArrayDimCount(fVar)>0)  
    fVar=GetIndexVar(fVar,0);
```

VarArrayHighBound(Array: VarArray, Dim: integer);

Returns the number of elements in the Array for dimension Dim.

Example

```
vCaptions = VarArray(VarArray('Code', 'Description', 'Category',3), VarArray('Code',  
'Name', 'Category', 3), 2);  
vCaptions = GetIndexVar(vCaptions,1); //Returns the 1st VarArray  
vSQLCaptions = VarArray('cCODE', 'cNAME', 'cMTRCATEGORY', 3)  
while (i <= VarArrayHighBound(vCaptions,1)) { //Compares the value I with the number  
of the elements of vCaptions(3 elements)  
    vLang=vLang+', '+QuotedStr(GetIndexVar(vCaptions,i))+ ' AS  
'+GetIndexVar(vSQLCaptions,i);  
    i=i+1;  
}
```

VarToStr(Param: Variant);

Converts Param to string.

Example

```
vRow = 100;  
vMess = vMess + 'Found ' + VarToStr(vRow) + ' files...'
```

XModule();

Returns the pointer of a Module.

Example

```
Form {
[TABLES]
    ImpTable=;;;Master;3;0
    TblDates=;;;Master;3;0

[TblDates]
    vDateFrom=11;25;1;1;0;Date From;;;
    vDateTo=11;25;1;1;0;Date To;;;
    vWRKHOURLS = 3;15;1;1;0;Work Hours;WRKHOURLS;;;

[ImpTable]
    vMess=16;64000;0;1;1;Job Messages...;;;
    vWhere=16;80000;0;1;0;Browser Selection;;;&SELRECS;&SELRECS

[PANELS]
    PANEL1=2;Shift selection;0;H40;
    PANEL13=4;Job Messages...;0;100,G10,N,L3
[PANEL1]
    TblDates.vDateFrom
    TblDates.vDateTo
    TblDates.vWRKHOURLS

[PANEL13]
    ImpTable.vMess
}
var x, vTblDates;
{
    vTblDates = GetDataSet(XModule, 'TblDates');
    x = DataSetFirst(vTblDates);
    //x = DataSetDisableControls(vTblDates);
    while (DataSetEof(vTblDates) = 0) {
        vTblDateFrom =
FormatDateTime('YYYYMMDD',GetFieldValue(vTblDates,'vDateFrom'));
        vTblDateTo = FormatDateTime('YYYYMMDD',GetFieldValue(vTblDates,'vDateTo'));
        vTblWRKHOURLS = GetFieldValue(vTblDates,'vWRKHOURLS');
        //Create Action
        vModule = CreateModule('SOPRSN,WARNINGS:OFF,NOMESSAGES:1');
        x = InsertModule(vModule);
        vTask = GetDataSet(vModule,'SOACTION');
        x = DataSetEdit(vTask);

        x = SetFieldValue(vTask,'SERIES',3001);
        //...
        x = DataSetNext(vTblDates);
    }
}
```

XSupport();

Returns the pointer to the Connection.

YearOfDate(DateTime: TDateTime): integer;

Returns the year of the given date.

Example

```
fTmp = :X.SYS.SYSDATE;
fYear = VarToStr(YearOfDate(fTmp));
```

G. SoftOne Libraries

In this module you will find an analysis of the SoftOne Libraries, including all methods and functions you can call from SBSL script. Commands are called either by using the Callpublished function, or directly after prior definition of the specific library using include command.

G.1 ModuleIntf

CreateSupport(XCOfile, UserName, Password, Company, Branch, LoginDate);

Creates a new connection to a SoftOne database defined from the "XCOfile". If the first three parameters (XCOfile, UserName, Password) are omitted then the parameters from the login connection are used.

Example

```
include 'ModuleIntf';
fCompany = :X.SYS.COMPANY;
fDefaultBranch = CallPublished('ProgLibIntf.GetDefaultBranch', VarArray(:X.SYS.USER,
fCompany, 2));
fModule = CreateSupport ('', '', '', fCompany, fDefaultBranch, :X.SYS.LOGINDATE);
```

DestroySupport(SupportHandle);

Frees the connection created to Soft1 from "CreateSupport" function.

Example

```
x = DestroySupport (fSupport);
```

CreateSupportModule(SupportHandle, ModuleName);

Creates an instance of a module (SoftOne object) for the Connection specified from "CreateSupport" function.

Example

```
vSupport = CreateSupport ('', '', '', 1000, 1000, :X.SYS.LOGINDATE);
fModuleName = 'SALDOC';
fModule = CreateSupportModule (vSupport, fModuleName);
```

GetTextValue(TextName);

Reads the Resources from the pre files (e.g. SoftOne script) of the application.

Example

Return data of the SBSL script 'CheckImportData'.

```
vData = GetTextValue ('CheckImportData');
```

OpenSubForm(ModuleHandle, SubFormName, OpenMode);

Opens or closes the sub form called SubFormName. The form always opens in modal mode.

OpenMode accepts the following values:

- 1: Opens the sub form and fires the event "Before show form"
- 1: Closes the sub form and fires the "Accept" event
- 2: Closes the sub form and fires the "Cancel" event

CreateModule(ModuleName);

Creates an instance of module (SoftOne object) only for the current login connection. In order to use a specific form, you have to use the SysRequest function CreateForm and then GetFormModule.

Example

```
vModule = CreateModule ('CUSTOMER, WARNINGS:OFF, NOMESSAGES:1');
```


DestroyModule(ModuleHandle);

Destroys the Module created using "CreateModule" or "CreateSupportModule" functions.

Example

```
x = DestroyModule(fModule);
```

FindXStrings(Module, StringListName: string);

Gets the string list of the defined module.

Example

```
var strlist;
strlist = FindXStrings(XModule, 'RELJOBS');
strlist = FindXStrings(XModule, 'BRMENU');
```

InsertModule(ModuleHandle);

Changes the status of a module (SoftOne object) to insert mode in order to accept data.

Example

Insert a new task (Series: 1003, Comments: 'Inserted from SBSL script')

This specific example does not 'include' the library ModuleIntf but alternatively it uses the function CallPublished to execute the functions of the library, though it's preferred to include libraries for better performance.

```
var x, fModule, vTask;

fModule =
CallPublished('ModuleIntf.CreateModule', 'SOTASK, WARNINGS:OFF, NOMESSAGES:1');
x = CallPublished('ModuleIntf.InsertModule', fModule);

vTask = CallPublished('ModuleIntf.GetDataset', VarArray(fModule, 'SOACTION', 2));
x = CallPublished('ModuleIntf.DatasetEdit', vTask);

x = CallPublished('ModuleIntf.SetFieldValue', VarArray(vTask, 'SERIES', 1003, 3));
x = CallPublished('ModuleIntf.SetFieldValue', VarArray(vTask, 'COMMENTS', 'Inserted from
SBSL script', 3));

x = CallPublished('ModuleIntf.DatasetPost', vTask);
x = CallPublished('ModuleIntf.PostModule', fModule);
x = CallPublished('ModuleIntf.DestroyModule', fModule);
```

LocateModule(ModuleHandle, RecordID);

Locates a record of a module (ModuleHandle) based on the RecordID.

Example

```
fModule = CreateModule('CUSTOMER, WARNINGS:OFF');
vCustomerID = 11999;
x=LocateModule(fModule, vCustomerID);
```

DeleteModule(ModuleHandle);

Deletes a record of a module (ModuleHandle) that is previously located.

Example

Delete the customer with id = 11999.

```
include 'ModuleIntf';
var x, fModule, vCustomerID;
fModule = CreateModule('CUSTOMER, WARNINGS:OFF');
vCustomerID = 11999;
x=LocateModule(fModule, vCustomerID);
x = DeleteModule(fModule);
```

PostModule(ModuleHandle);

Posts in the database the data of a module (SoftOne object), applying all SoftOne methods.

Example

Insert new customer with code '1000' and name 'TempCustomer'

```
include 'ModuleIntf';
var x, fModule, vTableTRDR;
fModule = CreateModule('CUSTOMER,WARNINGS:OFF');
x = InsertModule(fModule);
vTableTRDR = GetDataset(fModule,'TRDR');
x = DatasetEdit(vTableTRDR);
x = SetFieldValue(vTableTRDR, 'CODE','1000'); //Sets code 1000 to field TRDR.CODE
x = SetFieldValue(vTableTRDR, 'NAME','TempCustomer'); //Sets name 'TempCustomer' to
field TRDR.NAME
x = DatasetPost(vTableTRDR);
x = PostModule(fModule);
x = DestroyModule(fModule);
```

GetParentModule(ModuleHandle);

Returns the ParentModule of the current Module

GetChildModule(ModuleHandle, Name);

Returns the ChildModule of the current Module

GetDataset(ModuleHandle, Name);

Returns the Dataset of the Module

Example

Insert new supplier with code '2000' and name 'TempSupplier'

```
include 'ModuleIntf';
var x, fModule, vTableTRDR;
fModule = CreateModule('CUSTOMER,WARNINGS:OFF');
x = InsertModule(fModule);
vTableTRDR = GetDataset(fModule,'TRDR');
x = DatasetEdit(vTableTRDR);
x = SetFieldValue(vTableTRDR, 'CODE','2000'); //Sets code 2000 to field TRDR.CODE
x = SetFieldValue(vTableTRDR, 'NAME','TempSupplier'); //Sets name 'TempSupplier' to
field TRDR.NAME
x = DatasetPost(vTableTRDR);
x = PostModule(fModule);
x = DestroyModule(fModule);
```

GetFieldProperty(FieldHandle, PropertyIndex);

Returns the field's property value for the given property index.

PropertyIndex accepts the following values:

- 1: Field Name
- 2: Data Type
- 3: Size
- 4: Display Name
- 5: Required
- 6: Read-only

Example

```
vDs = ModuleIntf.GetDataSet(XModule,'CUSTOMER');
vFld1 = ModuleIntf.GetField(vDs,'NAME');
vProp1 = GetFieldProperty(vFld1,5); //returns 1 when field is mandatory
```

SetFieldProperty(FieldHandle, PropertyIndex, PropertyValue);

Modifies field property value for the given property index.

PropertyIndex accepts the following values:

5: Required (1=true, 0=false)

6: Read-only (1=true, 0=false)

Example – JavaScript

In conversion dialog object change the “required” property of the field “Delivery Date” (DELIVDATE) to mandatory (using Advanced JS).

```
function ON_CREATE() {
  var ds = X.EXEC('CODE:ModuleIntf.GetDataSet', X.MODULE, 'CONVERTFPRMS');
  var fld = X.EXEC('CODE:ModuleIntf.GetField', ds, 'DELIVDATE');
  X.EXEC('CODE:ModuleIntf.SetFieldProperty', fld, 5, 1);
}
```

GetParamValue(ModuleHandle, PrmName);

Returns the value of PrmName

Example

```
vFile = GetParamValue(XModule, 'MyParameterName');
```

SetParamValue(ModuleHandle, PrmName, PrmValue);

Sets a new value in the parameter “PrmName” of the Module.

Example

```
fDataset = GetDataset(vModule, 'SERIES');
x = SetParamValue(fDataset, 'MyParameter', 10); //Creates a new parameter
(MyParameter) and sets its value to 10
```

CreateDataset(ModuleHandle, DatasetName);

Creates a Dataset named “DatasetName”

DatasetLocate(DatasetHandle, FieldNames, FieldValues);

Returns True if a record is found in Dataset and then locates it.

Example 1

```
vFISCPRD = RefreshMemoryTable('FISCPRD');
x = DatasetLocate(vFISCPRD, 'FISCPRD', GetFieldValue(vQUESTIONS, 'FISCPRD'));
```

Example 2

```
x = CallPublished('ModuleIntf.DataSetLocate', VarArray(vItelines, 'MTRL;MTRLINES',
VarArray(vItem.MTRL, vItem.MTRLINES, 2), 3));
```

Example 3

```
x = CallPublished('ModuleIntf.DataSetLocate', VarArray(vSpcLines, 'MTRL;SPCLINES',
VarArray(vLines.MTRL, vLines.SPCLINES, 2), 3));
```

Example 4

```
x = CallPublished('ModuleIntf.DataSetLocate', VarArray(vCDIM, 'CDIMLINES1;CDIMLINES2',
VarArray(vCdimLines1, vCdimLines2, 2), 3));
```

DatasetEdit(DatasetHandle);

Puts the Dataset into Edit state.

Example

Locate a record in Sales document (id=1000) and add comments 'Comments from SBSL script'. This specific example uses the CallPublished function instead of include 'ModuleIntf'

```
var x, fModule, dsFinDoc, vfindocID;

fModule =
CallPublished('ModuleIntf.CreateModule','SALDOC,WARNINGS:OFF,NOMESSAGES:1');
dsFinDoc = CallPublished('ModuleIntf.GetDataset',VarArray(Module,'FINDOC',2));
vfindocID = 1000;
x = CallPublished('ModuleIntf.LocateModule',VarArray(Module, vfindocID,2));
x = CallPublished('ModuleIntf.DatasetEdit',DsFinDoc);

x = CallPublished('ModuleIntf.SetFieldValue',VarArray(DsFinDoc, 'COMMENTS', 'Comments
from SBSL script', 3));
x = CallPublished('ModuleIntf.PostModule', Module );
```

DatasetInsert(DatasetHandle);

Opens a new, empty record before the current record, and makes the empty record the current one so that field values for the record can be entered by code.

Example

```
include 'ModuleIntf';
var x, fModule, fSalDoc, fIteLines;

fModule=CreateModule('SALDOC,WARNINGS:OFF');
x = InsertModule(fModule);
fSalDoc = GetDataset(fModule,'SALDOC');
fIteLines = GetDataset(fModule,'ITELINES');
x = DatasetInsert (fIteLines);
```

DatasetAppend(DatasetHandle);

Opens a new, empty record at the end of the Dataset and makes the empty record the current one so that field values for the record can be entered by code.

Example

Insert new substitute item(code 101) in table MTRSUBSTITUTE for the located item with id=1000.

```
include 'ModuleIntf';
var x, fModule, fDsMtrsubstitute, vMTRLID;

fModule = CreateModule('ITEM,WARNINGS:OFF,NOMESSAGES:1');
vMTRLID = 1000;
x = LocateModule(fModule, vMTRLID);
fDsMtrsubstitute = GetDataset(fModule,'MTRSUBSTITUTE');
x = DatasetAppend (fDsMtrsubstitute);

x = SetFieldValue(fDsMtrsubstitute,'CODE','101');
x = SetFieldValue(fDsMtrsubstitute,'NAME','SubItem101');
x = SetFieldValue(fDsMtrsubstitute,'QTY1',1);

x = DatasetPost(fDsMtrsubstitute);
x = PostModule(fModule);
x = DestroyModule(fModule);
```

DatasetEof(DatasetHandle);

Returns True if the Dataset is on the last record

Example

```
ds = GetSQLDataset(0, 'SELECT FINDOC FROM FINDOC WHERE... ', null)
while (DatasetEof(Ds)=0)
{
    ...
    x = DatasetNext(Ds);
}
```

DatasetPrior(DatasetHandle);

Moves to the previous record of the Dataset

Example

```
vMyTable = GetDataset(XModule, 'CCCMYTABLE');
x = DatasetPrior(vMyTable);
```

DatasetFirst(DatasetHandle);

Moves to the first record of the Dataset

Example

```
x = DatasetFirst(vMyTable);
```

DatasetLast(DatasetHandle);

Moves to the last record of the Dataset

Example

```
x = DatasetLast(vMyTable);
```

DatasetNext(DatasetHandle);

Moves to the next record of the Dataset

Example

```
x = DatasetNext(vMyTable);
```

DatasetPost(DatasetHandle);

Posts the current record to the Dataset

Example*Insert new project with code '10' and name 'Project from SBSL script'*

```
include 'ModuleIntf';
var x, fModule, vTablePRJC;
fModule = CreateModule('PRJC, WARNINGS:OFF'); //SoftOne object is PRJC
x = InsertModule(fModule);
vTablePRJC = GetDataset(fModule, 'PRJC'); //Table is PRJC
x = DatasetEdit(vTablePRJC);
x = SetFieldValue(vTablePRJC, 'CODE', '10'); //Sets code 10 to field PRJC.CODE
x = SetFieldValue(vTablePRJC, 'NAME', 'Project from SBSL script');
x = DatasetPost(vTablePRJC);
x = PostModule(fModule);
x = DestroyModule(fModule);
```

DatasetDelete(DatasetHandle);

Deletes the current record of the Dataset

Example

```
x = DatasetDelete(vMyTable);
```

DatasetState(DatasetHandle);

Returns the State of the Dataset. The available values are summarized below:

0=dsInactive

Dataset closed. Data unavailable.

1=dsBrowse

Dataset open. Data can be viewed, but not changed. This is the default state of an open Dataset.

2=dsEdit

Dataset open. The current row can be modified.

3=dsInsert

Dataset open. A new row is inserted or appended.

6=dsFilter

Dataset open. Indicates that a filter operation is under way. A restricted set of data can be viewed, and no data can be changed.

DatasetSetFilter(DatasetHandle, Filterstring);

Applies "Filterstring" filter to Dataset

SetDatasetLinks(ModuleHandle, DatasetHandle, Value);

When you populate a Dataset that has fields linked to tables, then for every record added, a query runs for each of these fields in order to link them to the tables. This function provides the ability to send the queries once when all records are filled.

Value: 0= Disable links, 1= Enable links

Example

```
fModule = CreateModule('INST,WARNINGS:OFF');
x = InsertModule(fModule);
fDsInst = GetDataset(fModule,'INST');
x = SetDatasetLinks(fModule, fDsInst, 0);
```

GetFieldType(DatasetHandle, FieldName);

Returns a number that applies to the type of the field. Values returned are:

1=String, **2**=Smallint, **3**=Integer, **4**=Word, **6**=Float, **11**=DateTime, **16**=Memo

Example

```
fField = DatasetFieldName(fDs, fFieldsCounter);
fFieldType = GetFieldType(fDs, fField);
```

GetFieldValue(DatasetHandle, FieldName);

Returns the field value for the located record of a dataset

SetFieldValue(DatasetHandle, FieldName, FieldValue);

Assigns a value in the field for the located record of a dataset

SetValueFromFile(DatasetHandle, FieldName, FileName);

Saves data in a blob (image) field of a dataset.

Example

Locate an item with id=100 and insert an image from file (C:\myimage.png)

```
include 'ModuleIntf';
var x, fModule, vIteDocData, vMTRLID;

fModule = CreateModule('ITEM');
vMTRLID = 100;
x = LocateModule(fModule, vMTRLID);
vIteDocData = GetDataSet(Module, 'ITEDOCDATA');
x = DataSetEdit(vIteDocData);
x = SetValueFromFile(vIteDocData, 'SODATA', 'C:\myimage.png');
x = DataSetPost(vIteDocData);
x = PostModule(fModule);
x = DestroyModule(fModule);
```

DatasetRecordCount(DatasetHandle);

Returns the number of records in the Dataset

DatasetOpen(DatasetHandle);

Sets Dataset in Open state

DatasetClose(DatasetHandle);

Closes the Dataset

DatasetRecNo(DatasetHandle);

Returns the location of arecord in the Dataset

DatasetSetRecNo(DatasetHandle, RecNo);

Setsthe location of a record in the Dataset

DatasetDisableControls(DatasetHandle);

Disables Windows Controls to theDataset

DatasetEnableControls(DatasetHandle);

Enables Windows Controls to the Dataset

DatasetFieldCount(DatasetHandle);

Returns the number of fields of a Dataset

DatasetFieldName(DatasetHandle, FieldIndex);

Returns the field name of a Dataset based on the index of the field

GetSQLDataset(SupportHandle, SQLStatement, Params);

Returns aDatasetbased on the SQL statement.

Example

```
vds = GetSQLDataset(0, 'SELECT DISTINCT CODE, NAME FROM CCCMYTABLE ORDER BY CODE, NAME', null);
```

GetBand(ModuleHandle, BandName);

Return the Module Band

SetBandXProperty(BandHandle, FieldName, PropertyType, PropertyValue);

Sets the column properties of the Band

Property Types: **1**=Caption, **2**=Width. Decimals, **4**=Visible, **7**=HasSums, **8**=DebitCredit, **9**=Transfer

GetBandXProperty(BandHandle, FieldName, PropIndex);

Returns the properties of a column of the Band

AddBandField(BandHandle, FieldName);

Adds a new column in the Band

DeleteBandField(BandHandle, FieldName);

Deletes a Band column

GetField(DatasetHandle, FieldName);

Returns the Dataset field

AddFieldToDataset(DatasetHandle, FieldName, Type, Size, Label);

Adds a field to a Dataset

SetFieldEditor(ModuleHandle, FieldName, Editor, ClearOld);

Changes the Editor of a field

ResolveEditors(ModuleHandle, TableName);

Follows SetFieldEditor and activates new Editors on the table

SetProperty(ModuleHandle, ControlType, ControlName, Property, Value);

Sets the property defined to a Module with parameters 1,2,3 (e.g. 'PANEL','PANELNAME','VISIBLE','TRUE')

DebugString(text:string) published 'ModuleIntf.DebugString';

Text message is saved in the EventLog.

G.2 PiLib

AsciiImport(ParamsInfo, FieldsInfo);

Reads an ascii file with a specific layout and imports it in a table that is already created in the database.

ParamInfo = Parameters (Table name, ASCII file path, Decimals separator, first line to insert, Convert chars to 437, Date format)

FieldsInfo = Fields layout

Example

```
include 'PiLib';
vParamsInfo = 'TABLENAME=test_table'+##13+##10+
              'ASCIIINAME=C:\myDir\myfile.txt'+##13+##10+
              'THOUSANDSEPARATOR=.'+##13+##10+
              'DECIMALSEPARATOR=,'+##13+##10+
              'FIRSTLINE=1'+##13+##10+
              'OEMCONVERT=1'+##13+##10+
              'DATEFORMAT=';
vFieldsInfo = 'TRNDATE=1,8'+##13+##10+
              'CODE=010,012'+##13+##10+
              'NAME=023,039'+##13+##10+
              'TPRMS=063,003'+##13+##10+
              'FINCODE=067,009'+##13+##10+
              'WHOUSE=079,001'+##13+##10+
              'TRDCODE=082,005'+##13+##10+
              'QTY=098,012'+##13+##10+
              'TRNVALUE1=137,012'+##13+##10+
              'COMMENTS=149,039';
x = AsciiImport(vParamsInfo, vFieldsInfo);
```

CreateDir (Directory: string);

Creates the defined directory.

Example

```
x = CallPublished('PiLib.CreateDir', Directory);
```

DirectoryExists (Directory: string);

Returns true if the directory exists.

Example

```
x = CallPublished('PiLib.DirectoryExists', Directory);
```

ForceDirectories(Path: string);

Creates all the folders under a specific path, if they don't exist.

Example

```
x = CallPublished('PiLib.ForceDirectories', Path);
```

RemoveDir(Directory: string);

Deletes the folder, if it's empty.

Example

```
x = CallPublished('PiLib.RemoveDir', Directory);
```

FileExists(FileName: string);

Checks whether the file exists.

Example

```
include 'PiLib';
if (FileExists(FileName)=='True')
{
    //...
}
```

DeleteFile(FileName: string);

Deletes file FileName.

Example

```
vFileDel = DeleteFile(FileName) ;
```

RenameFile(OldName: string, NewName: string);

Renames the file "OldName" file to "NewName".

CreateText(FileName: string);

Creates the "FileName" file.

Example

```
vFileName = '.\folder1 '+'Myname.log';
vFile1 = CreateText(vFileName) ;
```

CreateTextCodePage(FileName: string, CodePage: integer);

Creates file "FileName" file using specific encoding.

Example

```
vFile2 = CreateTextCodePage(:ImpTable.vFile, 65001) ; //UTF-8 Unicode
```

OpenText(FileName:string);

Opens "FileName" file for Read or Write.

Example

```
vFile1 = OpenText(:ImpTable.vFile) ;
```

OpenTextCodePage(FileName: string, CodePage: integer);

Opens "FileName" file for Read or Write using the codepage defined.

Codepage identifiers can be found at the following links:

<https://docs.microsoft.com/en-us/windows/win32/intl/code-page-identifiers>

https://en.wikipedia.org/wiki/Windows_code_page

Example

```
vFile2 = OpenTextCodePage(:ImpTable.vFile, 65001) ; //UTF-8 Unicode
```

ReadLine(FileName: string);

Reads a line characters from the file "FileName".

Example

```
while (Eof(vFile)=0)
{
    vLine = ReadLine(vFile) ;
}
```

WriteLine(FileName: string, Text: string);

Writes the specified data, followed by the current line terminator, to the "FileName" file.

Example

```
x = WriteLine(vFile, 'Start job');
```

EOF(FileName: string);

Returns true if the file "FileName" is at the end.

Example

```
while (Eof(vFile)=0)
{
    vLine = ReadLine(vFile);
}
```

CloseText(FileName: string);

Closes file FileName, which was opened using "OpenText" or "OpenTextCodePage" functions.

Example

```
x = CloseText(vFile);
```

WriteTextToFile(FileName: string, Text: string);

Writes the specified data to the file "FileName", replacing any other text inside the file.

Example

```
x = WriteTextToFile(vPath+'PL-'+VarToStr(sDoc.FINDOC)+' .TXT', vText);
```

ReadTextFromFile(FileName: string);

Opens file FileName and reads all its contents.

Example

```
x = ReadTextFromFile(vFile);
```

ReadFile(FileName: string);

Opens file FileName and reads it in Stream.

Example

```
x = ReadFile(vFile);
```

AnsiToUTF8(Text: string);

Converts the text "Text" from ANSI to UTF8.

Example

```
x = AnsiToUtf8(vLine);
```

UTF8ToAnsi (Text: string);

Converts the text "Text" from UTF8 to ANSI.

Example

```
x = Utf8ToAnsi(vLine);
```

GetSupportQueryResults(Support: Handle, Query: string, Params: VarArray);

Returns the value for the Query of the Handle stated. Returns a value or an Array.

GetSupportQueryData(Support: Handle, Query: string, Params: VarArray);

Returns the result of the Query of the Handle stated in compressed format.
Fields are separated with '|' and entries with '|~'

FindFirst(Path: string, Attr: integer);

Searches a folder for the first file that use specific attributes (and name).
Returns an array with the following: [0]=Filename, [1]=Date, [2]=Size, [3]=Attributes
Don't forget to use FindClose after the use of this function in order to deallocate memory.

File Attributes

faReadOnly=1, faHidden=2, faSysFile=4, faVolumeID=8, faDirectory=16, faArchive=32, faAnyFile=63

Example

```
vFind = FindFirst ('C:\test.*', 63);
```

FindNext(FileHandle: FindFirstResult);

Searches for the next file with the specific name and attributes.
Don't forget to use FileClose after the use of this function in order to deallocate memory.

Example

```
x = FindNext (vFind) ;
```

FindClose (FileHandle: FindFirstResult);

Frees the memory after using FindFirst or FindNext functions.

Example

```
x = FindClose (vFind) ;
```

GetXStrings(StringListName: string);

Returns the data of the "Name" string list.

Example

See [Case Study 4](#)

TStringsCreate();

Defines a string list object and points the variable at it.

TStringsClear(TStringsHandle);

Clears the data of the string list.

TStringsGetValue(TStringsHandle, Key: string);

If there is a name-value pair in the list of strings that has name part "Key", then the corresponding value is returned.
If there is no such pair, an empty string is returned.

Example

```
var strlist, vDocLinksValue;  
strlist = FindXStrings(XModule, 'RELJOBS');  
vDocLinksValue = VarToStr(TStringsGetValue(strlist, '@FINDOC.XDOCLINKS')); //Returns  
the name of the relative job
```

TStringsValueOfIndex (TStringsHandle, Index: integer);

Moves to the specified index of the string list and returns its value.

TStringsSetValue(TStringsHandle, Key: string, Value: Variant);

Checks whether there exists a name-value pair in the list with name "Key". If such a pair is found, its value part is overwritten with the specified value. If no such pair is found, a new name-value pair is added with the specified "Key" and value.

TStringsGetItem(TStringsHandle, Index: integer);

Moves to the specified index of the string list and returns the item.

Example

See [Case Study 4](#)

TStringsGetItemName(TStringsHandle, Index: integer);

Moves to the specified index of the string list and returns its key.

TStringsSetItem(TStringsHandle, Index: integer, Value: Variant);

Moves to the specified index of the string list and sets the value "Value".

TStringsAdd(TStringsHandle, DataStr: string);

DataStr Structure: Key = Value

Adds "DataStr" data to the next record of the string list.

Example

```
mystrlist = GetXStrings('CCCMYLIST');  
x = TStringsAdd (mystrlist, '20=New Item In List');
```

TStringsCount(TStringsHandle);

Returns the number of records of the string list.

Example 1

```
mystrlist = GetXStrings('CCCMYLIST');  
x = TStringsCount (mystrlist);
```

Example 2

See [Case Study 4](#)

G.3 SysRequest

CreateForm('ObjectName[ObjectAttributes]');

Creates a pointer for a SoftOne object, allowing also to use a specific form. The module instance is created through the function GetFormModule.

Syntax:

```
vForm = CallPublished('SysRequest.CreateForm', VarArray('SOBJECT[FORM:MyForm]', 5, 2));
vFormModule = CallPublished('SysRequest.GetFormModule', vForm);
```

Example

```
vForm = CallPublished('SysRequest.CreateForm', VarArray('SOACTION[FORM:MyForm]', 5, 2));
vFormModule = CallPublished('SysRequest.GetFormModule', vForm);
fetch sDoc {
    x = InsertModule(vFormModule);
    DsSOACTION = GetDataSet(vFormModule, 'SOACTION');
    DsCCCMYTable = GetDataSet(vFormModule, 'CCCMYTABLE');
    x = DataSetEdit(DsSOACTION);
    x = SetFieldValue(DsSOACTION, 'SERIES', VarToStr(: ImpTable.vSeries));
    ...
    fetch sDocLines {
        x = DatasetAppend(DsCCCMYTable);
        x = SetFieldValue(DsCCCMYTable, 'CHEQUE', sDocLines.CHEQUE);
        ...
        x = DataSetPost(DsCCCMYTable);
    }
    newid = PostModule(vFormModule);
    if ((ImportError = 0) AND (newid > 0)) {
        ....
    }
}
x = CallPublished('SysRequest.DestroyObject', vFormModule); //Destroys pointer
```

DestroyObject(ObjectHandle);

Frees memory of the defined object.

Example

```
x = DestroyObject(fObject);
```

DoSendMail2(To, CC, BCC, Subject, BodyText, BodyHTMLText, Attachment, FromName);

Sends email to the given recipient with the parameters defined. It uses the outgoing email settings defined either in the Email Account of the login user or in SoftOne general settings (Internet tab).

Note that the last parameter (FromName) is not mandatory, so you can omit it, if not necessary.

Example

```
strTO = 'myemail@testemail.com';
strCC = '';
strBCC = '';
strSubject = 'Email Subject';
strBodyPlain = 'Email plain text - no html';
strBodyHTML = 'Dear Sir,<br>...';
strFromName = 'Last Name First Name';
strAttachment = 'C:\myfile.txt';
x = CallPublished('SysRequest.doSendMail2', VarArray(strTO, strCC, strBCC,
strSubject, strBodyPlain, strBodyHTML, strAttachment, strFromName, 8));
```

DoSendMail3(TO, CC, BCC, Subject, BodyText, BodyHTMLText, Attachment, FromName, EmailAccount);

Sends email to the given recipient using the email settings defined in the outgoing panel of given Email Account (Parameter 9).

Example

```
strTO = 'myemail@testemail.com';
strCC = '';
strBCC = '';
strSubject = 'Email Subject';
strBodyPlain = 'Email plain text - no html';
strBodyHTML = 'Dear Sir,<br>...';
strFromName = 'Last Name First Name';
strAttachment = 'C:\myfile.txt';
vEmailAccount = 10; //Defines the email account (Parameters - Users & Rights - Email
Accounts)
x = CallPublished('SysRequest.doSendMail3', VarArray(strTO, strCC, strBCC,
strSubject, strBodyPlain, strBodyHTML, strAttachment, strFromName, vEmailAccount,
9));
```

Evaluate(Module, InternalFunction: string);

Executes the internal function of the application.

Example

Get the id of a customer (Customer Code = 50) using the internal function ID.

```
sExpr = 'ID(' + QuotedStr('CUSTOMER') + ',' + QuotedStr('50') + ')';
id = CallPublished('SysRequest.Evaluate', VarArray(ImportModule('CUSTOMER'), sExpr,
2));
```

ExecuteXScript(Module, Language, ScriptName, Function);

Runs the Function of the script named "ScriptName". Use it when you want to create your own libraries and call functions from them.

Language: 1 = JavaScript, 2 = VBScript

Examples

1.RUN JavaScript code that displays message box (X.WARNING) in the screen.

```
x = ExecuteXScript(XModule,1,'function RUN() {X.WARNING('+#39+'Msg'+#39+');}','RUN');
//OR (Using double quotes)
x = ExecuteXScript(XModule,1,'function RUN() { X.WARNING("Msg");}','RUN');
```

2. Call and execute a function from the custom SBSL script named "myfunctions".

```
vScript = GetQueryResults('SoftOne', 'SELECT SOIMPORT FROM SOIMPORT WHERE CODE =
'+QuotedStr('myfunctions'),Null);
x = ExecuteXScript(vModule, 1, vScript, 'Convert');
```

3. Include "Advanced JavaScript" code and execute a specific function (myFunc).

```
Form{
}

connect Xplorer Softone {
    connect();
}

var x, vLibScript;

{
    vLibScript = 'lib.include("mylib");';
    x = CallPublished('SysRequest.ExecuteXScript', VarArray(XModule, 1, vLibScript, 'myFunc' ,
4));
}
```

ExecuteNetFunction(Dll FileName, Class (with Namespace), FunctionName, Params);

Runs the .NET DLL Function using the defined Params.

Class: Namespace.ClassName

Attention: .NET function must be defined as static int.

Example

JavaScript: Call .NET function (CallFromSoftOne) using browser right-click menu option (specified command)

```
function ON_CREATE() {
    var vBrowserMenu = X.EXEC('CODE:ModuleIntf.FindXStrings', X.MODULE, 'BRMENU');
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '20220101=1;Run DLL Job');
    X.EXEC('CODE:SysRequest.RefreshPopupMenu', X.MODULE, 'BRMENU', 1);
}

function EXECCOMMAND(cmd) {
    if (cmd == 20220101) {
        var xplorerDir = X.DIR('EXE');
        X.EXEC('CODE:SysRequest.ExecuteNetFunction', xplorerDir + 'S1DevInProcess.dll',
'S1DevInProcess.S1', 'CallFromSoftOne', 'my message;50');
    }
}
```

C# function CallFromSoftOne – Login – SELRECS – Object Events – DLL Form as new Tab in Customer object.

```
namespace S1DevInProcess
{
    public class S1DevInProcess
    {
        public static XSupport s1Support;
        public static string strAppName = "S1Dev ver." + string.Format("{0}.{1}",
Assembly.GetExecutingAssembly().GetName().Version.Major, Assembly.GetExecutingAssembly().GetName().Version.Minor);
    }

    public class S1
    {
        public static int CallFromSoftOne(string data)
        {
            string[] args = data.Split(';');
            //string param1 = args[0];
            //int param2 = int.Parse(args[1]);
            MessageBox.Show("Test Msg " + args[0], S1DevInProcess.strAppName, MessageBoxButtons.OK, MessageBoxIcon.Information);
            return 1;
        }
    }

    public class S1Init : TXCode
    {
        public override void Initialize()
        {
            MessageBox.Show("DLL Login OK", S1DevInProcess.strAppName, MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }

    [WorksOn("ITEM")]
    public class Items : TXCode
    {
        public XTable XItems;
        public override void Initialize()
        {
            base.Initialize();
            XModule.SetEvent("ON_ITEM_NAME", MyfunctionToGetTheEvent);
            XItems = this.XModule.GetTable("ITEM");
        }
        public override void BeforePost()
        {
            base.BeforePost();
            MessageBox.Show("Posted from DLL. Current Name is:" + XItems.Current["NAME"].ToString());
        }

        private void MyfunctionToGetTheEvent(object Sender, XEventArgs e)
        {
            string str = string.Format("Name changed to '{0}'", XItems.Current["NAME"]);
            MessageBox.Show(str, S1DevInProcess.strAppName);
        }
    }
}
```



```

[WorksOn("SERIES")]
public class Series : TXCode
{
    public XTable XSeries;
    public override void Initialize()
    {
        base.Initialize();
        XSeries = this.XModule.GetTable("SERIES");
    }
    public override void AfterLocate()
    {
        base.AfterLocate();
        MessageBox.Show(String.Format("Located Series with SOSOURCE {0}", XSeries.Current["SOSOURCE"]), "DLL Msg");
    }
    public override void BeforePost()
    {
        base.BeforePost();
        MessageBox.Show("Posted Series from dll", S1DevInProcess.strAppName);
    }
}

[WorksOn("CUSTOMER")]
public class S1Customer : TXCode
{
    public XTable XCustomer;
    public override void Initialize()
    {
        XCustomer = this.XModule.GetTable("CUSTOMER");
        XModule.SetEvent("ON_TRDR_NAME", TRDR_NAME_Changed);
    }

    public override void OnFormLoad(string formname)
    {
        base.OnFormLoad(formname);
        try
        {
            FormInCustomer fInCustomer = new FormInCustomer(); //DLL Form
            XModule.InsertControl(fInCustomer, "+PAGE(Pg2,TabCaption)"); //Adds the DLL Form after the tab "Pg2"
        }
        catch (Exception ex)
        {
            throw new Exception("Could not add the DLL form. Error message:" + ex.Message);
        }
    }

    public override void AfterLocate()
    {
        base.AfterLocate();
    }
    public override object ExecCommand(int Cmd)
    {
        switch (Cmd)
        {
            {
                case -2:
                    MessageBox.Show("Copy from Buffer", S1DevInProcess.strAppName);
                    break;
                case 20190219:
                    string message = "";
                    if (XModule.Params.Any(x => x != null && x.StartsWith("SELRECS")))
                    {
                        message = XModule.Params.First(x => x.StartsWith("SELRECS")); //SelectedRecords parameter (String array)
                    }
                    else
                    {
                        message = "No Customers selected!";
                    }
                    MessageBox.Show(message, S1DevInProcess.strAppName, MessageBoxButtons.OK, MessageBoxIcon.Information);
                    break;
            }
        }
        return base.ExecCommand(Cmd);
    }
    public override void BeforePost()
    {
        base.BeforePost();
        if (XCustomer.Current["Name"].ToString() == "Test")
        {
            throw new Exception("Test is not permitted");
        }
    }

    private void TRDR_NAME_Changed(object Sender, XEventArgs e)
    {
        string s = string.Format("Name changed to '{0}'", XCustomer.Current["Name"]);
        MessageBox.Show(s, S1DevInProcess.strAppName);
    }
}
}
}

```

ExecuteReport(ObjectName, ListName, Filters, Output, Template);

Runs the report using the specified filters or template (predefined list) and exports it to the given filename or printer. Parameter "Template" can be omitted.

ObjectName: ReportObjectName or Report (for reports created using the tool "Open Designed Report")

Filters: Reports Filters using the same syntax as in FORCEFILTERS

Output: Printer Name or Filename

Examples

1. Run the report "Customer Statement" using the photo 'Template1' and specific extra filters.

Export it in txt, pdf or printer.

```
// vOutput = 'C:\\'+ VarToStr(sTRDR.NAME)+' .txt';
// vOutput = 'C:\\'+ VarToStr(sTRDR.NAME)+' .pdf';
// vOutput = 'HP LaserJet Series';
x = executeReport('CUST_STM','B2B',
'QUESTIONS.FCODE=104&QUESTIONS.TCODE=104&QUESTIONS.FROMDATE=01/02/2022&QUESTIONS.TODA
TE=01/03/2021',vOutput,'Template1');
```

2. Run the custom Open Designer Report named "Custom Statement" and export it in Excel format

```
x = executeReport('REPORT','Custom Statement','vFilter1=101&vFilter2=01/04/2022',
'C:\\Temp\\Statement.xls');
```

3. Javascript code that prints the custom Open Designer Report named "WS_Customer Statement" and exports it in pdf format. The code runs for the selected Customer records through right click menu in the Browser.

```
function ON_CREATE() {
    var vBrowserMenu = X.EXEC('CODE:ModuleIntf.FindXStrings', X.MODULE, 'BRMENU');
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '--');
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '202106103=1;Execute Open Designed
Fast Report and save to PDF file');
    X.EXEC('CODE:SysRequest.RefreshPopupMenu', X.MODULE, 'BRMENU', 1);
}

function EXECCOMMAND(cmd) {
    if (cmd == 202106103) {
        ExecOpenDesignedFastReportSavePDF();
    }
}

function ExecOpenDesignedFastReportSavePDF() {
    var vSelRecs;
    var vFilename = "";
    var vFilters = "";
    vSelRecs = X.GETPARAM('SELRECS');
    vSelRecs = vSelRecs.replace(/\/?/g, ",");
    var ds = X.GETSQLDATASET("SELECT TRDR, CODE, NAME, EMAIL FROM TRDR WHERE " +
vSelRecs, null);
    //var vCusCodes = X.SQL('SELECT STRING_AGG(CODE, ';') FROM TRDR WHERE ' +
vSelRecs, null); //SQL Server 2017
    if (ds.RECORDCOUNT > 0) {
        ds.FIRST;
        while (!ds.EOF) {
            vFilename = "C:\\MyTemp\\" + ds.NAME + ".pdf";
            vFilters = 'vCustomer=' + ds.TRDR + '&vFilter1=' +
X.EVAL("FormatDateTime('dd/mm/yyyy',X.SYS.FYDATE)") + '&vFilter2=' +
X.EVAL("FormatDateTime('dd/mm/yyyy',X.SYS.LOGINDATE)");
            X.EXEC('CODE:SysRequest.executeReport', 'REPORT', 'WS_Customer
Statement', vFilters, vFilename);
            ds.NEXT;
        }
    }
}
```

GetFieldCaption(Field);

Returns the caption of the defined field.

Example

Custom javascript function that returns the caption of any field.

```
function GetCaption(strTable, strField) { //Get Field Caption
    var ds = X.EXEC('CODE:ModuleIntf.GetDataSet', X.MODULE, strTable);
    var field = X.EXEC('CODE:ModuleIntf.GetField', ds, strField);
    return X.EXEC('CODE:SysRequest.GetFieldCaption', field);
}
//Example of use
GetCaption("SALDOC", "TRDR");
```

GetFormModule(Object);

Creates an instance of the module for the object that was previously created using the function CreateForm.

Syntax:

```
vForm = CallPublished('SysRequest.CreateForm', VarArray('S1Object[FORM:MyForm]', 5, 2));
vFormModule = CallPublished('SysRequest.GetFormModule', vForm);
```

GetNewID(Module);

Returns the id of the record that has already been posted to "Module".

Example

Get the ids of the created sales document records.

```
Form {
    [TABLES]
        ImpTable=;;;Master;3;0
    [ImpTable]
        vSeries=2;15;1;1;0;Sales Series;SERIES(W[SOSOURCE=1351]);;
        vWhere=16;80000;0;1;0;Browser Selecton;;;&SELRECS;&SELRECS
        vMess=16;64000;0;1;1;Messages.....;
    [PANELS]
        PANEL10 = 0;Options;0;100,G10,N
        PANEL11 = 4;Job Messages;0;100,G10,N,L3
    [PANEL10]
        ImpTable.vSeries
    [PANEL11]
        ImpTable.vMess
}

include 'PILib';
include 'ModuleIntf';
include 'SysRequest';
var x, vMess, vTot, vCurRow, vErroRow, vDsHead, vModule, vDsLines, vNewID;

Connect Xplorer SoftOne {
    connect ();

    sHeader = SELECT DISTINCT FINDOC.FINDOC
                ,FINDOC.TRNDATE
                ,FINDOC.FINCODE
    FROM FINDOC
    WHERE ($REMOVEQUOTES(:$ImpTable.vWhere))
           AND FINDOC.SOSOURCE=1351
           AND FINDOC.COMPANY=:$X.SYS.COMPANY
           AND FINDOC.ISCANCEL=0;

    sLines = SELECT DISTINCT FINDOC.FINDOC
              ,MTRLINES.MTRLINES
              ,MTRLINES.MTRL
```

```

        ,MTRLINES.QTY1
        , (MTRLINES.QTY1-MTRLINES.QTY1COV-MTRLINES.QTY1CANC) AS NEWQTY
        ,MTRLINES.PRICE
        ,MTRLINES.DISC1PRC
        ,MTRLINES.LINEVAL
FROM FINDOC
INNER JOIN MTRLINES ON FINDOC.FINDOC=MTRLINES.FINDOC
WHERE ($REMOVEQUOTES(:$ImpTable.vWhere))
      AND FINDOC.SOSOURCE=1351
      AND FINDOC.COMPANY=:$X.SYS.COMPANY
      AND FINDOC.ISCANCEL=0
      AND FINDOC.FULLYTRANSF IN (0,2)
      AND FINDOC.FINDOC=:$sHeader.FINDOC;
}

{
  fetch sHeader {
    vModule = CreateModule('SALDOC,WARNINGS:OFF,NOMESSAGES:1');
    x = InsertModule(vModule);
    vDsHead=GetDataSet(vModule,'FINDOC');
    x = DataSetEdit(vDsHead);
    x = SetFieldValue(vDsHead,'SERIES',:$ImpTable.vSeries);
    x = SetFieldValue(vDsHead,'TRNDATE',:$X.SYS.LOGINDATE);
    x = SetFieldValue(vDsHead,'TRDR',sHeader.TRDR);
    x = DataSetPost(vDsHead);

    vDsLines = GetDataSet(vModule,'ITELINES');
    x = DataSetEdit(vDsLines);
    fetch sLines {
      x = DataSetAppend(vDsLines);
      x = SetFieldValue(vDsLines,'MTRL',sLines.MTRL);
      x = SetFieldValue(vDsLines,'QTY1',sLines.NEWQTY);
      x = SetFieldValue(vDsLines,'PRICE',sLines.PRICE);
      x = SetFieldValue(vDsLines,'DISC1PRC',sLines.DISC1PRC);
      x = SetFieldValue(vDsLines,'FINDOCS',sLines.FINDOC);
      x = SetFieldValue(vDsLines,'MTRLINESS',sLines.MTRLINES);
      x = DataSetPost(vDsLines);
    }

    x = PostModule(vModule);
    vNewID = GetNewID(vModule);
    x = DestroyModule(vModule);
  }
}

```

GetTableFieldInfos(TableName);

Returns text containing information for each field in the table.

GetTableFieldNames(TableName);

Returns the field names of table "TableName".

GetTableMoreInfo(TableName);

Returns text with the table information.

GetXplorerDataDir(Params);

Returns the installation directory of SoftOne application.

GetXplorerLogDir(Params);

Returns the directory of log files.

GetXplorerOfflineDir(Params);

Returns the directory of offline files.

GetXplorerTempDir(Params);

Returns the directory of temp files.

PrintForm(Module, Form, Printer, FileName);

Prints the form defined by the 'Form' code for the given Module and Printer. In case of printing to a file, then FileName must be defined.

Example

Locate the record id=1000 in purchase documents and print it to ASCII file using the form with code=100.

```
iFINDOC = 1000;
vFile = 'C:\\Softone\\' + VarToStr(:X.SYS.COMPANY) + '_' + iFINDOC + '.TXT'; //
ASCII File Name and Path
vPURModule = CallPublished('ModuleIntf.CreateModule','PURDOC,WARNINGS:OFF');
x = CallPublished('ModuleIntf.LocateModule',VarArray(vPURModule,iFINDOC,2)); //Locate
record
x = CallPublished('SysRequest.PrintForm', VarArray(vPURModule,100,'928', vFile,4));
//Save to ASCII file
```

RefreshPopupMenu

Refreshes the items of the Context Menu. Combined with PiLib.TStringsSetItem or PiLib.TStringsAdd.

Example

Form Javascript - Add menu item in Browser Context Menu of any object.

```
function ON_CREATE() {
    var vBrowserMenu = X.EXEC('CODE:ModuleIntf.FindXStrings', X.MODULE, 'BRMENU');
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201705291=One Record Job');
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201705292=1;Many Records Job');
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201705292=1;SubJob1?MySubMenu');
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201705292=1;SubJob2?MySubMenu');
    X.EXEC('CODE:PiLib.TStringsAdd', vBrowserMenu, '201910023=3;Displayed on Lines
Analysis Browser');
    X.EXEC('CODE:SysRequest.RefreshPopupMenu', X.MODULE, 'BRMENU', 1);
}
```

RequestEdit(Command: string);

Executes the command. It works only on client scripts.

Example

Execute the custom design report named "Myreport" filtered for TRDR = 1000.

Dialog filter field linked to TRDR is "USRREPORTV1L".

```
var vTRDR, sObj, x;
vTRDR = 1000;
sObj = 'REPORT[LIST=MyCustomReport,FORCEFILTERS=USRREPORTV1L='+VarToStr(vTRDR)+' ,AUTOEXEC=1]';
x = CallPublished('SysRequest.RequestEdit', sObj);
```

ShowHtmlMessageForm(TStringsHandle);

Opens a Web form with the text set defined in the "TStringsHandle" string list. Similar to the results screen of SBSL scripts.

Example

```
fText = 'mytext';
fStrings = TStringsCreate;
x = TStringsSetText(fStrings, fText);
x = ShowHtmlMessageForm( fStrings);
```

ShowInTheMap(Params1, Params2, Params3);

Returns in a web form the location defined by the coordinates.

Params1: Latitude, **Params2:** Longitude, **Params3:** Web Form to open.

Example

```
vDs = CallPublished('ModuleIntf.GetDataSet',VarArray(XModule,'GPNT',2));
vFld1 = CallPublished('ModuleIntf.GetFieldValue',VarArray(vDs,'SOWIDTH',2));
vFld2 = CallPublished('ModuleIntf.GetFieldValue',VarArray(vDs,'SOLENGTH',2));
vFld3 = CallPublished('ModuleIntf.GetFieldValue',VarArray(vDs,'CODE',2));
Data = CallPublished('SysRequest.ShowInTheMap', VarArray(vFld1,vFld2,vFld3, 3));
```

ShowWebPage(URL: string);

Opens the HTML page defined by the URL string.

Example

```
x = ShowWebPage('www.softone.gr');
```

XSendWebSMS(PhoneNumber: string, message: string);

Sends SMS to the defined Phone Number. Uses the WEB SMS Account defined in SoftOne Settings (Xplorer.cfg)

Example

```
x = CallPublished('SysRequest.XSendWebSMS',VarArray('123456789', 'mymessage', 2));
```

H. Case Studies

1. ReUpdate Sales Documents

Reupdate sales documents (based on a specific query) by changing the value of the field PAYMENT.

```
form {
}

connect Xplorer xData {
  connect();
  findoc=select findoc, sosource, fincode, trndate, sumamnt
    from findoc
    where company=:$X.SYS.COMPANY
    and fiscprd=:$X.SYS.FISCPRD
    and sosource = 1351
    and fprms in
(7061,7062,7063,7064,7066,7067,7071,7072,7073,7076,7082,7127,7128)
    --and trndate >= '20180110'
  order by 1 desc;
}

var
  x, Module, DsFinDoc, Dslines, vFieldVal, vFieldVal1, Count;
{
  Module =
  CallPublished('ModuleIntf.CreateModule','SALDOC,WARNINGS:OFF,NOMESSAGES:1');
  DsFinDoc = CallPublished('ModuleIntf.GetDataset',VarArray(Module,'FINDOC',2));
  Count = 0;
  fetch finDoc {
    Count = Count + 1;
  }
  x = SendResponse(Count, 'RESULTS.TOTREC');

  Count = 0;
  fetch findoc {
    x = CallPublished('ModuleIntf.LocateModule',VarArray(Module,findoc.findoc,2));
    Count = Count + 1;
    x = SendResponse(Count, 'RESULTS.CURREC');
    x = CallPublished('ModuleIntf.DatasetEdit',DsFinDoc);

    x = CallPublished('ModuleIntf.SetFieldValue',VarArray(DsFinDoc, 'PAYMENT', 1000,
3));

    x = CallPublished('ModuleIntf.PostModule', Module );
  }
  x = SendResponse(Count, 'RESULTS.CURREC');
}
```

2. Import Customers from Excel

Import customers from Excel file into a temp table of the database and then insert the records in the customers module. The script retrieves data from an excel file that uses columns with specific headers (Line, Name, VAT, City, Address, Branch code).

```
Form {
  [TABLES]
  ImpTable =;;; Master; 3; 0

  [ImpTable]
  vMess=16;64000;0;1;1;Transfer messages...;;;
  vExcelFile=1;4000;1;1;0;Excel file;$Filename;;'C:\test1.xlsx';
  vDBTable=1;64;1;0;0;TableName;;;CCCEXCEL_CUST
  vDrop=2;15;1;0;0;Drop and Create Table;$Y;;1;
  vInsRec=2;15;1;0;0;Insert Records;$Y;;1;

  [PANELS]
  PANEL10=0;;0;50,G10,N
  PANEL13=4;Transfer messages...;0;100,G10,N,L3

  [PANEL10]
  ImpTable.vExcelFile
  ImpTable.vDBTable
  ImpTable.vDrop
  ImpTable.vInsRec

  [PANEL13]
  ImpTable.vMess
}

Import ImpCUSTOMER(sCus,sCusBranch) into 'CUSTOMER, WARNINGS:OFF, NOMESSAGES:1'
{
  CUSTOMER sCus {
    CODE          = '*';
    NAME          = sCus.cNAME;
    AFM           = sCus.cAFM;
    CITY          = sCus.cCITY;
    ADDRESS       = sCus.cADDRESS;
  }

  CUSBRANCH sCusBranch {
    CODE          = sCusBranch.bCODE+'.01';
    NAME          = sCusBranch.bNAME;
    ISCENTER      = sCusBranch.bISCENTER;
    ADDRESS       = sCusBranch.bADDRESS;
    CITY          = sCusBranch.bCITY;
  }
}

Connect Xplorer SoftOne {
  connect ();

  sCus = SELECT
          [Line]          AS Line
        , [Name]          AS cNAME
        , [VAT]           AS cAFM
        , [City]          AS cCITY
        , [Address]       AS cADDRESS
    FROM $REMOVEQUOTES (: $ImpTable.vDBTable);

  sCusBranch = SELECT
```



```

        [BranchCode] AS bCODE
                        ,[Name] AS bNAME
                        ,1 AS bISCENTER
                        ,[Address] AS bADDRESS
                        ,[City] AS bCITY
FROM $REMOVEQUOTES(:$ImpTable.vDBTable)
WHERE [Line] = :$sCus.Line ;
    }

var
    Jstr, objExcel, vscript, x, vTot, vRow, vRowCancel, vMess, Ds, vError, str1,
vModule, ifin, vTask, vSalesman, newCusCode, vMessErrors, vMessTask, vCurRec;
{
    vError=0;
    vMess='';
    if (vError=0)
    {
        vMess = '===== J O B   S T A R T ====='+#13+#10+#13+#10;
        x=SendResponse( vTot, vRow, vRowCancel, 'Start job...', vMess,
'RESULTS.TOTREC;RESULTS.CURREC;RESULTS.CANREC;RESULTS.LABELTEXT;ImpTable.vMess');
//>>>===== Transfer Excel to SQL Table - Table initialization =====
        if(:ImpTable.vDrop=1)
        {
            x = SafeExecSQL('SoftOne','DROP TABLE ' + :ImpTable.vDBTable, Null );
            x = CallPublished('SysRequest.Evaluate', VarArray(ImportModule('ImpCUSTOMER'),
'ExcelImport('+#39+:ImpTable.vExcelFile+#39+', '+#39+:ImpTable.vDBTable+#39+',1,2)',
2));
        }
//<<<===== Transfer Excel to SQL Table - Table Initialization =====

//>>>===== Insert CUSTOMER (Customers) =====
        vCurRec = 0;
        vTot = 0;
        vRowCancel = 0;
        fetch sCus vTot = vTot + 1;
        x = SendResponse( vTot, 0, 0, 'RESULTS.TOTREC;RESULTS.CURREC;RESULTS.CANREC' );
        vMess = vMess + 'A. CUSTOMERS IMPORT... '+#13+#10;
        x = SendResponse( vMess, 'Customer Import...', 'ImpTable.vMess;RESULTS.LABELTEXT'
);
        fetch sCus {
            if(:ImpTable.vInsRec = 1)
            {
                ImpCUSTOMER(sCus,sCusBranch);
                vCurRec = vCurRec + 1;
                if (ImportError = 0)
                {
                    newCusCode = GETQUERYRESULTS('SoftOne','SELECT CODE FROM TRDR WHERE
TRDR='+VarToStr(resultnum),NULL);
                    vMess = vMess + ' ' + VarToStr(vCurRec) + '. Name:
'+VarToStr(sCus.cName)
                                + ' ...OK - New Code:
'+VarToStr(newCusCode)+#13+#10;
                    x=SafeExecSQL('SoftOne','UPDATE ' + :ImpTable.vDBTable + '
SET [Customer code]='+VarToStr(newCusCode)+' ,[Branch code]='+QUOTEDSTR('01')+' WHERE
LINE='+VarToStr(vCurRec), Null );
                }
                else
                {
                    vRowCancel = vRowCancel+1;
                    vMess = vMess + ' ' + VarToStr(vCurRec) + '. Name: '
                                +VarToStr(sCus.cName)+ ' ...ERROR - '+
ErrorMessage+#13+#10;
                    x=SafeExecSQL('SoftOne','UPDATE ' + :ImpTable.vDBTable + '
SET [Customer code]='+QUOTEDSTR('error')+' ,[Branch code]='+QUOTEDSTR('error')+' WHERE
LINE='+VarToStr(vCurRec), Null );
                }
            }
        }
    }
}

```

```

    }
        vSalesman = sCus.cSalesman;
        x = SendResponse(vTot, vCurRec, vRowCancel, vMess,
'RESULTS.TOTREC;RESULTS.CURREC;RESULTS.CANREC;ImpTable.vMess' );
    }
}
vMess = vMess + '- CUSTOMER IMPORT COMPLETION'+#13+#10+#13+#10;
x = SendResponse(vTot, vCurRec, vRowCancel, vMess,
'RESULTS.TOTREC;RESULTS.CURREC;RESULTS.CANREC;ImpTable.vMess' );
//<<<===== Import CUSTOMER (Customers) =====

vMess = vMess +#13+#10+ '===== J O B   E N D ====='+#13+#10;
x=SendResponse(vCurRec, vRowCancel, vMess, 'End of job...',
'RESULTS.CURREC;RESULTS.CANREC;ImpTable.vMess;RESULTS.LABELTEXT');
}
}

```

3. Add Customer Collections (UI)

Batch job that inserts customer collections for specific documents (based on SQL query). The collections documents are displayed in users screen and wait for the user to save them (command ",T").

```
Form {
  [TABLES]
  ImpTable =;;; Master; 3; 0

  [ImpTable]
  vMess=16;64000;0;1;1;Transfer messages...;;;

  [PANELS]
  PANEL13=4;Transfer messages...;0;100,G10,N,L3

  [PANEL13]
  ImpTable.vMess
}

Import ImpCFNCUSDOC(sDoc) into 'CFNCUSDOC,WARNINGS:OFF,NOMESSAGES:1,T' { // ,T
displays record in screen
  FinDoc sDoc {
    SERIES      = 3800;
    TRNDATE     = sDoc.TRNDATE;
    TRDR        = sDoc.TRDR;
    FINDOCS     = sDoc.FINDOC;
    COMMENTS    = 'Import from Import Script';
  }
  CashLines sDoc {
    SOPAYTYPE = 1;
    LINEVAL   = sDoc.sumamnt;
  }
}

connect Xplorer xData {
  connect();
  sDoc = select findoc, trdr, sosource, fincode, trndate, sumamnt
        from findoc
        where company=:$X.SYS.COMPANY
          and fiscprd=:$X.SYS.FISCPRD
          and sosource = 1351
          and fprms in
(7061,7062,7063,7064,7066,7067,7071,7072,7073,7076,7082,7127,7128)
          and trndate = '20030110'
        order by 1 desc;
}

var
  x, Module, DsFinDoc, Dslines, vFieldVal, vFieldVal1, Count, vMess, vRow,
  vRowCancel, vTot;
{
  vTot = 0;
  fetch sDoc vTot = vTot + 1;
  vMess = 'Start job.....'+#13+#10;
  x = SendResponse( vTot, 0, 0, 'Import collection documents...', vMess,
'RESULTS.TOTREC;RESULTS.CURREC;RESULTS.CANREC;RESULTS.LABELTEXT;ImpTable.vMess' );
  fetch sDoc {
    ImpCFNCUSDOC(sDoc);
    if (ImportError = 0) vRow = vRow + 1;
    else vRowCancel = vRowCancel + 1;
    x=SendResponse(vRow, vRowCancel, 'RESULTS.CURREC;RESULTS.CANREC');
  }
  vMess=vMess+'End of transfer...'+#13+#10;
  x=SendResponse(vRow, vRowCancel, vMess, 'End of import of collection
documents...', 'RESULTS.CURREC;RESULTS.CANREC;ImpTable.vMess;RESULTS.LABELTEXT');
}
```

4. Get String List Data

Batch job that displays the keys and values of the given string list. Enter a SoftOne string list in filters and run it to get its values. Use it for any SoftOne string list (SODTYPE, SOSOURCE, TAXDEVTYPE, etc.)

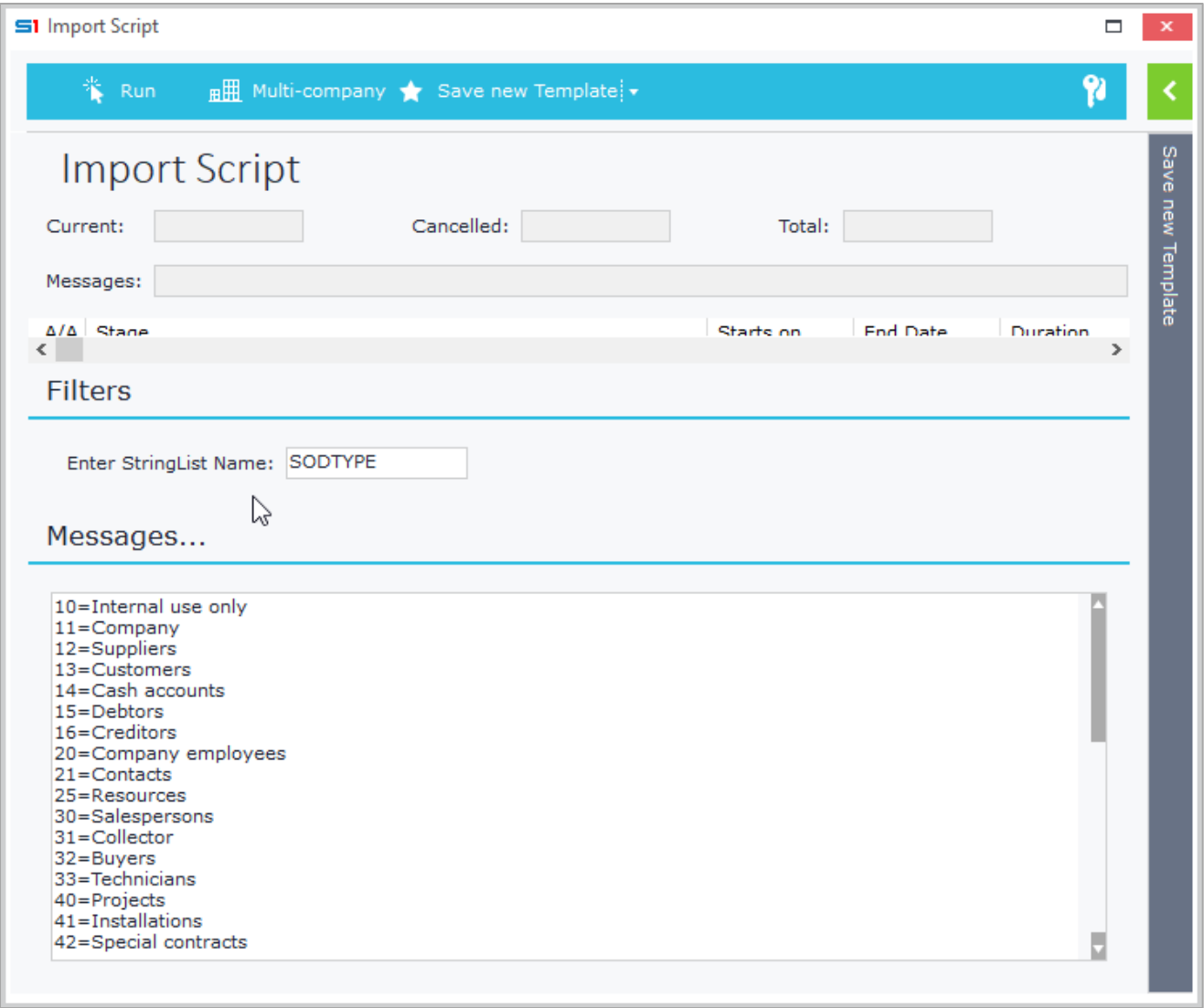


Figure CS4.1 – SOSOURCE string list

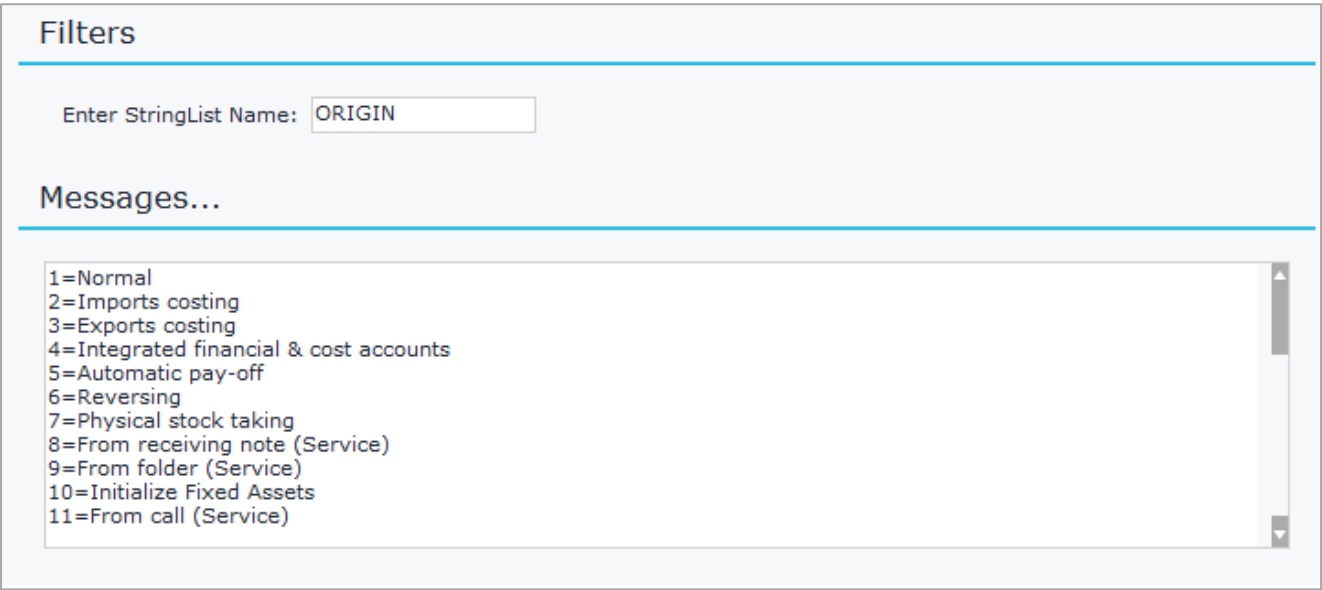


Figure CS4.2 – ORIGIN string list

```

Form
{
    [TABLES]
    ImpTable=;;;Master;3;0

    [ImpTable]
    vStringList= 1;25;0;1;0;Enter StringList Name;
    vImpMess = 16;64000;0;1;1;Messages...;;;2

    [PANELS]
    PANEL11=0;Filters;0;40,G11
    PANEL12 = 4;Messages...;0;100,H50

    [PANEL11]
    ImpTable.vStringList

        [PANEL12]
        ImpTable.vImpMess

}
include 'PILib';

Var
    vMess,x, tStringList, tStringListCount,i,ListItem;

Connect Xplorer SoftOne
{
    connect();
}

{
    vMess = '';
    tStringList = GetXStrings(VarToStr(:ImpTable.vStringList)); //Handler

    if (tStringList<>0)
    {
        tStringListCount = TStringsCount(tStringList); //Get ListCount
        i=0;
        while( i <= (tStringListCount-1))
        {
            ListItem = TStringsGetItem(tStringList,i); //Get Current Item
            (e.g l=Item1)
            if (Copy(ListItem,Pos('=' ,ListItem)+1,Len(ListItem)) <> '')
            //Check if Item Value is not null
            {
                vMess = vMess + ListItem + #13 + #10 ;
                x = SendResponse(vMess, 'ImpTable.vImpMess');
            }
            i = i + 1;
        }
    }
    else
    {
        vMess = '';
        x = SendResponse(vMess, 'ImpTable.vImpMess');
        x = RAISEEXCEPTION('String List does not exist !!!!! ');
    }
}

```

5. Read ASCII file using specific Codepage

Batch job that reads an ASCII file, iterates through file lines and displays the data in a memo text box.

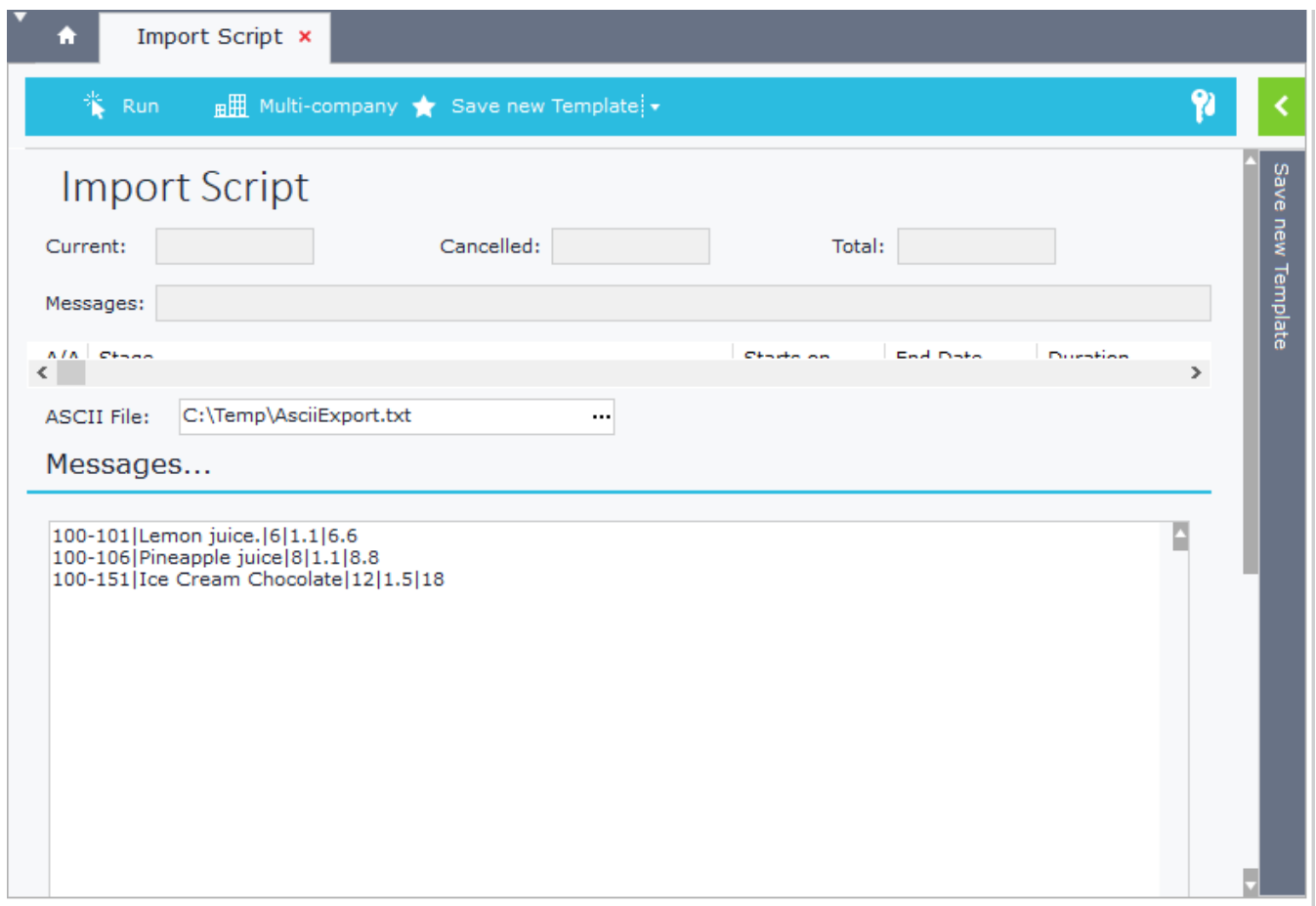


Figure CS5.1 – Job Results

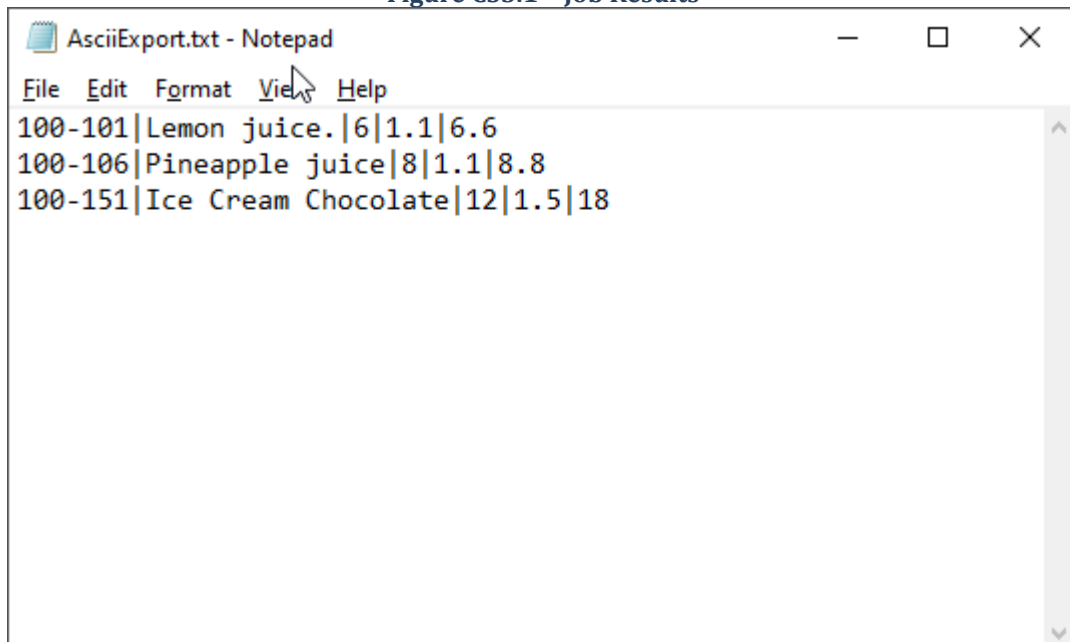


Figure CS5.2 – Ascii File

```

form
{
    [TABLES]
    ImpTable=;;;Master;3;0

    [ImpTable]
    vTxtFile=1;255;1;1;0;ASCII File;$Filename;;;
    vImpMess=16;64000;0;1;1;Messages...;;;2

    [PANELS]
    PANEL11=0;;0;050,000,000,000,N,G10
    PANEL14=4;Messages...;0;100,G10,N,L20

    [PANEL11]
    ImpTable.vTxtFile

    [PANEL14]
    ImpTable.vImpMess
}

include 'PiLib';

Connect Xplorer Softone {
    connect();
}

var x, vRow, vRowCancel, UserResp, vMess, TxtRead, vLine;
{
    TxtRead=OpenTextCodePage(:ImpTable.vTxtFile,65001); //UTF-8
    //TxtRead = OpenText(:ImpTable.vTxtFile);
    vLine = '';
    vMess = '';
    UserResp = SendResponse(vMess , 'ImpTable.vImpMess');
    while (Eof(TxtRead) = 0)
    {
        vLine = ReadLine(TxtRead);
        if (Len(vLine) > 0)
        {
            vMess = vMess + vLine + Char(13);
            UserResp = SendResponse(vMess , 'ImpTable.vImpMess');
        }
    }
}
}

```

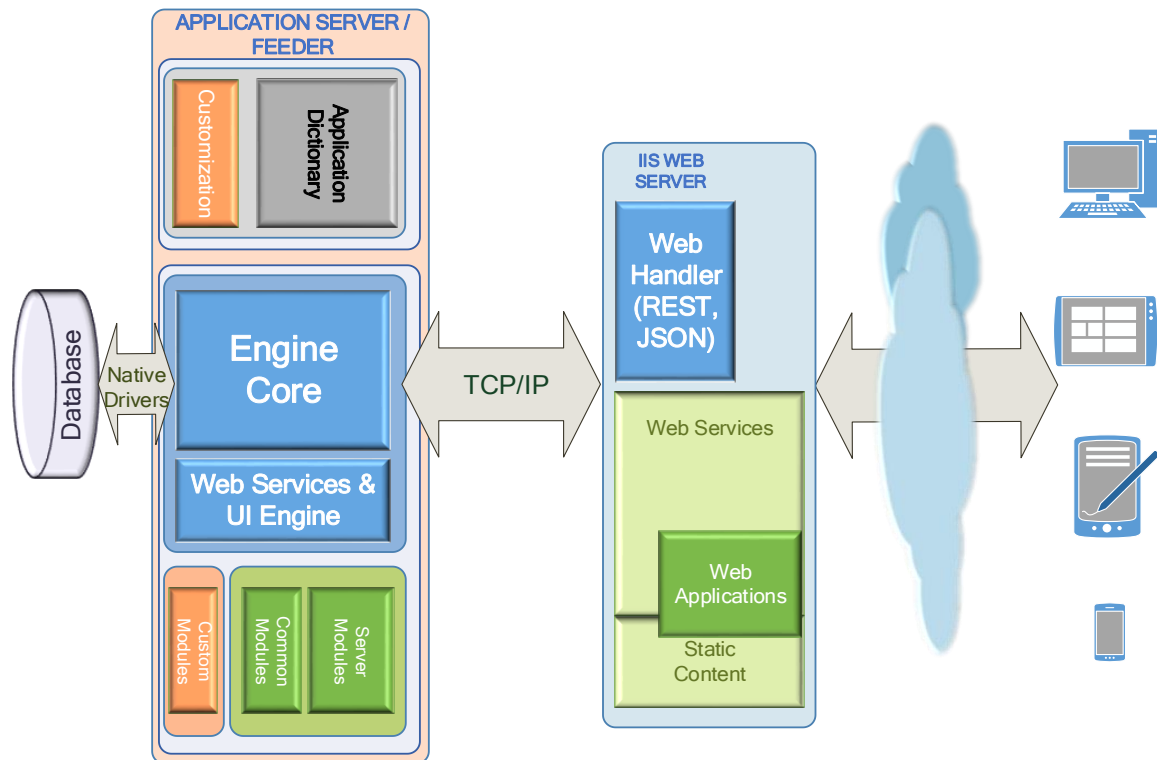
12. WEB SERVICES

- A. [Overview](#)
- B. [Open Enterprise Engine](#)
- C. [Methods / API Calls](#)
- D. [Case Studies](#)

A. Overview

SoftOne API provides Web Services that give you the ability to connect and manage data through your website or external application. Almost all the data and functionality of SoftOne is accessible through web services so the integration possibilities are endless.

Web services are available through Cloud or On Premise SoftOne installations using the SoftOne Open Enterprise Engine. This service can be activated through one click and makes data available using Secure TCP/IP connections.



The web services use **HTTP requests** to URLs given by SoftOne and the data are returned in **JSON format**. The generic format for the API endpoint is:

```
https://<RegisteredName or Soft1SerialNumber>.oncloud.gr/s1services?
```

where <RegisteredName> should be replaced with the name you used to register the SoftOne Open Enterprise Service. Suppose the registered name is *mycompany* then the API endpoint would be:

```
https://mycompany.oncloud.gr/s1services?
```

A simple webservice call example (HTTP-GET request) is: <https://demo.oncloud.gr/s1services?ping>

Please make note of the following:

- The majority of the calls use **HTTP-POST methods**.
- All HTTP-Post methods are accessed via: <https://RegisteredName.oncloud.gr/s1services>
- All methods (after Login) require a unique client identifier (**clientID**) which is obtained through the **Authenticate** method.
- Compression (Content encoding) used is **gzip**.

Tests on SoftOne web services can be done using the data of the demo database, which is accessed through the url: <https://webdemo.oncloud.gr/s1services>

B. Open Enterprise Engine

SoftOne Open Enterprise Engine is a SoftOne module. This module enables a one-click setup service, that connects on-premises application server with the cloud infrastructure.

B.1 Prerequisites & Setup

Web services are enabled only if you have activated the "Soft1 Open Enterprise Engine" License Module (Figure B1). Note that you can have one site registration per serial number.

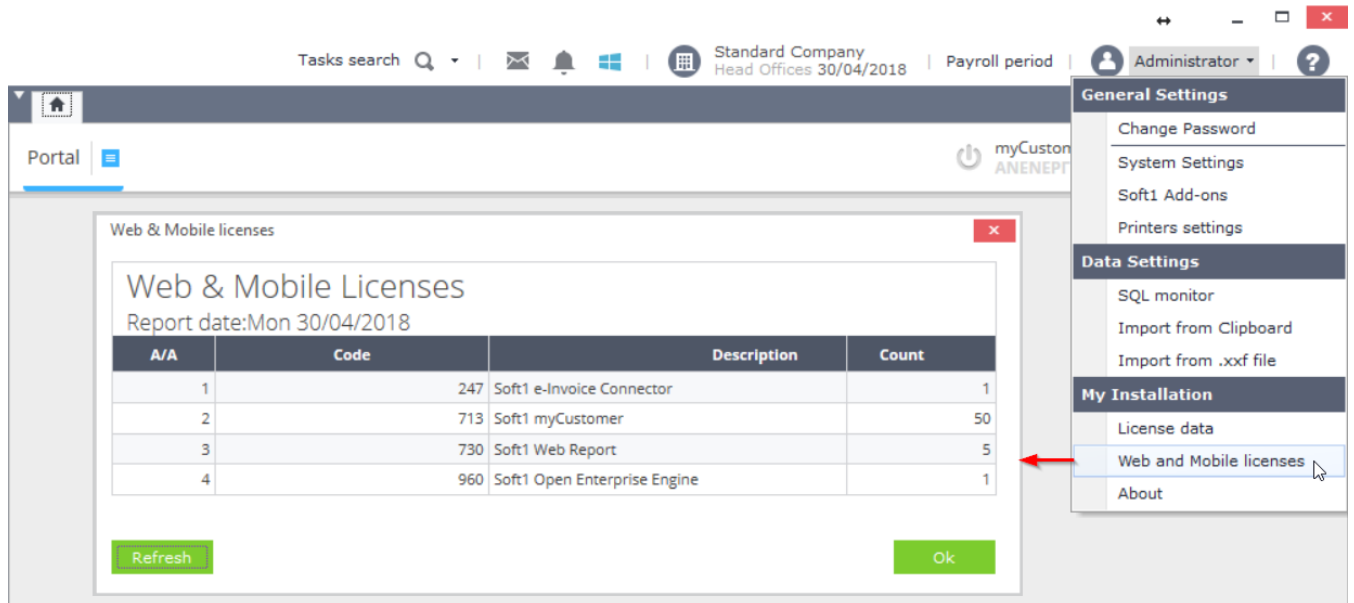


Figure B1

On-Premises installations require the local execution of the S1 Agent Service. This Service should run on a workstation that is always online to transfer data from the local SoftOne database to the Internet (using SoftOne web services). It is advisable to activate S1 Agent on the same workstation where SoftOne application server sits. Cloud installations on the other hand do not require any further setup for enabling the web services, but only the activation of the "Soft1 Open Enterprise Engine" License Module.

The service, in both on-premises or cloud installations, is started using the "cloud" button that sits on the main toolbar of SoftOne (Figure B2). This button is displayed only to users defined as Administrators by the application.

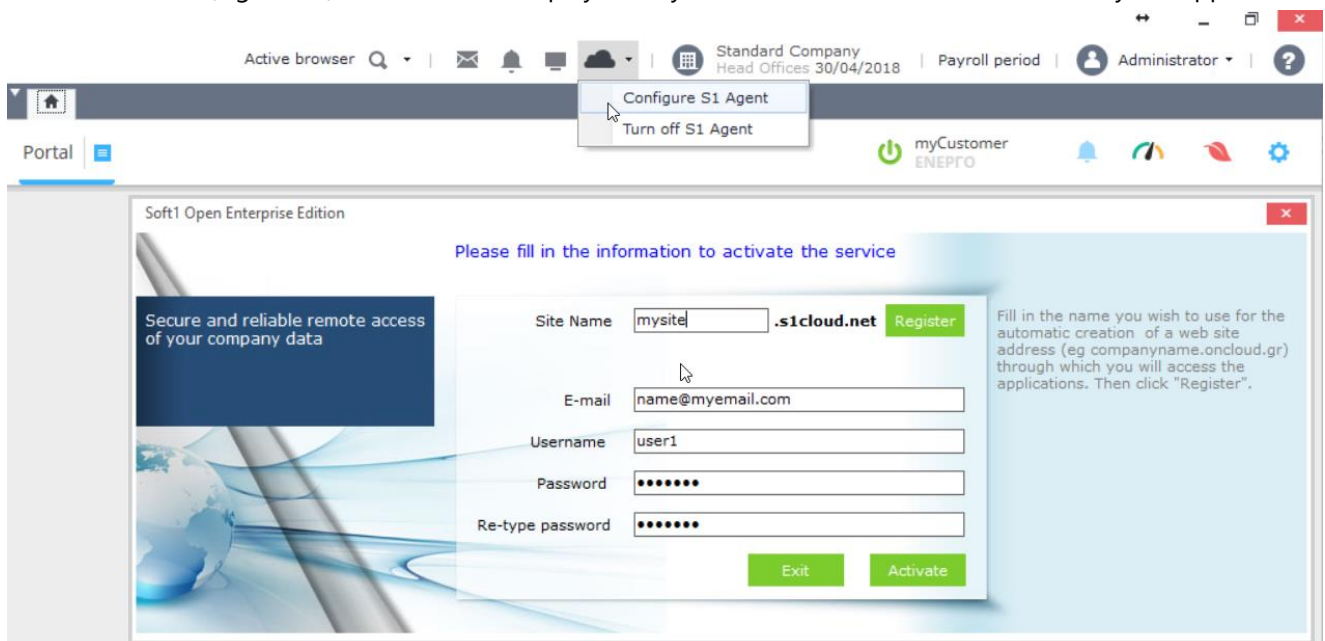


Figure B2

Before activating the S1 Agent Service make sure that the Windows user has administration rights.

Click on "**Configure S1 Agent**", enter a domain name and register the site.

Note: The domain name should not be prefixed with numbers and should not contain any capital letters.

The fields below the site name (e-mail address, UserName and password) are used to automatically create a new web account and menu, that are explained in the next section.

Finally, click on "**Activate**" button to complete the process of activating the S1 Agent Service.

Upon activating the Agent Service, a new connection file "**WEB.XCO**" is created inside the SoftOne folder and an e-mail is also send to the address entered, that informs the user that the service is activated. WEB.XCO file keeps all the info needed to connect the local SoftOne database to the internet thus enables the web services for this specific database.

In order to change the pc that serves your domain name you need first to **deactivate** the service from the initial workstation and then activate it in the new workstation.

Note that S1 Agent Service uses **SoftOne Licence Manager**, which means that it is preferable to use LM as a service, for logging off without losing the connection.

B.2 Custom Web & Mobile Apps

Custom Web and mobile applications must be defined inside the *Web Services* entity. Open *Web Services* module, create a new record and enter the web service id, code and description (Figure B4).

Note: **Web Service IDs should be greater than 1000**, since SoftOne uses the numbers below 1000 for internal web and mobile applications.

The id will be later used in all your web service calls (**applID**).

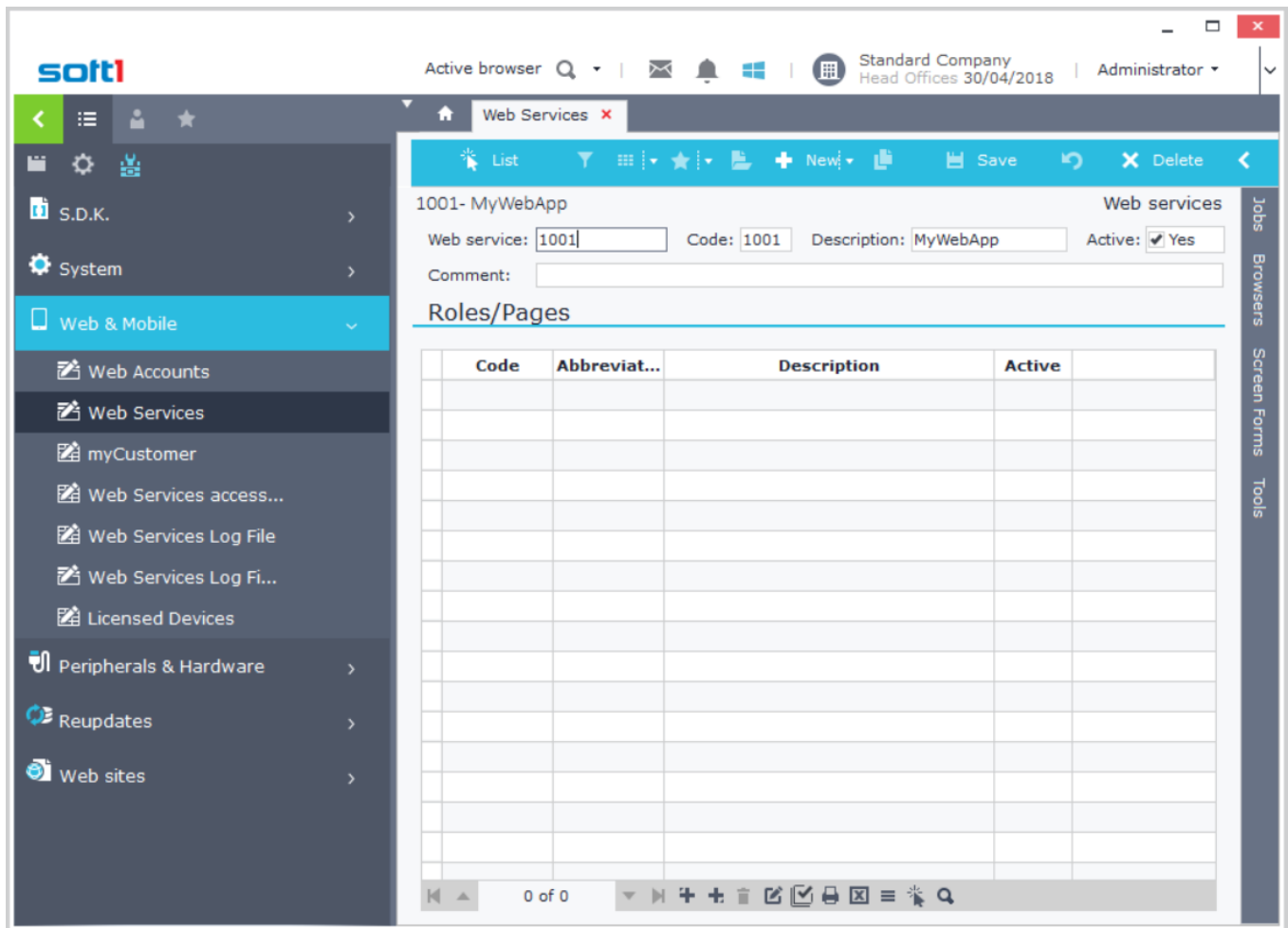


Figure B4

B.3 Web Accounts

Web accounts define the web users who will have access to your web and mobile applications. They are linked to a SoftOne database and transfer data to it. The procedure of creating a web account may be performed either automatically or manually.

Automated Process

Web accounts are auto created using the **"Configure S1 Agent"** option (as seen in the previous section). This brings up a window that requires fields: E-mail, Username and Password in order to Activate the service (Figure B5).

After completing this process, notice that the following tasks have been performed automatically:

- A new Web Account is created that uses the above credentials.
- Webservices: Web360, Soft1B2b, QuickView and MyPortal are auto-inserted for this account.
- Classic menus are created that can be used in Web accounts (or related jobs of objects).

Active browser | Standard Company | Head Offices 30/04/2018 | Payroll period | Administrator

Configure S1 Agent

Soft1 Open Enterprise Edition

Please fill in the information to activate the service

Secure and reliable remote access of your company data

Site Name: mysite .s1cloud.net Register

E-mail: name@myemail.com

Username: user1

Password:

Re-type password:

Exit Activate

Fill in the name you wish to use for the automatic creation of a web site address (eg companyname.oncloud.gr) through which you will access the applications. Then click "Register".

Figure B5

Manual Process

Web & Mobile services configuration is performed through the Menu: Tool –Parameters – Web & Mobile. Open the **Web accounts** entity to define the Web users that will use your applications (web services) (Figure B6).

demo- Web accounts

Account data

Code: demo Password: ***** Master E-mail: demo@myemail.gr Active: ☒ Yes

Relations Other data

New

	Web service	Login user	Login Company	Login branch	Entity	Code	Description
1	Web360	demo user	Standard Company	Head Offices	Employee	900	Demo User
2	Web360	Salesman	Standard Company	Head Offices	User		
3	Web360	Sales Supervisor	Standard Company	Head Offices	User		
4	Web360	General Manager	Standard Company	Head Offices	User		
5	Soft1 B2B	Administrator	Standard Company	Head Offices	Customer	101	Salesconsul GmbH
6	Soft1 B2B	teo	Standard Company	Head Offices	Customer	103	Holding AE
7	Soft1 SFA	Salesman	Standard Company	Head Offices	User		
8	QuickView	teo	Standard Company	Head Offices	User		
9	MyPortal	teo	Standard Company	Head Offices	User		
10	mCRM	teo	Standard Company	Head Offices	User		
11	web	demo user	Standard Company	Head Offices	User		

Figure B6

Upon creating a new web account entry, enter the code, the password and the email. Inside the tab "Relations" define the services that the web user will have access, using the button "New" (Figure B7). This button brings up the form "Relates to...", where you can state the web service application, the login user, the company and the branch. Double click a line to modify the data of the related service.

Note: Web accounts must not exceed the number of installation user licenses, because an error will occur when you try to make a request to a SoftOne webservice.

SI Relates to...

Web service: 157 Web360

Login user: 500 demo user

Login Company: 1000 Standard Company

Login branch: 1000 Head Offices

Entity: Employee

Employer: 900 User

Menu: 815 SelfService

OK Cancel

Figure B7

C. Methods / API Calls

All the following methods use a custom **appID (3001)** that needs to be created in SoftOne Web Accounts.

Ping (Get Method)

Pings the SoftOne API. It is the simplest method that you can use to see that the API is functioning.

Request URL

```
https://RegisteredName.oncloud.gr/s1services?ping
```

Response Sample

```
Ping from Softone WebModule  
ISAPI is working - Module configured  
Running in TCP mode [127.0.0.1:22099 Connected]  
Cached: 39 Forwarded: 28647 Responses (0%)  
Cache: X:\ProgramData\SoftOne\webdata\
```

Refresh(Get Method)

Restarts and refreshes the SoftOne API for the specific URL and returns OK when job is completed.

Usually used when web application parameters are changed in SoftOne and user needs to see the changes immediately (e.g. change of the user web menu in SoftOne S360 application).

Request URL

```
https://RegisteredName.oncloud.gr/s1services?refresh
```

Response

```
OK
```

Login(Post Method)

It is used to Login the web account defined by *RegisteredName* in request URL. It returns a temporary access token (clientID) that can be later used in the "authenticate" method. It also returns the available set of login permissions for the User (Company, Branch, etc.).

Request URL

```
https://RegisteredName.oncloud.gr/s1services
```

Request Sample

```
{
  "service": "login",
  "username": "MyUser",
  "password": "123456",
  "appId": "3001"
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed. Current method is <i>login</i> .
username	string	Username (Code) defined in SoftOne Web Accounts
password	string	Password defined in SoftOne Web Accounts
appId	string	ID of the Service defined in SoftOne Web Accounts

Response Sample

```
{
  "success": true,
  "clientID": "9J8pJ6La...Ga5mHG",
  "objs": [ {
    "COMPANY": "1000",
    "COMPANYNAME": "Standard Company",
    "BRANCH": "1000",
    "BRANCHNAME": "Head Offices",
    "MODULE": "0",
    "MODULENAME": "<Blank>",
    "REFID": "500",
    "REFIDNAME": "demo",
    "USERID": "500",
    "FINALDATE": "1899-12-30 00:00:00",
    "ROLES": ""
  } ],
  "ver": "4.00.514.10630",
  "sn": "01234567890123"
}
```

Response Properties

Property	Description		
clientID (string)	Temporary access token issued by SoftOne that verifies the Service is active and can be later used is Authentication method		
	Array containing the following keys		
objs (array)	COMPANY	integer	SoftOne Company ID
	COMPANYNAME	string	SoftOne Company Description
	BRANCH	string	SoftOne Branch ID
	BRANCHNAME	string	SoftOne Branch Description
	MODULE	integer	SoftOne Entity (12=Suppliers, 13=Customers, etc.)

	MODULENAME	string	SoftOne Entity Description
	REFID	integer	Rereference ID
	REFIDNAME	string	Reference Name
	USERID	integer	SoftOne User ID
	FINALDATE	datetime	User Expiration Date
	ROLES	string	User Roles
ver (string)	SoftOne Client version		
sn (string)	Serial Number of the SoftOne Account.		

Error Response Samples

1. Error Code -1: Invalid Credentials.

```
{
  "success": false,
  "errorcode": -1,
  "error": "Login fails due to invalid login credentials."
}
```

2. Error Code -3: Invalid appID, or Module not activated.

```
{
  "success": false,
  "errorcode": -3,
  "error": "Access denied. Selected module not activated."
}
```


Authenticate(Post Method)

Authenticates and gives access to the Web Application using the temporary access token (clientID) obtained through the login method. Returns the access token that is used in all the subsequent methods.

Request Sample

```
{
  "service": "authenticate",
  "clientID": "9J8p9J...2KqC0",
  "company": "1000",
  "branch": "1000",
  "module": "0",
  "refid": "1"
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (Authenticate).
clientID	string	Temporary access token obtained through the Login method
company	integer	Company obtained through the Login method
branch	integer	Branch obtained through the Login method
module	integer	Module obtained through the Login method
refid	integer	RefID obtained through the Login method

Response Sample

```
{
  "success": true,
  "clientID": " 9J8pH7...HL5L9GG ",
  "slu": 1,
  "hyperlinks": 1,
  "canexport": 1
}
```

Response Properties

Property	Type	Description
success	string	Validates user data and returns True or False
clientID	string	Access token issued by SoftOne that is used in all API requests later on
s1u	integer	SoftOne Internal Use Only
hyperlinks	integer	SoftOne Internal Use Only
Canexport	integer	SoftOne Internal Use Only

Error Response Samples

1. Error Code -1: Invalid ClientID.

```
{
  "success": false,
  "errorcode": -1,
  "error": "Invalid request. Please login first"
}
```

2. Error Code -2: Invalid Credentials, Company, Branch, Module or RefID.

```
{
  "success": false,
  "errorcode": -2,
  "error": "Authenticate fails due to invalid credentials."
}
```

ChangePassword (Post Method)

Changes the password of the login User.

Request Sample

```
{
  "service": "changepassword",
  "clientID": "9J8pH7...HL5L9GG",
  "appId": "3001",
  "old": "oldpassword",
  "new": "newpassword"
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (ChangePassword)
clientID	string	Unique client identifier obtained from the Authenticate Method
appId	integer	ID of the Service defined in SoftOne Web Accounts
old	string	Old Password
new	string	New Password

Response Sample

```
{
  "success": true
}
```

GetObjects (Post Method)

Returns all SoftOne objects (EditMaster) and tables in memory (EditList).

Request Sample

```
{
  "service": "getObjects",
  "clientId": "9J8pH7...HL5L9GG",
  "appId": "3001"
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (GetObjects)
clientId	string	Unique client identifier obtained from the Authenticate Method
appId	integer	ID of the Service defined in SoftOne Web Accounts

Response Sample

```
{
  "success": true,
  "count": 1327,
  "objects": [{
    "name": "ITEM",
    "type": "EditMaster",
    "caption": "Stock Items"
  },
  ...
  ,{
    "name": "$ITEGROUP",
    "type": "EditList",
    "caption": "Item Groups"
  },
  ...
  ,{
    "name": "CUSTOMER",
    "type": "EditMaster",
    "caption": "Customers"
  },
  ...
  ,{
    "name": "SALDOC",
    "type": "EditMaster",
    "caption": "Sales Documents"
  },
  ...
  ,{
    "name": "ZIP",
    "type": "EditMaster",
    "caption": "Postal codes"
  }
  ]
}
```

GetObjectTables (Post Method)

Returns all tables defined in a SoftOne object (EditMaster).

Request Sample

```
{
  "service": "getObjectTables",
  "clientID": "9J8pH7...HL5L9GG ",
  "appId": "3001",
  "object": "SALDOC"
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (GetObjectTables)
clientID	string	Unique client identifier obtained from the Authenticate Method
appld	integer	ID of the Service defined in SoftOne Web Accounts
object	string	SoftOne EditMaster Object Name

Response Sample

```
{
  "success": true,
  "count": 34,
  "tables": [{
    "name": "SALDOC",
    "dbname": "FINDOC",
    "caption": "Transactions (Sales)",
    "filltype": "SQL"
  }, {
    "name": "MTRDOC",
    "dbname": "MTRDOC",
    "caption": "Materials transactions",
    "filltype": "SQL"
  },
  ...
  {
    "name": "ITELINES",
    "dbname": "MTRLINES",
    "caption": "Item lines",
    "filltype": "SQL"
  },
  ...
  {
    "name": "SNLINES",
    "dbname": "SNLINES",
    "caption": "Serial number per material lines",
    "filltype": "SQL"
  }, {
    "name": "EXPANAL",
    "dbname": "EXPANAL",
    "caption": "Expense analysis",
    "filltype": "SQL"
  },
  ...
  ]
}
```

GetTableFields (Post Method)

Returns the table fields and properties of a SoftOne object (EditMaster).

Request Sample

```
{
  "service": "getTableFields ",
  "clientID": "9J8pH7...HL5L9GG",
  "appId": "3001",
  "object": "SALDOC",
  "table": "SALDOC"
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (GetTableFields)
clientID	string	Unique client identifier obtained from the Authenticate Method
appId	integer	ID of the Service defined in SoftOne Web Accounts
object	string	SoftOne EditMaster Object Name
table	string	Table (database name or alias name)

Response Sample

```
{
  "success": true,
  "count": 204,
  "fields": [
    ...{
      "name": "TRNDATE",
      "alias": "",
      "fullname": "SALDOC.TRNDATE",
      "caption": "Date",
      "size": "8",
      "type": "DateTime",
      "editttype": "CalendarDate",
      "defaultvalue": "'20150322'",
      "decimals": "",
      "editor": "",
      "readOnly": false,
      "visible": true,
      "required": true,
      "calculated": false
    },
    ...{
      "name": "TRDR",
      "alias": "",
      "fullname": "SALDOC.TRDR",
      "caption": "Customer",
      "size": "4",
      "type": "Integer",
      "editttype": "Selector",
      "defaultvalue": "",
      "decimals": "",
      "editor": "CUSTOMER",
      ...
    }, ...
  ]
}
```

GetDialog (Post Method)

Returns all filter fields from a specific browser or report dialog, together with their format.

Request Sample

```
{
  "service": "getDialog",
  "clientID": "9J8pH7...HL5L9GG",
  "appId": "3001",
  "object": "CUSTOMER",
  "list": ""
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (GetDialog)
clientID	string	Unique client identifier obtained from the Authenticate Method
appId	integer	ID of the Service defined in SoftOne Web Accounts
object	string	SoftOne EditMaster Object Name
list	string	List Name.

Response Sample

```
{
  "success": true,
  "title": "Customers",
  "model": {
    "CUSTOMER": {
      "relationship": "OneToOne",
      "editable": true,
      "fields": [{
        "name": "CODE",
        "type": "string"
      }, {
        "name": "CODE_TO",
        "type": "string"
      }],
    }
  },
  "form": [{
    "xtype": "slcont",
    "items": [{
      "xtype": "slfields",
      "flex": 0.50,
      "group": 1,
      "items": [
        slEditor("0;CUSTOMER.CODE;Code,from;;;1;0;1;0;0;;;")
      ]
    }, {
      ...
    }
  ]},
  "editable": true
}]
}
```

GetFormDesign (Post Method)

Returns all fields and design properties from a specific form of an object.

Request Sample

```
{
  "service": "getFormDesign",
  "clientID": "9J8pH7...HL5L9GG",
  "appId": "3001",
  "object": "CUSTOMER",
  "form": ""
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (GetFormDesign)
clientID	string	Unique client identifier obtained from the Authenticate Method
appId	integer	ID of the Service defined in SoftOne Web Accounts
object	string	SoftOne EditMaster Object Name
form	string	Form Name.

Response Sample

```
{
  "success": true,
  "model": {
    "CUSTOMER": {
      "relationship": "OneToOne",
      "editable": true,
      "fields": [{
        "name": "CODE",
        "type": "string"
      }],
      ...
    },
    ...
  },
  "form": [{
    "xtype": "slcont",
    "items": [{
      "xtype": "slfields",
      "flex": 0.25,
      "group": 110,
      "items": [
        slEditor("0;CUSTOMER.CODE;Code;;;1;0;1;1;0;;;")
      ]
    }, ...
    "items": [{
      "xtype": "slcont",
      "items": [{
        "xtype": "slfields",
        ...
      ]
    }, ...
  ]
}
```

GetBrowserInfo(Post Method)

Executes a browser and returns its reference code (reqID) along with the structure of the browser fields, the columns of the browser, the number of records and the total sums of the numeric fields.

Response field **"ZOOMINFO"** indicates the id of the main table of the object.

Request Sample

```
{
  "service": "getBrowserInfo",
  "clientID": "9J8pH7...HL5L9GG",
  "appId": "3001",
  "object": "SALDOC",
  "list": "",
  "filters": "SALDOC.TRNDATE=2015-03-01&SALDOC.TRNDATE_TO=2015-03-31&SALDOC.FINCODE=sis*"
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (GetBrowserInfo)
clientID	string	Unique client identifier obtained from the Authenticate Method
appId	integer	ID of the Service defined in SoftOne Web Accounts
object	string	SoftOne EditMaster Object Name
list	string	List Name
filters	string	<p>Operation is always "=", you can not have "<>" and the syntax is the following: FieldName1=FilterValue1&FieldName2=FilterValue2&...&FieldNameX=FilterValueX</p> <p>Filter Fields are the same as the ones found in Browser Filters Screen (using CTRL + SHIFT + F12). Note that the format for filtering datetime fields is "yyyy-dd-dd". If you set filters in the FROM field then it will be translated as "=". If you set filters in both FROM and TO fields then it will be translated as ">=" (DATEFROM) and "<" (DATETO-1)</p>

Response Properties

Property	Type	Description
success	string	Validates user data and returns True or False
formDesign	string	Access token issued by SoftOne that is used in all API requests later on
reqID	String	Browser reference code
totalcount	Integer	Total number of records
fields	Array	Field Names and types
columns	Array	Field labels and parameters as defined in SoftOne browser
sums	Array	Totals for fields with numeric values

Response Sample

```

{
  "success": true,
  "formDesign": "SALDOC",
  "reqID": "0018374974105073328",
  "totalcount": 5,
  "fields": [{
    "name": "ZOOMINFO",
    "type": "string"
  }, {
    "name": "SALDOC.TRNDATE",
    "type": "date"
  }, {
    "name": "SALDOC.FINCODE",
    "type": "string"
  },
  ...{
    "name": "SALDOC.SUMAMNT",
    "type": "float"
  }
],
  "columns": [{
    "dataIndex": "SALDOC.TRNDATE",
    "header": "Date",
    "width": 120,
    "sortable": true
  }, {
    "dataIndex": "SALDOC.FINCODE",
    "header": "Document",
    "width": 120,
    "sortable": true
  },
  ...{
    "dataIndex": "SALDOC.SUMAMNT",
    "header": "Collection Amount",
    "width": 168,
    "sortable": true,
    "align": "right"
  }
],
  "sums": [
    "",
    "",
    "",
    "",
    "89531.06"
  ]
}

```

GetBrowserData(Post Method)

Returns data from a browser with a specific reference ID. Records can be limited using limit property.

Request Sample

```
{
  "service": "getBrowserData",
  "clientID": "9J8pH7...HL5L9GG",
  "appId": 3001,
  "reqID": "0018374974105073328",
  "start": 0,
  "limit": 2
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (GetBrowserData)
clientID	string	Unique client identifier obtained from the Authenticate Method
appId	integer	ID of the Service defined in SoftOne Web Accounts
reqID	string	Referece ID obtained through GetBrowserInfo
start	integer	Start Record (zero based, 0 is the first record)
limit	integer	Number of returned records. Note that totalcount property returns the total number of records.

Response Sample

```
{
  "success": true,
  "totalcount": 5,
  "rows": [
    [
      "01351;594",
      "2015-03-29 00:00:00",
      "SISN000019",
      "219",
      "Customer 1",
      "62115"
    ],
    [
      "01351;593",
      "2015-03-21 00:00:00",
      "SISN000018",
      "155",
      "Customer 2",
      "8074.95"
    ]
  ]
}
```

GetReportInfo(Post Method)

Executes a SoftOne report and returns its reference code (reqID) along with the number of the pages.

Request Sample

```
{
  "service": "getReportInfo",
  "clientID": "9J8pH7...HL5L9GG",
  "appId": 3001,
  "object": "CUST_STM",
  "list": "",
  "filters": "CUSTOMER.CODE=102*"
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (GetReportInfo)
clientID	string	Unique client identifier obtained from the Authenticate Method
appId	integer	ID of the Service defined in SoftOne Web Accounts
object	string	SoftOne Report Object Name
list	string	Report Name
filters	string	Filters syntax is the following: FieldName1=FilterValue1&FieldName2=FilterValue2&...&FieldNameX=FilterValueX

Response Sample

```
{
  "success": true,
  "reqID": "0881579405109468171",
  "npages": 1
}
```

GetReportData(Post Method)

Returns a page of a report in HTML format. The report is defined by the specific reference ID obtained through GetReportInfo.

Request Sample

```
{
  "service": "getReportData",
  "clientID": "9J8pH7...HL5L9GG",
  "appId": 3001,
  "reqID": "0881579405109468171",
  "pagenum": 1
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (GetReportData)
clientID	string	Unique client identifier obtained from the Authenticate Method
appId	integer	ID of the Service defined in SoftOne Web Accounts
reqID	string	Referece ID obtained through GetReportInfo
pagenum	integer	Page number that will be returned

Response Sample

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN">
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=windows-1253" />
    <title>Softone Report</title>
    <link rel="stylesheet" type="text/css"
href="https://slsites01.blob.core.windows.net/lib/weblib/prints/prints.css"/>
    <script type="text/javascript"
src="https://slsites01.blob.core.windows.net/lib/weblib/prints/prints.js"></script>
  </head>
  <body>
    <table cellpadding="0" cellspacing="0" class="outtertable">
      <tr class="header">
        <td align="center" width="38px">A/A</td>
        <td align="center" width="75px">Code</td>
        <td align="center" width="262px">Name</td>
        <td align="center" width="150px">T.R.No.</td>
      </tr>
      ...
      <td width="105px" align="right">152,396.80</td>
      <td width="105px" align="right">40,416.65</td>
      <td width="105px" align="right">&nbsp;</td>
      </tr>
      <tr class="footer">
        <td colspan="4" align="left">Totals</td>
        <td width="105px" align="right">128,522.70</td>
        <td width="105px" align="right">95,700.00</td>
        <td width="105px">&nbsp;</td>
        <td width="105px">&nbsp;</td>
        <td width="105px">&nbsp;</td>
        <td width="105px">&nbsp;</td>
      </tr>
    </table>
  </td></table>
</body>
```

GetData(Post Method)

Returns the data of a record of a Business Object. The LOCATEINFO parameter defines the fields that will be returned; leave it null to return all tables and fields.

Request Sample

```
{
  "service": "getData",
  "clientID": "9J8pH7...HL5L9GG",
  "appId": "3001",
  "object": "CUSTOMER",
  "form": "",
  "key": 47,
  "locateinfo": "CUSTOMER:CODE,NAME,AFM;CUSTEXTRA:VARCHAR02,DATE01"
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (GetData)
clientID	string	Unique client identifier obtained from the Authenticate Method
appId	integer	ID of the Service defined in SoftOne Web Accounts
object	string	SoftOne Object Name
form	string	Form Name
key	integer	Primary key of the record
locateinfo	string	Return fields in the following syntax: TableName1:FieldName1,FieldName2;TableName2:FieldName1,FieldName2;...

Response Sample

```
{
  "success": true,
  "readOnly": false,
  "data": {
    "CUSTOMER": [{
      "CODE": "107",
      "NAME": "SoftOne AE",
      "AFM": "999863881"
    }]
  },
  "prtname": "",
  "caption": "107 - SoftOne AE",
  "calc": false,
  "einvoice": false
}
```

FileName(Get Method)

Returns the file of a SoftOne Business Object record. The full name of the file is returned from the getData method. Photos saved inside Business records can be retrieved through the Field SODATA of the alias table of XTRDOCDATA (LNUM=0, RefObjID=ObjectRecordID).

URLs or Dropbox files that are saved in Related Files of a record can be retrieved using the field SOFNAME of the table XTRDOCDATA (LNUM>0).

Request URL

[https://webdemo.oncloud.gr/s1services?FileName=\[serial_number\]/EA78807CFF1D36DB.jpg](https://webdemo.oncloud.gr/s1services?FileName=[serial_number]/EA78807CFF1D36DB.jpg)

Example

Download the photo saved in an item's record (ID=1634)

Step 1 – Use getData to get the filename (Alias Table of XTRDOCDATA is ITEDOCDATA)

Request Sample

```
{
  "service": "getData",
  "clientId": "9J8pH7...HL5L9GG",
  "appId": "3001",
  "OBJECT": "ITEM",
  "FORM": "",
  "KEY": 1634,
  "LOCATEINFO": "ITEDOCDATA:SODATA"
}
```

Response

```
{
  "success": true,
  "readOnly": false,
  "data": {
    "ITEDOCDATA": [
      {
        "SODATA": "12345678901234/EA78807CFF1D36DB.jpg"
      }
    ]
  },
  "prtname": "",
  "caption": "10014 - Rescue Life Jackets",
  "calc": false,
  "einvoice": false
}
```

Step 2– Use FileName get method to display the file

Request URL

<https://webdemo.oncloud.gr/s1services?FileName=12345678901234/EA78807CFF1D36DB.jpg>

SetData(Post Method)

Inserts or modifies the data of a SoftOne business object (Customers, Items, Sales, etc.).
If the property "KEY" is empty or missing then it inserts a new record, else it locates the record using the given number and modifies its data.
In order to add or update data to child (detail) tables you also need to define the field LINENUM. For adding new records use numbers from 9000001 and up. Notice that if the child table contains records already, and you need to update or add new, you have to insert the LINENUM field number of the existing ones (without using any other fields), or else those records will be deleted. Child table records can be updated by entering the LINENUM field and then the fields that will be updated.

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (SetData)
clientID	string	Unique client identifier obtained from the Authenticate Method
appld	integer	ID of the Service defined in SoftOne Web Accounts
OBJECT	string	SoftOne Object Name
OBJECTPARAMS	JSON array	Object parameters. e.g. {"param1":1, "param2":"test"}
key	integer	Primary key of the record that you need to modify. Leave it blank if you want to insert a new record.
data	array of tables	Array containing the fields of the tables that will be modified or inserted
		TABLE 1 (array of table fields) <div>FIELD 1 FIELD 2 ...</div>
		TABLE 2 (array of table fields) <div>FIELD 1 FIELD 2 ...</div>

Response Sample

```
{
  "success": true,
  "id": "47"
}
```

Request Example

Modify Customers Record. Update Customer fields: Name, Address, T.R.No, City, Phone, Remarks.

Insert also data for fields Varchar01 and Int01 of the extra table CusExtra.

Add a new customer branch with Code=111 and modify the branch name of the record with Linenum=2.

Make sure that the customer branch with linenum=1 is not deleted.

```
{
  "service": "setData",
  "clientID": "9J8pH7...HL5L9GG",
  "appId": "3001",
  "object": "CUSTOMER",
  "objectparams": {"param1": 0},
  "key": "47",
  "data": {
    "CUSTOMER": [{
      "NAME": "New Name",
      "ADDRESS": "New Address",
      "AFM": "0123456789",
      "CITY": "Athens",
      "PHONE01": "0302102102102",
      "ZIP": "12345",
    }
  ],
  "CUSEXTRA": [{
    "VARCHAR01": "Extra 1",
    "INT01": 1
  }
  ],
  "CUSBRANCH": [{
    "LINENUM": 1
  }, {
    "LINENUM": 2
    "NAME": "UpdatedBranchName"
  }, {
    "LINENUM": 9000001,
    "CODE": 111,
    "NAME": "NewBranchName"
  }
  ]
}
```


Calculate(Post Method)

Returns the calculated data based on the user input, such as: item prices in documents, total amount of documents (as calculated from SoftOne client), displays the alert messages, updates fields based on other field editors ([U attribute](#)). The data and the fields returned are defined in property **"LOCATEINFO"**. It doesn't cause any modifications to the record, only calculations.

Request Sample

```
{
  "service": "calculate",
  "clientId": "Wj8T3tvs... ..tlrT8",
  "appId": "3001",
  "object": "CUSTOMER",
  "key": "47",
  "locateinfo": "CUSTOMER:CODE,NAME,AFM;CUSTEXTRA:VARCHAR02,DATE01"
  "data": {
    "CUSTOMER": [
      {
        "CODE": "100",
        "NAME": "Soft One Technologies S.A.",
        "AFM": "999863881"
      }
    ],
    "CUSEXTRA": [
      {
        "VARCHAR01": "Extra 1",
        "VARCHAR02": "Extra 2"
      }
    ]
  }
}
```

Request Parameters

Property	Type	Description				
service	string	Defines the name of the method that will be executed (SetData)				
clientID	string	Unique client identifier obtained from the Authenticate Method				
appld	integer	ID of the Service defined in SoftOne Web Accounts				
object	string	SoftOne Object Name				
key	integer	Primary key of the record that will be used in calculation.				
locateinfo	string	Tables separated by ; and fields separated by commas. Array containing the tables and their fields that will be used in calculation				
data (array of tables)	<table><tr><td rowspan="3">TABLE 1 (array of table fields)</td><td>FIELD 1</td></tr><tr><td>FIELD 2</td></tr><tr><td>...</td></tr></table>		TABLE 1 (array of table fields)	FIELD 1	FIELD 2	...
TABLE 1 (array of table fields)	FIELD 1					
	FIELD 2					
	...					

Response Sample

```
{
  "success": true,
  "readOnly": true,
  "data": {
    "CUSTOMER": [{
      "CODE": "100",
      "NAME": "Soft One Technologies S.A.",...
    }]
  }
}
```

DelData(Post Method)

Deletes the record of a SoftOne business object that is defined by the property **"key"**.

Request Sample

```
{
  "service": "delData",
  "clientID": "Wj8T3tvs... ..tlrT8",
  "appId": "3001",
  "object": "CUSTOMER",
  "form": "",
  "key": 47
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (SetData)
clientID	string	Unique client identifier obtained from the Authenticate Method
appId	integer	ID of the Service defined in SoftOne Web Accounts
object	string	SoftOne Object Name
form	string	Form Name
key	integer	Primary key of the record that will be deleted.

Response Sample

```
{
  "success": true
}
```

GetSelectorData(Post Method)

Returns the data of selector lists, memory tables and string lists (data control, see [C.7 Selector List](#)). The fields returned are the ones defined in the "[selector fields](#)" property of the table inside the database designer.

Request Sample

```
{
  "service": "getSelectorData",
  "clientId": "Wj8T3tvs... ..tlrT8",
  "appId": "3001",
  "editor": "1|CUSTOMER||ISPROSP=0 AND ISACTIVE=1|",
  "value": "100*"
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (SetData)
clientId	string	Unique client identifier obtained from the Authenticate Method
appId	integer	ID of the Service defined in SoftOne Web Accounts
editor	string	SoftOne Internal use only
value	string	Filters the data that will be returned. The filter is applied on the first column of the selector editor, as defined in the "selector fields" tab inside database designer.

Response Sample

```
{
  "success": true,
  "data": [
    [
      "40",
      "1",
      "100",
      "Demo Client AE",
      "999863881",
      ""
    ],
    [
      "365",
      "1",
      "1000",
      "100454656",
      "",
      ""
    ]
  ]
}
```

SelectorFields(Post Method)

Returns the data of a table for the fields specified by "**resultfields**" parameter. Filters are added using the properties "**keyname**" and "**keyvalue**".

Request Sample

```
{
  "service": "selectorFields",
  "clientId": "Wj8T3tvs... ..tlrT8",
  "appId": "3001",
  "tablename": "CUSTOMER",
  "keyname": "TRDR",
  "keyvalue": 47,
  "resultfields": "CODE,NAME,AFM"
}
```

Request Parameters

Property	Type	Description
service	string	Defines the name of the method that will be executed (SetData)
clientId	string	Unique client identifier obtained from the Authenticate Method
appId	integer	ID of the Service defined in SoftOne Web Accounts
tablename	string	SoftOne Table Name
keyname	string	Field Names that will be used to filter the records
keyvalue	integer	Values of the fields that will be used to filter the records
resultfields	string	Fields that will be returned

Response Sample

```
{
  "success": true,
  "totalcount": 1,
  "rows": [
    {
      "CODE": "100",
      "NAME": "Soft One Technologies S.A.",
      "AFM": "999863881"
    }
  ]
}
```

D. Case Studies

All the following examples require Login and Authentication. Use the site registered to SoftOne to do all the web service calls: <https://RegisteredName.oncloud.gr/s1services>

1. Get Items

Retrieve items as they appear in a SoftOne browser, matching specified filters (Code=20*, Item Group=106 or 107).

Step 1 – Create browser in SoftOne

Inside SoftOne client create a browser in "Items" entity, add the fields you need to retrieve and save it using any name (e.g. WSItems).

Step 2 – Get browser reqID

Use *getBrowserInfo* (post method) to get the reference code of the browser you created in the previous step. The object name is *ITEM* and list is name is the one you used to save your browser (WSItems). Add criteria using the *filters* parameter.

Request Sample

```
{
  "service": "getBrowserInfo",
  "clientID": "9J8pH7...HL5L9GG",
  "appId": "3001",
  "object": "ITEM",
  "list": "WSItems",
  "filters": "ITEM.CODE=20*&ITEM.MTRGROUP=106,107"
}
```

Response Sample

```
{
  "success": true,
  "formDesign": "ITEM",
  "reqID": "0911051864780364875",
  "totalcount": 3,
  "fields": [{
    ...
  },
  ...
]
```

Step 3 – Get Items data

Use *getBrowserData* (post method), combined with the reqid given from the response of the previous step, to retrieve customers data. Records can be limited using the *limit* parameter.

Request Sample

```
{
  "service": "getBrowserData",
  "clientID": "9J8pH7...HL5L9GG",
  "appId": 3001,
  "reqID": "0911051864780364875",
  "start": 0,
  "limit": 50
}
```

Response Sample

```
{
  "success": true,
  "totalcount": 3,
  "rows": [
    [
      "ITEM;1653",
      "1",
      "20014",
      "Maritime Compass",
      "Piec",
      "444.6",
      "461.7"
    ],
    [
      "ITEM;1655",
      "2",
      "20016",
      "Gyro compass",
      "Piec",
      "629.2",
      "653.4"
    ],
    [
      "ITEM;1656",
      "3",
      "20017",
      "Altimeter aviation",
      "Piec",
      "697.84",
      "724.68"
    ]
  ]
}
```

2. Add Customer

Insert a new customer in SoftOne.

Use **setData** (post method) to add a new customer, giving a null value to the parameter "**key**". The object name is *Customer*. Add data to tables that exist in Customer object by using their name and fields.

Request Sample

```
{
  "service": "setData",
  "clientID": "9J8...NM44",
  "appId": "157",
  "object": "CUSTOMER",
  "key": "",
  "data": {
    "CUSTOMER": [{
      "CODE": "123456",
      "NAME": "New Name",
      "ADDRESS": "New Address",
      "REMARKS": "My Remarks!"
    }],
    "CUSEXTRA": [{
      "VARCHAR01": "Extra 1",
      "INT01": 1
    }]
  ]
}
```

Response Sample

```
{
  "success": true,
  "id": "1450",
  "message": "Posted Customer: New Name\n(From Advanced Javascript)"
}
```

The screenshot displays the 'Customers' module in the SoftOne application. The main window shows a form for a new customer with the following details:

- Code:** 123456
- Name:** New Name
- Active:** ☒ Yes
- T.R.No.:** (empty)
- Branch:** 1000 Head Offices
- is Prospect:** ☐ No
- Alternative Code:** (empty)
- Reference code:** (empty)
- is Competitor:** ☐ No

Below the form, there are two sections: 'Text' and 'Tables'.

Text	Tables
Text 1: Extra 1	Table 1: (dropdown)
Text 2: (empty)	Table 2: (dropdown)
Text 3: (empty)	Table 3: (dropdown)
Text 4: (empty)	Table 4: (dropdown)
Text 5: (empty)	Table 5: (dropdown)

The interface includes a top navigation bar with 'List', 'New', 'Save', 'Delete', and other icons. A sidebar on the right contains links for 'Jobs', 'Browsers', 'Screen Forms', and 'Tools'.

Figure CS2.1 – New customer added from web service

3. Update Items

Update the description of the item with id 1634.

Use `setData` (post method) to update item's data. Enter the id of the record in the parameter "**key**".
Object name is *ITEM*.

Request Sample

```
{
  "service": "setData",
  "clientID": "938...NM44",
  "appId": "157",
  "object": "ITEM",
  "key": "1634",
  "data": {
    "ITEM": [{
      "NAME": "Description updated from web service"
    }]
  }
}
```

Response Sample

```
{
  "success": true,
  "id": "1634",
  "message": "From Advanced JavaScript - Works in All Forms"
}
```

The screenshot shows a web application interface for 'Stock Items'. The main title is '10014- Description updated from web service'. Below the title, there are input fields for 'Code' (10014), 'Description' (Description updated from web service), and 'Active' (checked Yes). The interface is divided into several sections:

- General data**: Includes tabs for 'UoM, Alternative Codes', 'Trade', 'Replenishment/Supply', 'Accounting', 'Tracking & Variation Management', 'Data per Company', and 'User-defi'.
- Fast entry**: Contains fields for 'Base UoM' (101), 'VAT Group' (1310), 'Accounting category' (101), 'BarCode' (241.46.3.023), 'Factory code' (241.46.3.023), and 'Technical code'.
- Grouping/ Classification**: Contains fields for 'Commercial category' (101), 'Item Group' (106), 'Business unit' (104), 'Correlation Item', 'Season', and 'Commission category'.
- Photo**: Displays a small image of a red and black backpack.
- Additional data**: Contains fields for 'Description 2', 'Web page', 'Invoice Warning', 'Replaced by', and 'Exclusively for'.
- Allocation data**: Contains fields for 'Manufacturer', 'Brand', 'Model', 'Country of origin', and 'Made in'.
- Prices & Discounts**: Contains fields for 'Wholesale Price' (75.00), 'Retail Price' (90.00), and three discount percentage fields.

Figure CS3.1 – Item description updated from web service

4. Add Customer Branch

Add a new customer branch inside the customer with id 100. Make sure that branches with LINENUM 1 and 2 are not deleted.

Use `setData` (post method) to locate a customer record and add a new branch. Notice that the new branch is added using LINENUM 9000001. Branches with LINENUM 1 and 2 are also defined, in order not to be deleted.

Request Sample

```
{
  "service": "setData",
  "clientID": "9J8p...M44",
  "appId": "157",
  "object": "CUSTOMER",
  "key": "47",
  "data": {
    "CUSBRANCH": [{
      "LINENUM": 1
    }, {
      "LINENUM": 2
    }, {
      "LINENUM": 9000001,
      "CODE": "003",
      "NAME": "New Branch Name"
    }
  ]
}
```

Response Sample

```
{
  "success": true,
  "id": "47",
  "message": "Posted Customer: SoftOne AE\n(From Advanced Javascript)"
}
```

The screenshot shows a web application for managing customers and their branches. The main header is 'Customers'. Below it, the customer '107- SoftOne AE' is displayed with various fields: Code (107), Name (SoftOne AE), Active (Yes), T.R.No. (99986), Branch (1000 Head Offices), is Prospect (No), Alternative Code, Reference code, and is Competitor (No). The 'Associated Companies, Branches' tab is selected, showing a table of branches. The table has columns for Code, Name, Installation No., and Address. Three branches are listed: Branch 1 (Code 1 001), Branch 2 (Code 2 002), and New Branch Name (Code 3 003). The 'Is also a supplier' checkbox is unchecked.

Code	Name	Installation No.	Address
1 001	Branch 1		
2 002	Branch 2		
3 003	New Branch Name		

Figure CS4.1 – New Branch added without deleting Branches 001 and 002

5. Get Sales Documents

Retrieve Sales Documents as they appear in a SoftOne browser, matching specified fields

Step 1 – Create browser in SoftOne

Inside SoftOne client create a browser in "Sales Documents" entity, add the fields you need to retrieve and save it using any name (suppose it's WSSales).

Step 2 – Get browser reqID

Use **getBrowserInfo** (post method) to get the reference code of the browser you created in the previous step. The object name is *SALDOC* and the list name is the one you used to save your browser (WSSales). Add criteria using the *filters* parameter.

Request Sample

```
{
  "service": "getBrowserInfo",
  "clientId": "9J8pH7...HL5L9GG",
  "appId": "3001",
  "object": "SALDOC",
  "list": "WSSales"
  "filters": ""
}
```

Step 3 – Get sales documents data

Use **getBrowserData** (post method), combined with the reqid given from the response of the previous step, to retrieve sales documents. Records can be limited using the *limit* parameter.

Request Sample

```
{
  "service": "getBrowserData",
  "clientId": "9J8pH7...HL5L9GG",
  "appId": 3001,
  "reqID": "0018374974105073328",
  "start": 0,
  "limit": 50
}
```

Response Sample

```
{
  "success": true,
  "totalcount": 31,
  "rows": [
    [
      "01351;609",
      "1",
      "2015-04-05 00:00:00",
      "SINV000005",
      "Sales Invoice",
      "Head Offices",
      "121",
      "Alexis Chalkidis",
      "369"
    ],
    [
      "01351;596",
      "3",
      "2015-04-05 00:00:00",
      "SISN000021",
      ...
    ],
    ...
  ], ...
}
```

6. Add Sales Document

Insert a new Sales Document with 2 item lines.

Use **setData** (post method) to add a new sales document, giving a null value to the parameter "**key**". The object name is **SALDOC**. Table name for item lines is **ITELINES**. Table name for service lines is **SRVLINES**. For each line added don't forget to add data to the field **LINENUM** using numbers greater than 9000000.

Request Sample

```
{
  "CLIENTID": "9J8pH7...HL5L9GG ",
  "APPID": 3001,
  "SERVICE": "SetData",
  "OBJECT": "SALDOC",
  "KEY": "",
  "FORM": "",
  "DATA": {
    "SALDOC": [
      {
        "SERIES": "7001",
        "TRDR": "40",
        "PAYMENT": "1003"
      }
    ],
    "ITELINES": [
      {
        "LINENUM": 9000002,
        "VAT": "1310",
        "MTRUNIT4": "101",
        "MTRCATEGORY": "204",
        "MTRL": 2016,
        "QTY1": 2,
        "PRICE": 5
      },
      {
        "LINENUM": 9000001,
        "VAT": "1310",
        "MTRUNIT4": "101",
        "MTRCATEGORY": "204",
        "MTRL": 2015,
        "QTY1": 1,
        "PRICE": 10
      }
    ],
    "SRVLINES": []
  }
}
```

Response Sample

```
{
  "success": true,
  "id": "919"
}
```

7. Add HTML data

Custom Web Service, created in Advanced JavaScript, that locates a SoftOne Object record (e.g. SOEMAIL) and adds HTML data to a blob field (e.g. SOMAIL.SOBODY).

```
function AddHTMLData(obj) {
    var resp = {};
    resp.success = false;
    if ((obj.clientID) && (obj.clientID != "")) {
        try {
            var SIModule = X.EXEC('CODE:ModuleIntf.CreateModule', obj.ObjectName);
            X.EXEC('CODE:ModuleIntf.LocateModule', SIModule, obj.RecordID);
            var vdsTable = X.EXEC('CODE:ModuleIntf.GetDataSet', SIModule, obj.TableName);
            var vBlobField = X.EXEC('CODE:ModuleIntf.GetField', vdsTable, obj.FieldName);
            var vBlobPointer = X.EXEC('CODE:SOHTMLDOC.CreateDoc', 1, obj.HTMLCode);
            X.EXEC('CODE:ModuleIntf.DatasetEdit', vdsTable);
            X.EXEC('CODE:SOHTMLDOC.SaveDocToBlob', vBlobPointer, vBlobField);
            X.EXEC('CODE:ModuleIntf.DatasetPost', vdsTable);
            X.EXEC('CODE:ModuleIntf.PostModule', SIModule);
            X.EXEC('CODE:SOHTMLDOC.DestroyDoc', vBlobPointer);
            X.EXEC('CODE:ModuleIntf.DestroyModule', SIModule);
            resp.success = true;
        }
        catch (e) {
            resp.error = e.message;
        }
    }
    else {
        resp.error = "Authenticate failed due to invalid credentials!";
    }
    return resp;
}
```

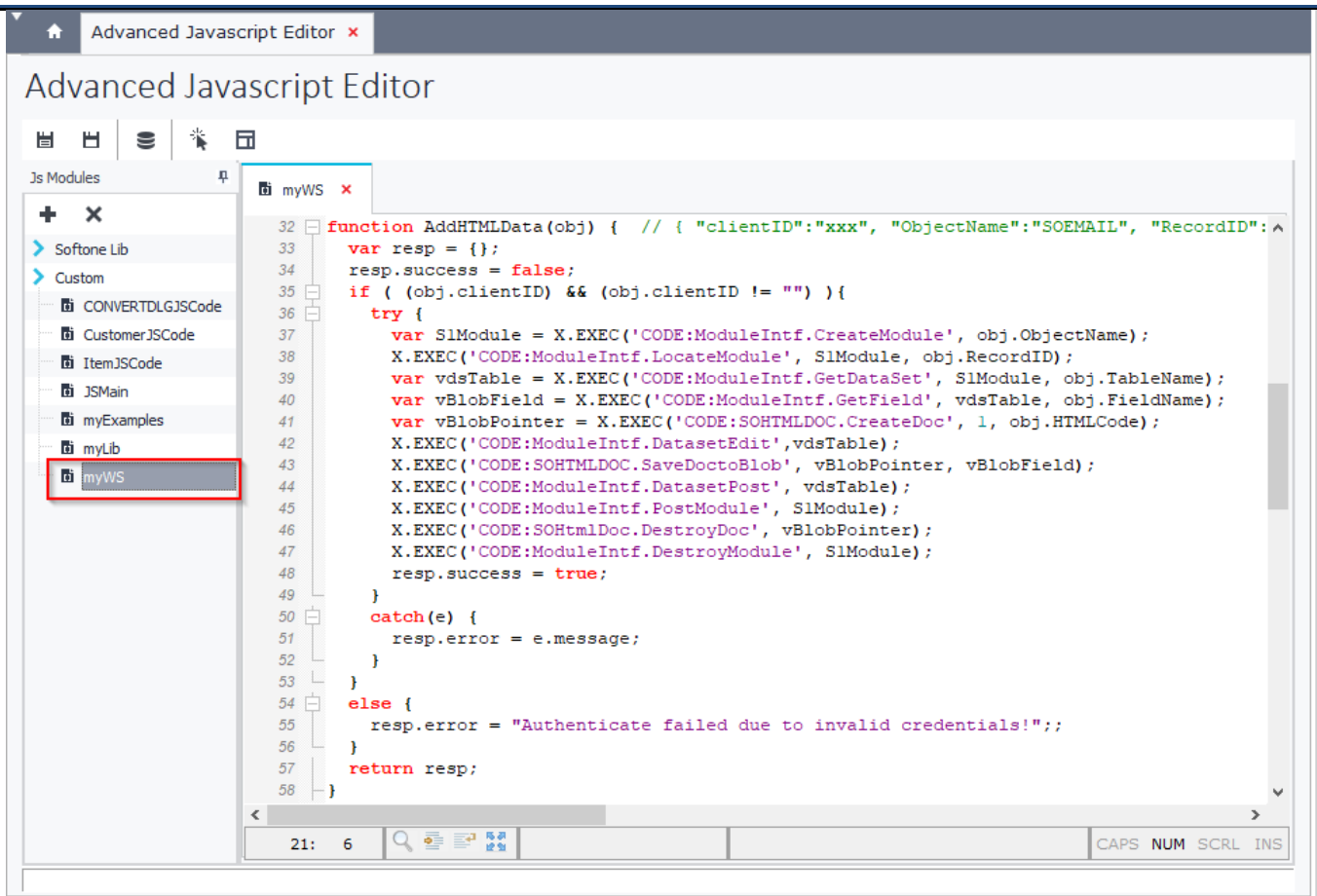


Figure CS7.1 – Advanced JavaScript – Custom WebService

Request URL

```
https://RegisteredName.oncloud.gr/s1services/JS/myWS/AddHTMLData
```

Request Sample

```
{
  "clientID": "9J8...M44",
  "ObjectName": "SOEMAIL",
  "RecordID": 3362,
  "TableName": "SOMAIL",
  "FieldName": "SOBODY",
  "HTMLCode": "<p>This is a <span style='color: #0000ff;'>test</span> email from <span style='color: #ff0000;'><strong>web service call</strong></span>.</p>"
}
```

Response Sample

```
{
  "success": true
}
```

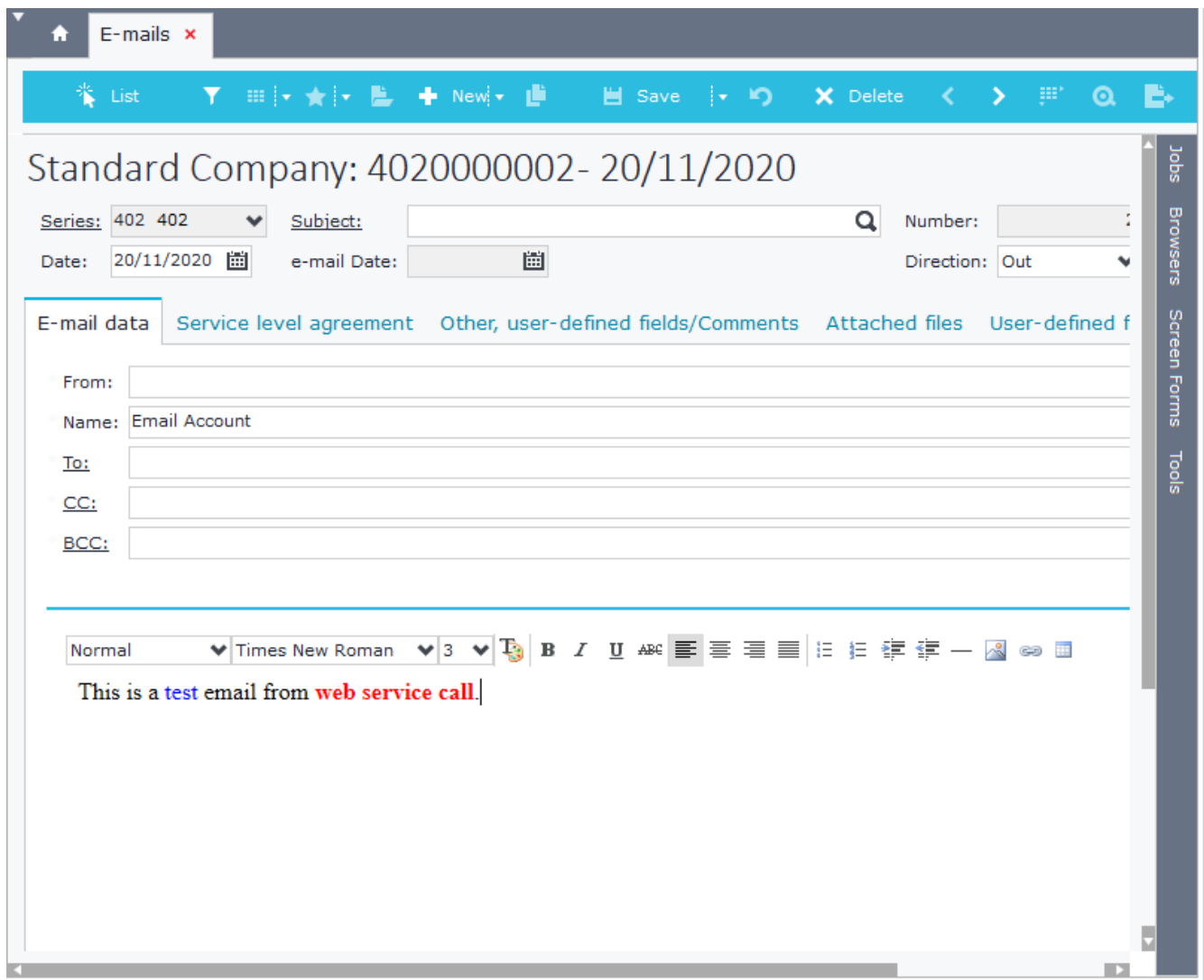


Figure CS7.2 – HTML data posted from web service call

8. Get HTML data

Custom Web Service, created in Advanced JavaScript, that locates a SoftOne Object record (e.g. SOEMAIL) and retrieves HTML data from a blob field (e.g. SOMAIL.SOBODY).

```
function GetHTMLData(obj) {
    var resp = {};
    resp.success = false;
    resp.data = "";
    if ((obj.clientID) && (obj.clientID != "")) {
        try {
            var SIModule = X.EXEC('CODE:ModuleIntf.CreateModule', obj.ObjectName);
            X.EXEC('CODE:ModuleIntf.LocateModule', SIModule, obj.RecordID);
            var vdsTable = X.EXEC('CODE:ModuleIntf.GetDataSet', SIModule, obj.TableName);
            var vBlobField = X.EXEC('CODE:ModuleIntf.GetField', vdsTable, obj.FieldName);
            var vBlobPointer = X.EXEC('CODE:SOHTMLDOC.CreateDoc', 2, vBlobField);
            resp.data = X.EXEC('CODE:SOHTMLDOC.GetDocHTMLPart', vBlobPointer);
            X.EXEC('CODE:SOHTMLDOC.DestroyDoc', vBlobPointer);
            X.EXEC('CODE:ModuleIntf.DestroyModule', SIModule);
            resp.success = true;
        }
        catch (e) {
            resp.error = e.message;
        }
    }
    else {
        resp.error = "Authenticate failed due to invalid credentials!";
    }
    return resp;
}
```

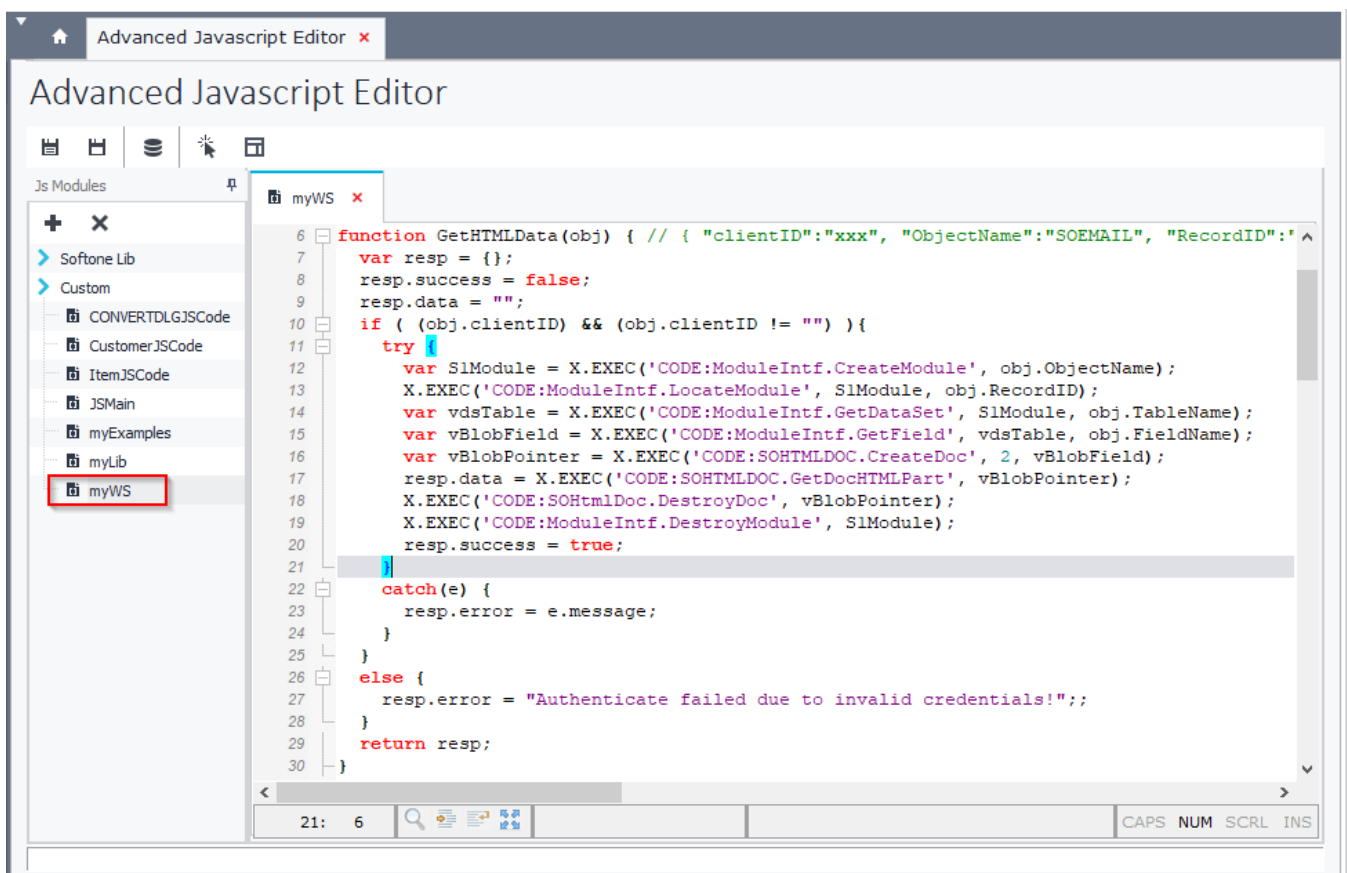


Figure CS8.1 – Advanced Javascript – Custom WebService

Request URL

```
https://RegisteredName.oncloud.gr/s1services/JS/myWS/GetHTMLData
```

Request Sample

```
{
  "clientID": "9J8...M44",
  "ObjectName": "SOEMAIL",
  "RecordID": 3363,
  "TableName": "SOMAIL",
  "FieldName": "SOBODY"
}
```

Response Sample

```
{
  "success": true,
  "data": "<html><head></head><body><p>Hello,</p><p>this is a test email.</p><p><img align=\"baseline\" alt=\"\" src=\"https://www.softone.gr/wp-content/themes/Softone/images/logo-mobile.png\" border=\"0\" hspace=\"0\"></p></body></html>"
}
```

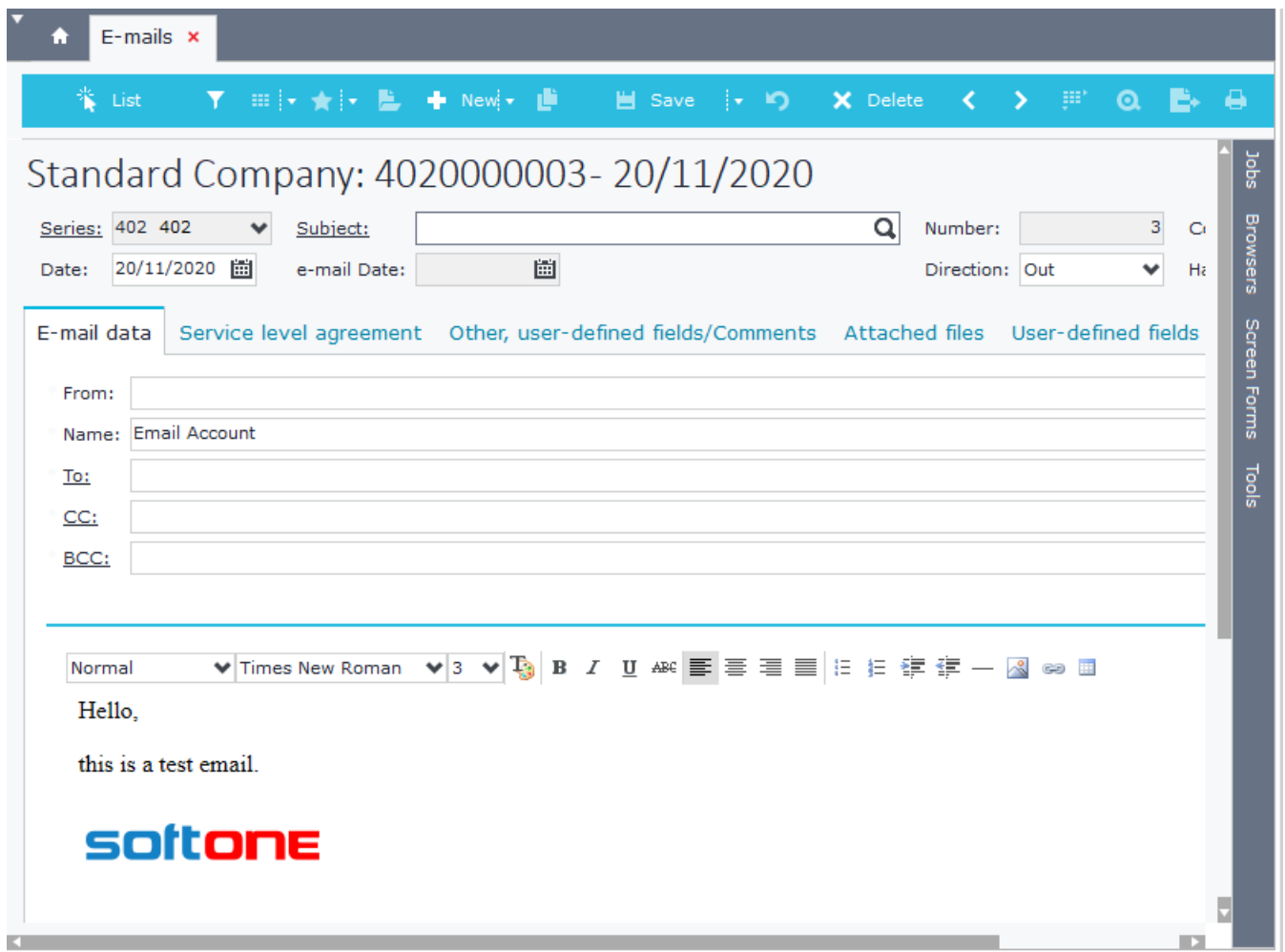


Figure CS8.2 – HTML data retrieved from GetHTMLData web service

9. Add Related Files

Locate a record and add related documents using the web service method [SetData](#).

Notice that some SoftOne objects, like SALDOC (Sales Documents) link to Related Documents using Relative Jobs. In these cases, you need to manually create a custom form, add the table XTRDOCDATA "Related Documents" and make the webservice call for this specific form.

Use LINENUM field as in every detail table added from web service call (e.g. [Add Customer Branch](#)).

Request Sample

```
{
  "service": "setData",
  "clientId": "9J8...M44",
  "appId": "157",
  "OBJECT": "SALDOC[FORM=Related Files]",
  "KEY": "5422",
  "data": {
    "XTRDOCDATA": [{
      "LINENUM": "1"
    }, {
      "LNUM": "1",
      "LINENUM": "9000002",
      "NAME": "https://www.softone.gr/wp-content/uploads/2017/10/SCREEN-SFA-2.png",
      "SOFNAME": "https://www.softone.gr/wp-content/uploads/2017/10/SCREEN-SFA-2.png"
    }
  ]
}
```

Response Sample

```
{
  "success": true,
  "id": "5422"
}
```

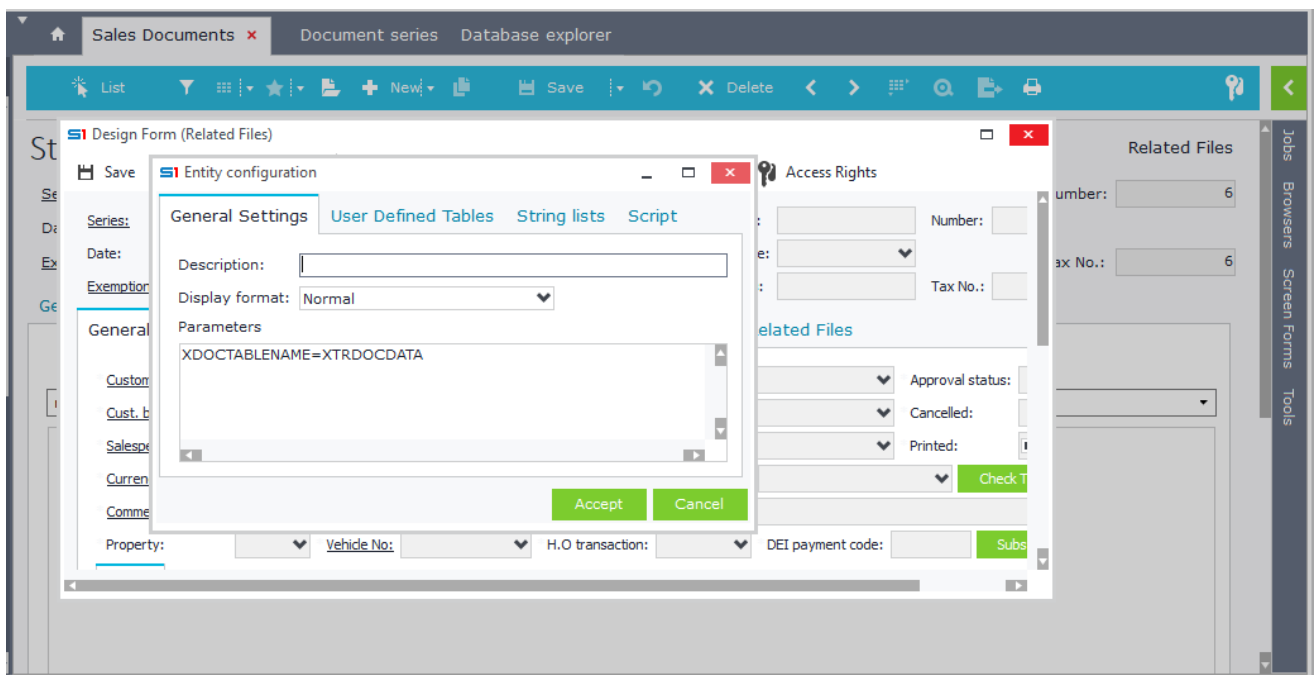


Figure CS9.1 – Add XTRDOCDATA table in Sales Documents form (Related Files)

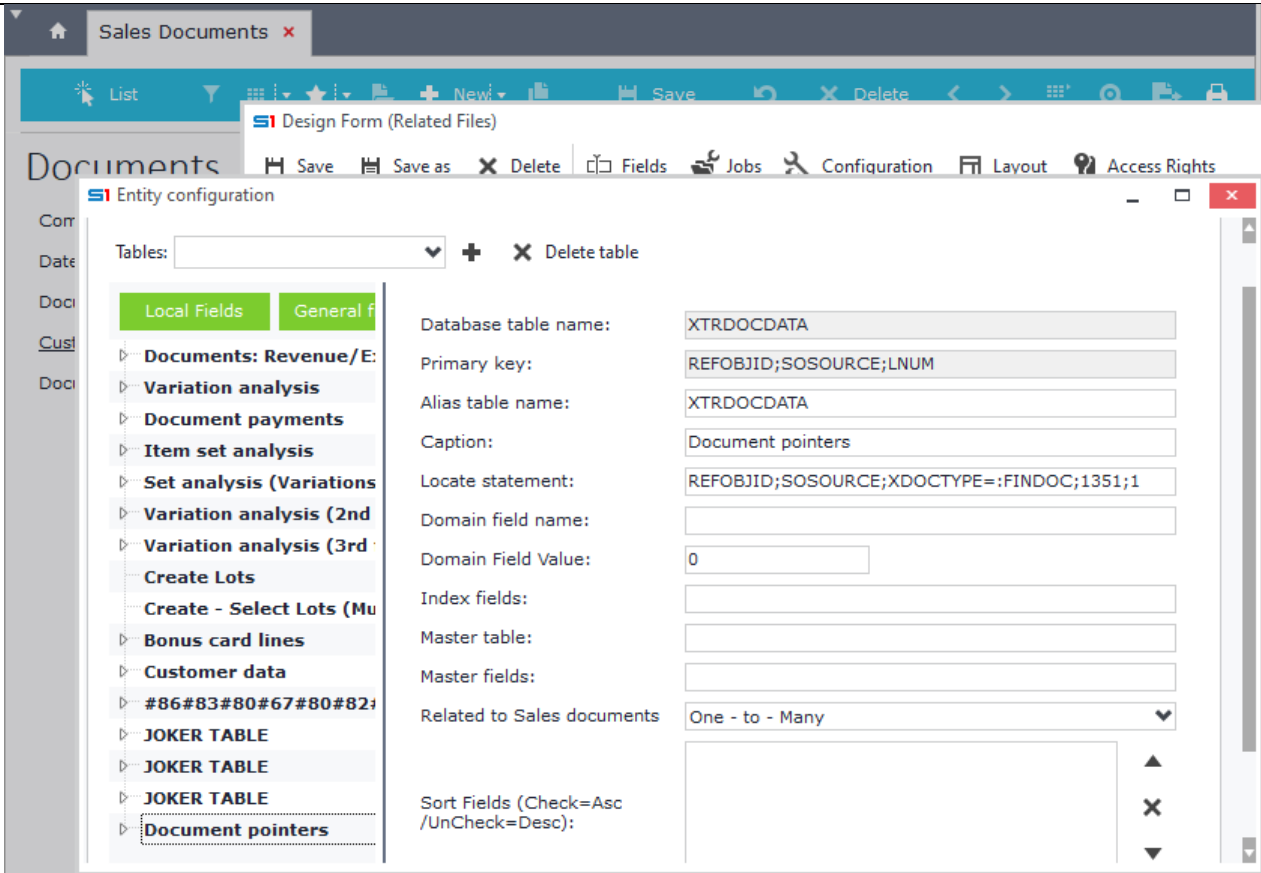


Figure CS9.2 – Locate statement of XTRDOCDATA table

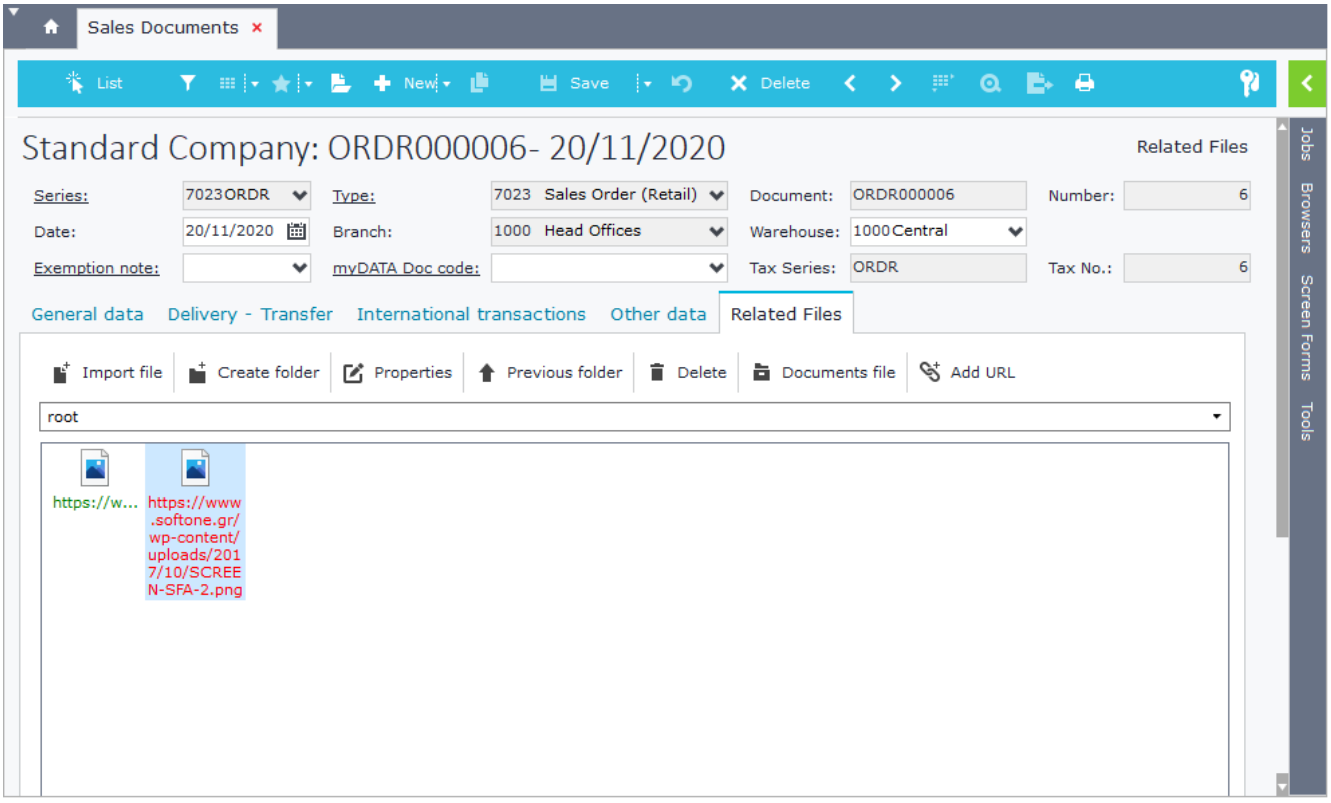


Figure CS9.3 – Related File added from web service call (SetData)

13. BUSINESS AUTOMATION MACHINE

A. [Overview](#)

B. [Setup](#)

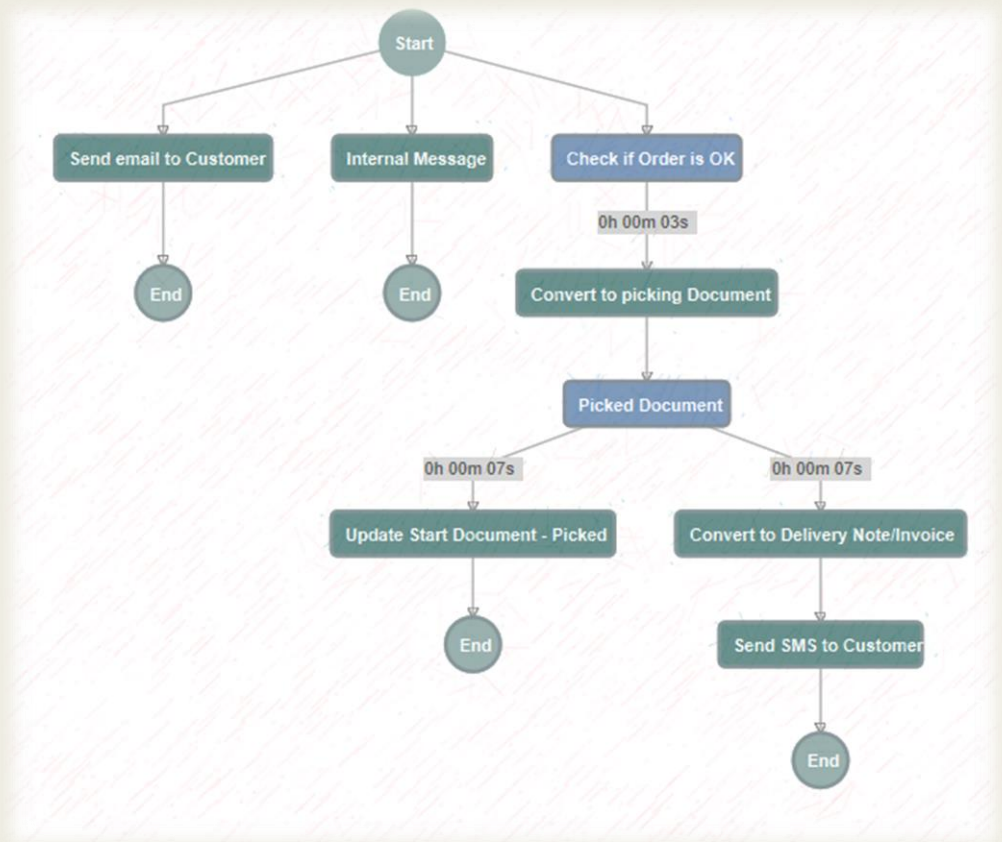
C. [Design & Mode](#)

D. [Variables](#)

E. [Process Steps](#)

F. [Runtime](#)

G. [Audit & Reports](#)



A. Overview

Business Automation Machine (B.A.M.) is an advanced Workflow tool that combines all the SoftOne customization tools in order to run jobs over different SoftOne Business objects.

BAM runs in **server-side**, using Cloud Agent and executes all processes **asynchronously**. Processes can be triggered through user-interaction (entities events) or through SoftOne scheduler. The results of a process are displayed in a visual viewer using diagrams.

A graphical modelling view is used, which is user-friendly and provides an overview of the different steps and their properties, both in design mode and in runtime. In design mode, the screen is vertically splitted and displays the modeling view on the left, while on the right-hand side it shows different actions and commands that change upon the type of the selected step.

There are numerous jobs that can be executed using BAM, such as send e-mail /sms / internal messages, export files, insert/update records, run sql commands, print forms, create reminders, run data flows or even make HTTP requests.

A development environment is offered so as to test Workflows before publication. There is also an option of deactivating workflows that were used in the past and their records need to be maintained in the system.

Sub-workflows are also available for design, which can be nested in any workflow. They are processed as different flows and allow parameters and results to be transferred between parent and child flows.

At runtime, the records of SoftOne objects (including the custom ones) that are affected from a BAM workflow display the flow of the process through the related job "Business Process". This job displays the process in the same graphical modeling view that it was originally designed inside BAM designer.

B.A.M. design is saved in the blob field SODATA of the table CSTINFO (CSTTYPE=19) and can be exported / imported through different installations as .cst or .auv files using the "[Custom Administration](#)" tool.

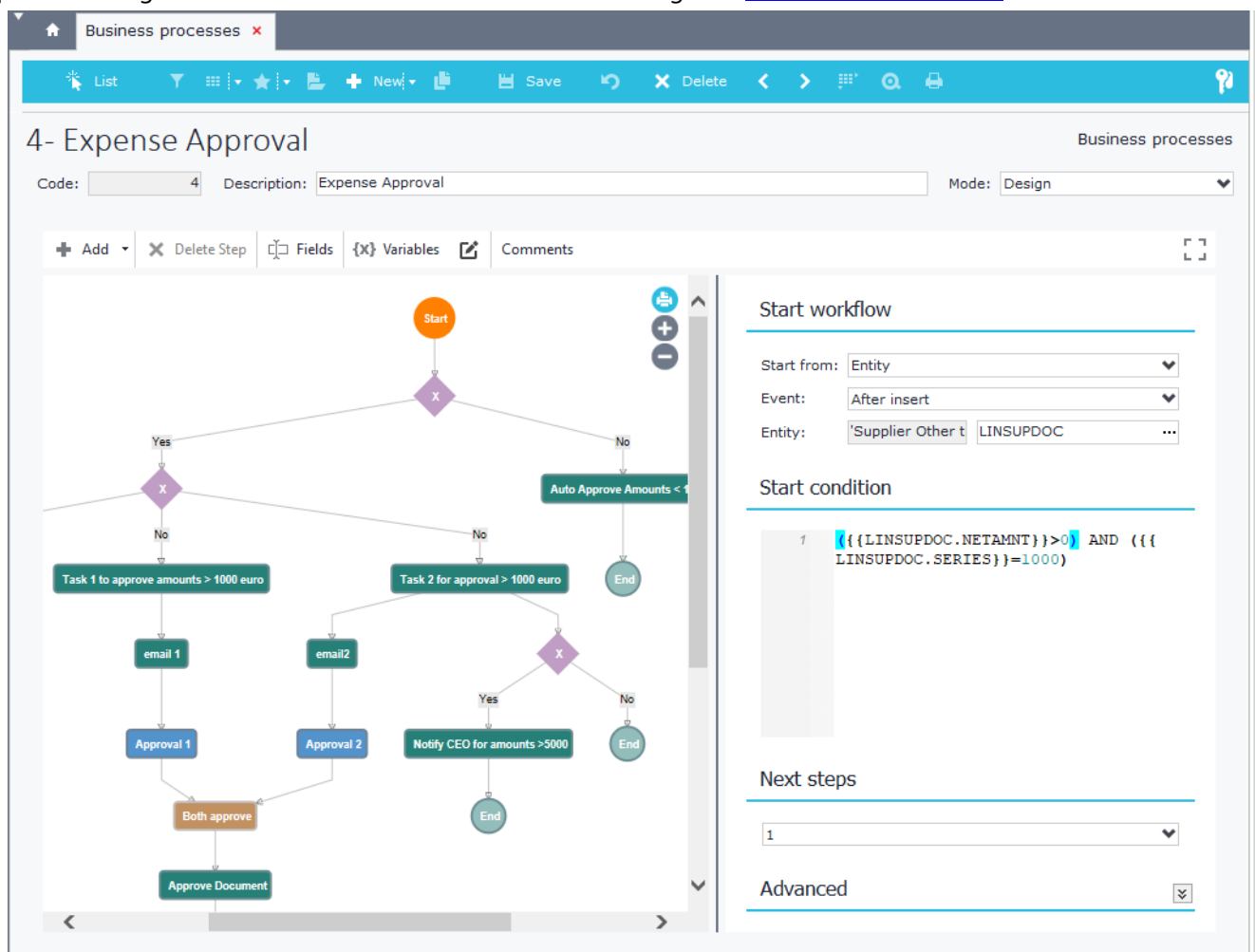


Figure A1

B. Prerequisites & Setup

B.A.M. requires the module Soft1 B.A.M. which also adds Soft1 Open Enterprise Engine (Figure B1). Processes of B.A.M. are activated as long as you turn on S1 Agent (StandAlone or Azure installations).

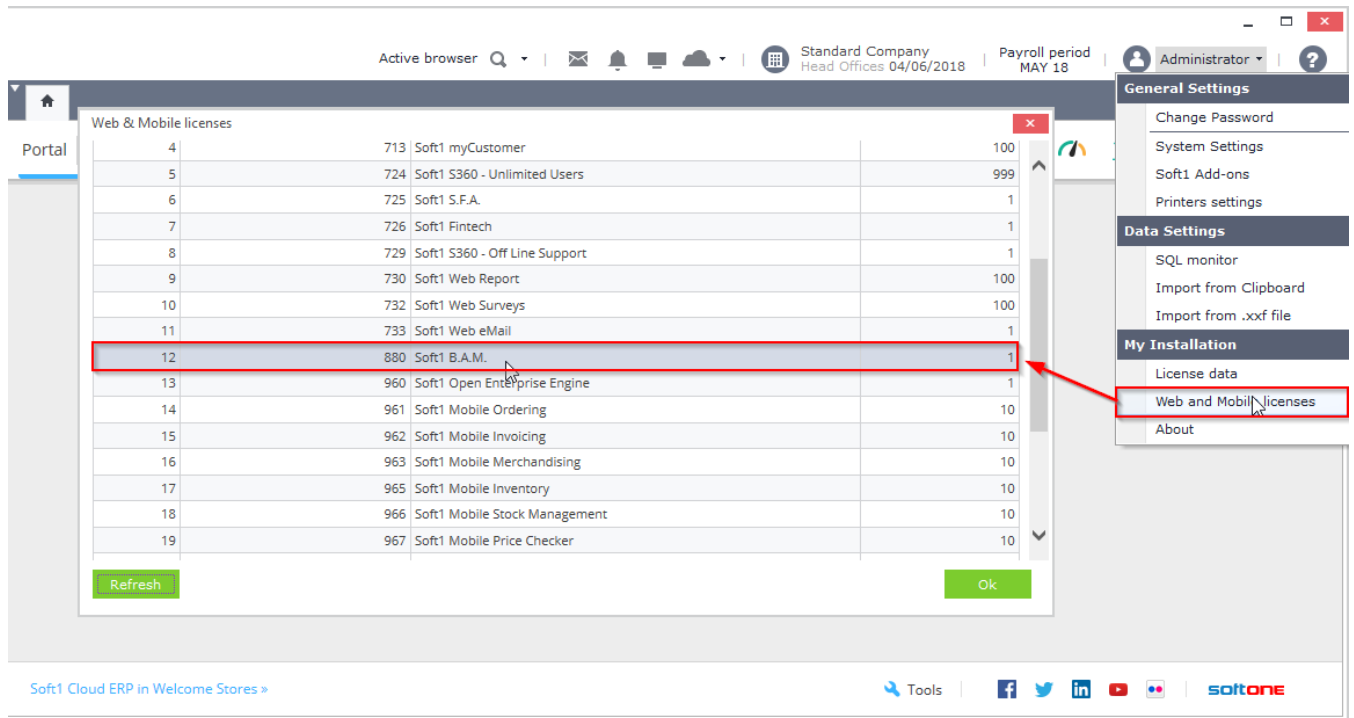


Figure B1

S1 Agent setup is the same as in Web Services. Click on the "Cloud" button on top of SoftOne application and then select "Configure S1 Agent". On top of the form that opens, enter the domain name of your site, which will be domainname.oncloud.gr and click on "Register". Then, fill in the rest of the form and click on "Activate" (Figure B2).

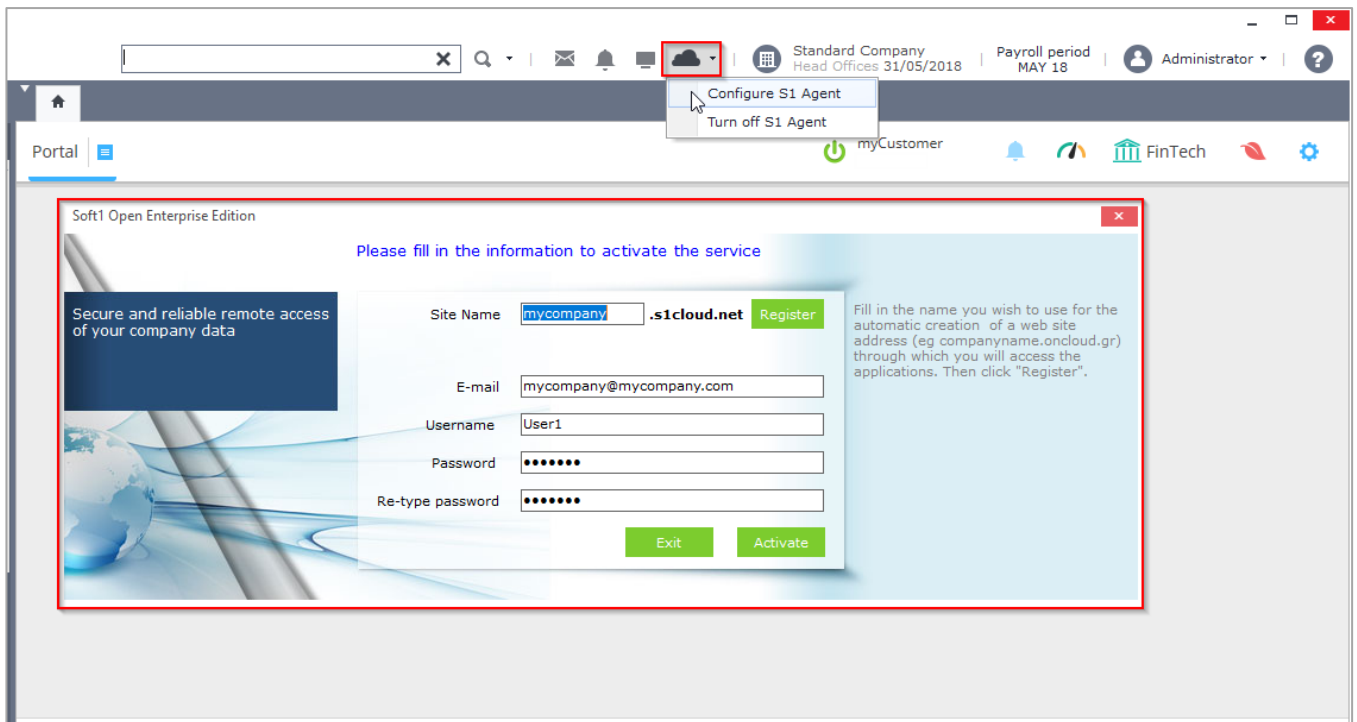
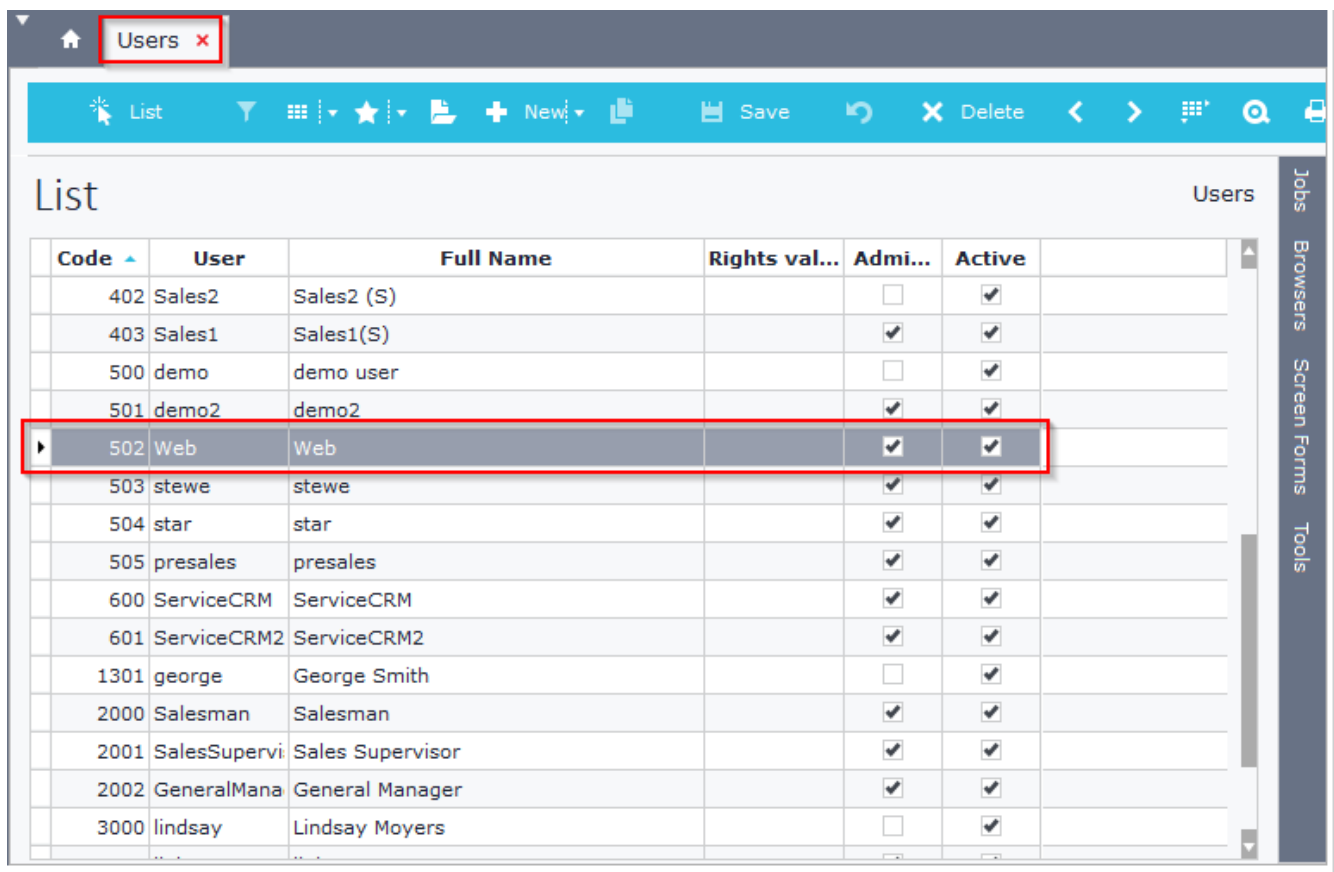


Figure B2

S1 Agent works as SoftOne server and automatically logs into SoftOne using a user named "WEB". If no user is found under that name, then it uses the first Administrator user that it finds (Figure B3).



The screenshot shows a web application interface with a 'Users' tab selected. The interface includes a toolbar with options like List, Filter, New, Save, and Delete. The main area displays a table of users. The user 'Web' (Code 502) is highlighted with a red box, indicating it is the user used by the S1 Agent.

Code	User	Full Name	Rights val...	Admi...	Active
402	Sales2	Sales2 (S)		<input type="checkbox"/>	<input checked="" type="checkbox"/>
403	Sales1	Sales1(S)		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
500	demo	demo user		<input type="checkbox"/>	<input checked="" type="checkbox"/>
501	demo2	demo2		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
502	Web	Web		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
503	stewe	stewe		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
504	star	star		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
505	presales	presales		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
600	ServiceCRM	ServiceCRM		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
601	ServiceCRM2	ServiceCRM2		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
1301	george	George Smith		<input type="checkbox"/>	<input checked="" type="checkbox"/>
2000	Salesman	Salesman		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
2001	SalesSupervi	Sales Supervisor		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
2002	GeneralMana	General Manager		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3000	lindsay	Lindsay Moyers		<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figure B3

C. Design & Mode

C1. Available Tools

Business processes consist of steps that are processed indifferently. Steps “Start” and “End” are auto inserted and cannot be deleted. Steps properties allow you to add fields and custom variables (toolbar buttons).

All tools (toolbar buttons, steps, fields, variables ,etc.) are available only in design mode.

Toolbar buttons (Figure C1.1) are explained in the table that follows.

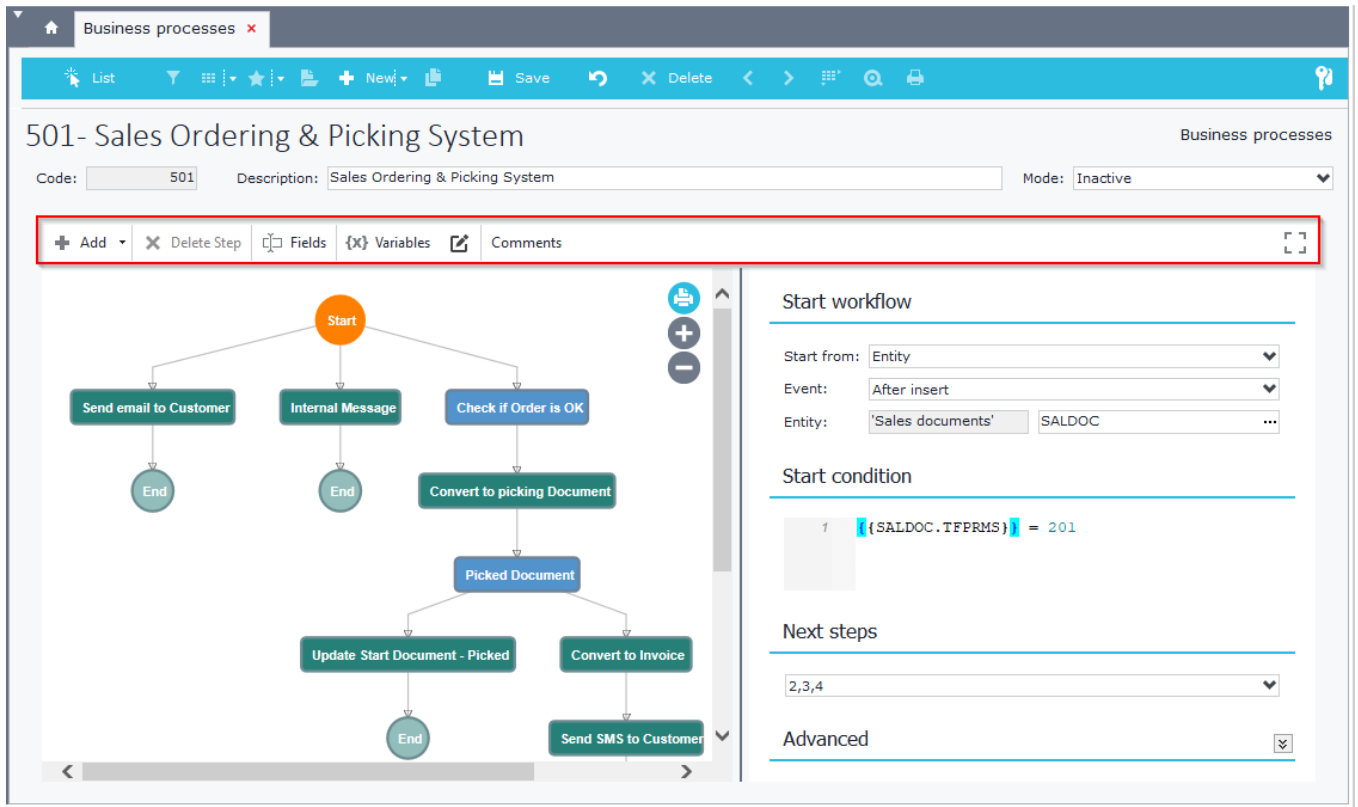


Figure C1.1

BUTTON	DESCRIPTION
Add	Adds a new step in the process.
Delete step	Deletes the selected step.
Fields	Lists the fields of the “Start” entity and is available only if the process is Started from an entity. Fields can be added (using drag and drop) in steps or variables.
Variables List	Lists the created variables of the process that can be added in other variables or in steps.
Variables Edit	Window-grid that allows to insert or edit custom variables
Comments	User-comments about the business process
Screen layout	Maximize / Minimize design layout

C2. Mode

The different modes of a BAM process are explained in the table below.

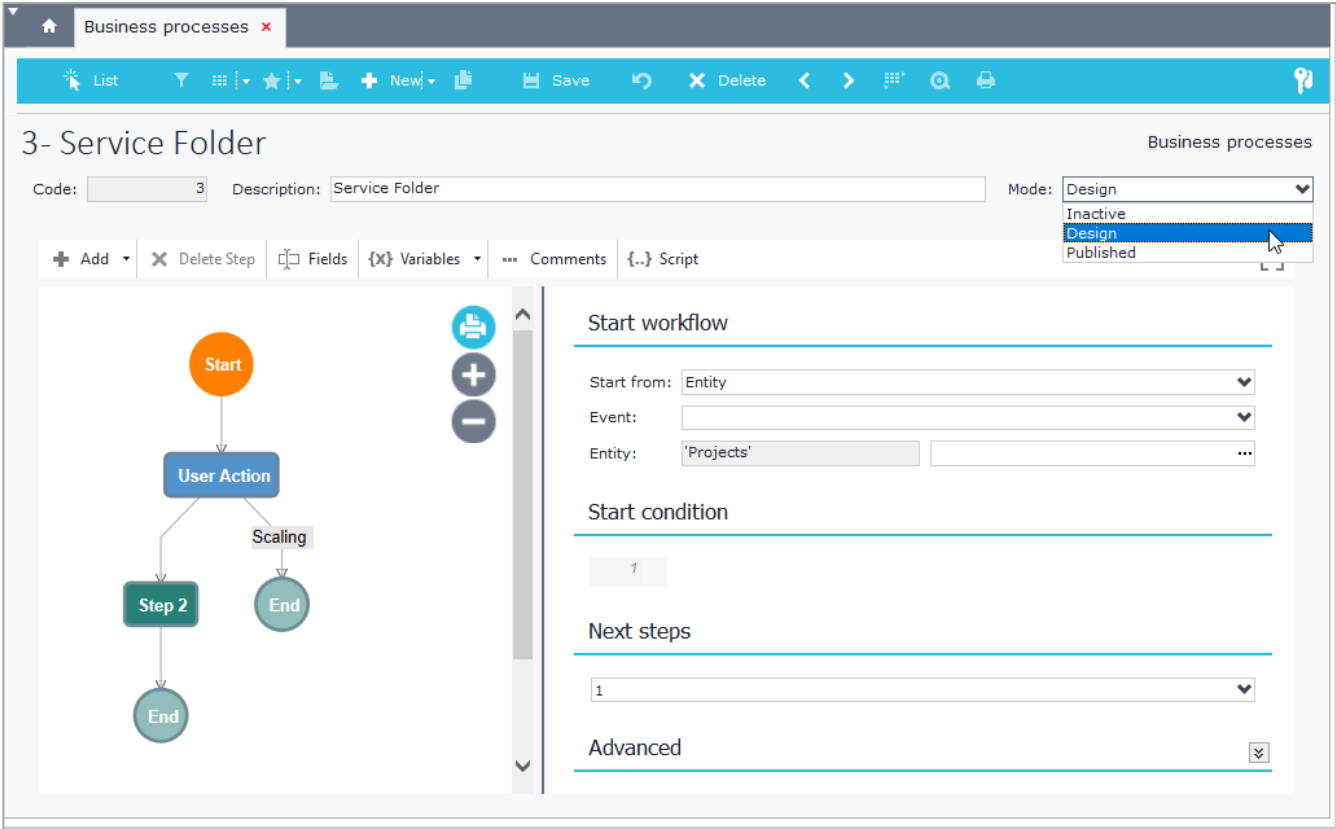


Figure C2.1

MODE	DESCRIPTION
Inactive	Business process is not active, but there might be pending processes that need to be completed. Design is not permitted. Use this mode in order to deactivate a BAM.
Design	Business process is not active and not running. Process is available for design. Use this mode when you are designing a BAM and you have not yet set it to run.
Publication	Business process is active and running. Design is not permitted. In order to stop the process from running you need to change the mode to inactive, where design is not permitted because there might be pending processes that need to be completed.

Note that in order to change the design of a BAM process that is published, it's recommended to turn it to "Inactive" mode and then design a copy of it. Inactive mode allows pending processes (steps that have not yet completed) to be executed normally. If you alter the design of a BAM that was previously published and pending processes exist, then they will not be executed and errors will occur.

C3. Process Testing

BAM is offered for **testing** in standalone installations, if you run S1Agent using webfeeder and SoftOne using the switch /BAM:LOCAL. This switch opens SoftOne and at the same time it activates a local BAM server, which executes all business processes that are in **Publish** or **Design** mode. This lets you test a business process, while in Design mode, which means that it will run as if it was published.

Steps for testing BAM processes are:

1. Create a shortcut for Softone and add the switch /WEBFEED:LOCAL (Figure C3.1)
2. Create a shortcut for Softone and add the switch /BAM:LOCAL (Figure C3.2)
3. Create an XCO file named WEB.XCO, add the configuration data of your installation and place it inside the SoftOne application folder.
4. Run Webfeeder (Step 1). This creates a web server using web.xco file.
5. Run SoftOne using the shortcut of Step 2. This creates the BAM server that serves processes locally and opens SoftOne for testing design mode business processes.

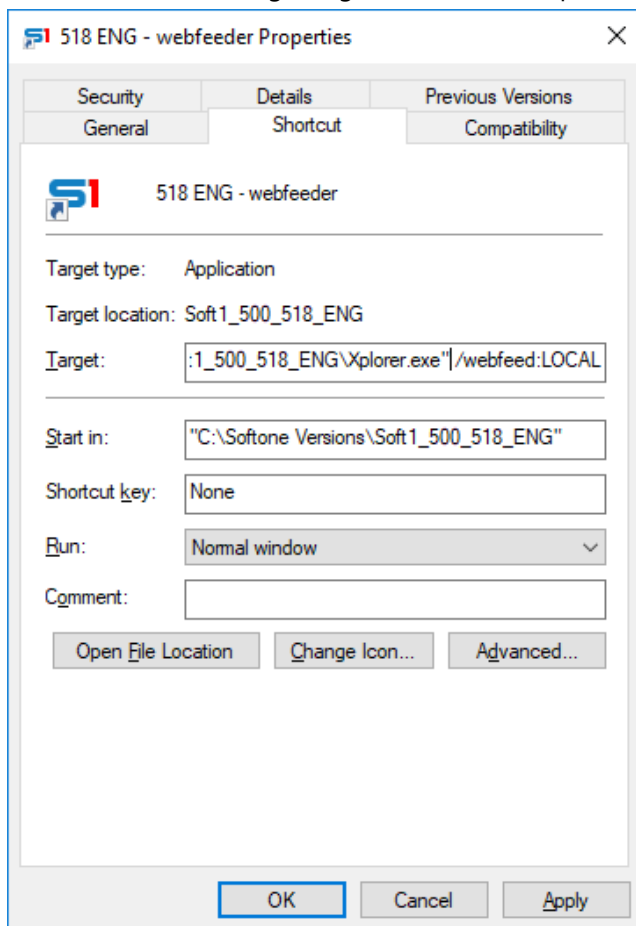


Figure C3.1

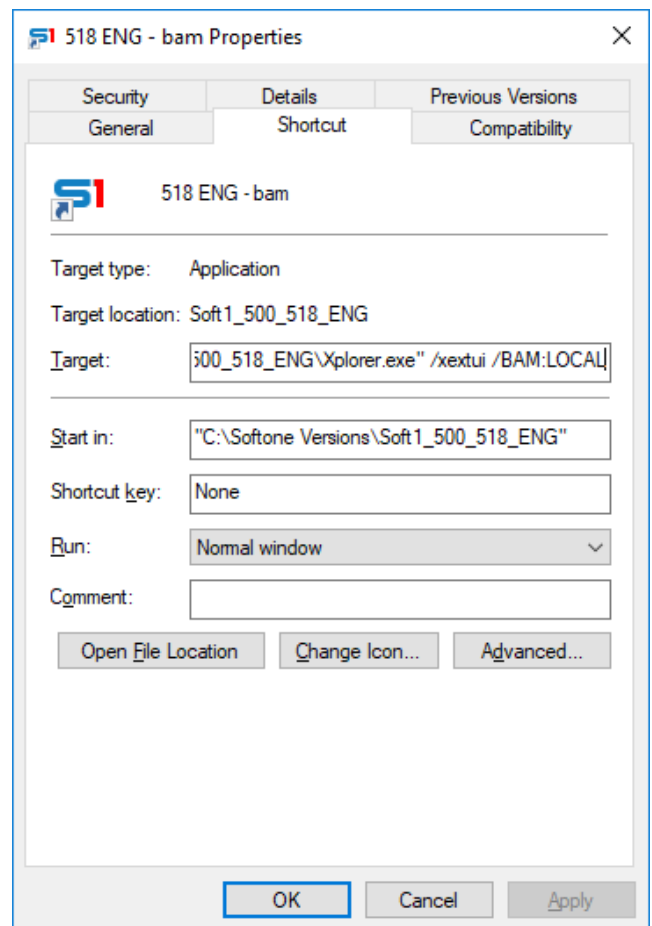


Figure C3.2

D. Variables

Variables are defined in design mode and are used to store the results of calculations inside the steps of the business process. They can also store the results of a step and then used in other steps of the business process. The available types of variables are **Calculation**, **SQL Command**, **Function** and **Soft1360**.

Variables are calculated **only** when you use them inside the properties of a step ("Results", "Conditions", "Locate", "Initialize Variables", etc.). A variable is initialized (calculated for the first time) if it has no assigned value. This occurs when you use it inside a property of a step (e.g. "Conditions" or "Mapping Fields"), except for the property "Results" where its value is reassigned. This means that if for example you have a variable that selects data from a table and you use it more than one times (in different steps), then the sql statement will not be executed every time, if it already has data, but it will use the data from the previous steps. It's obvious that "Results" property of a step produces different data, so each time you use a variable inside that property it will be reassigned with the new data. If you need to store different record fields of the same object, that appear in more than one steps then you need to create different values. For example if you need to store the customers of tasks (SOACTION.TRDR) that are inserted in different steps of your process, then you need to create different variables (vTRDR1 – Calculation – SOACTION.TRDR and vTRDR2 – Calculation – SOACTION.TRDR) and use them in the result fields of the steps.

Variables are created through the toolbar button "**Variables – Create**". This opens a window, where you can define variables that may be used inside any step of your workflow (Figure D1). A new variable is inserted in the grid when you use the down arrow of the keyboard.

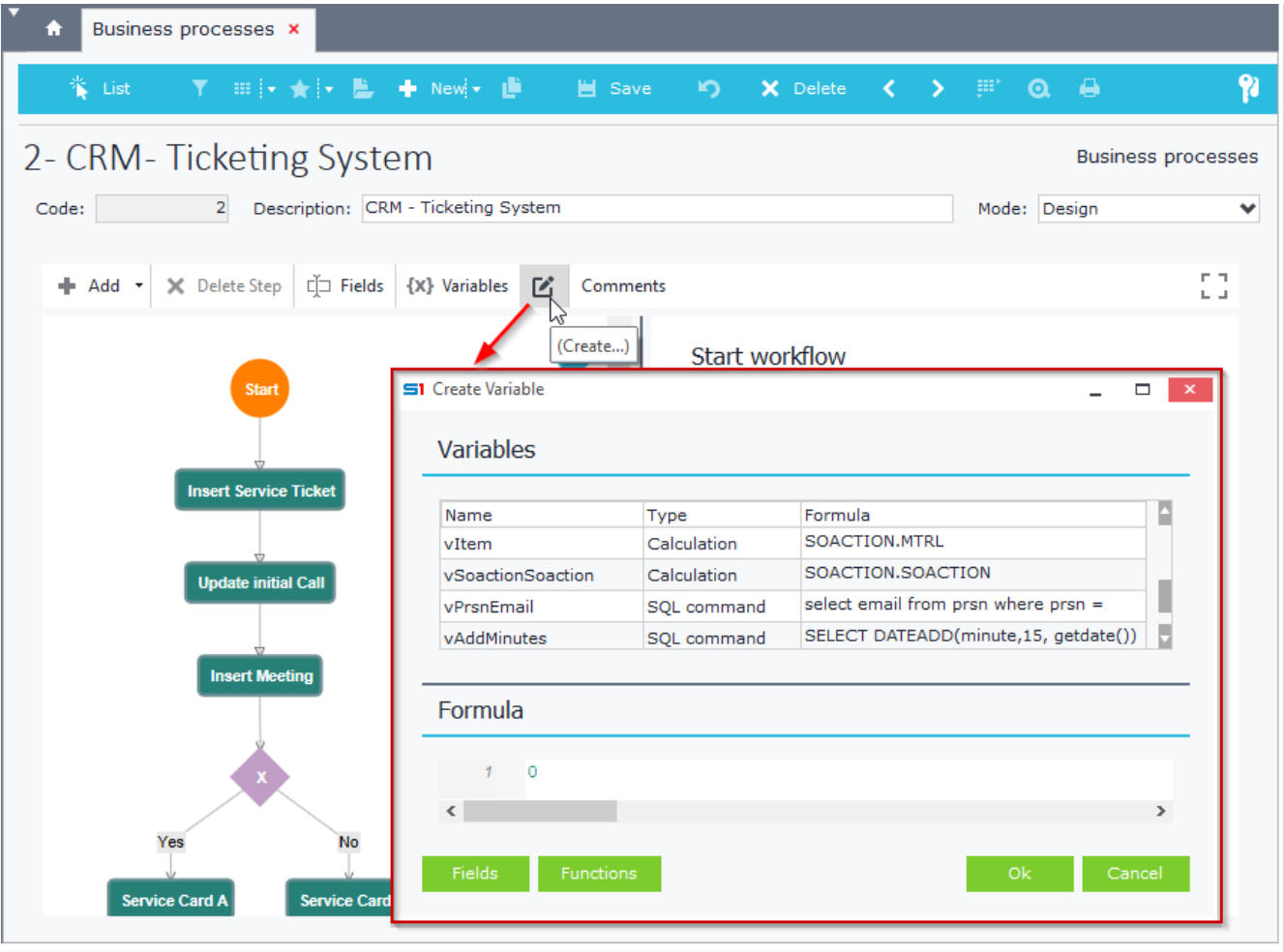


Figure D1

D.1 Calculation

It is used for storing values that are processed inside steps and can be later used in any other step of the workflow. The results that are produced by some types of steps can be stored in calculation variables. A step type “**Job**” that inserts or update a record in a specific object can store the posted recordset in a calculation variable. The name of the variable must be entered in the “**Result**” textbox. The id of the result recordset is referenced using the expression `{{VariableName.id}}`, while all the result fields are referenced using the expression: `{{VariableName.data.TableName[0].FieldName}}`.

`{{VariableName.data.TableName[0].FieldName}}`.

In the example of Figure D1.1 the step “**Task for Lead**” inserts a new task (SOTASK) and the result recordset of SOACTION is stored in the variable “**Qtask**”.

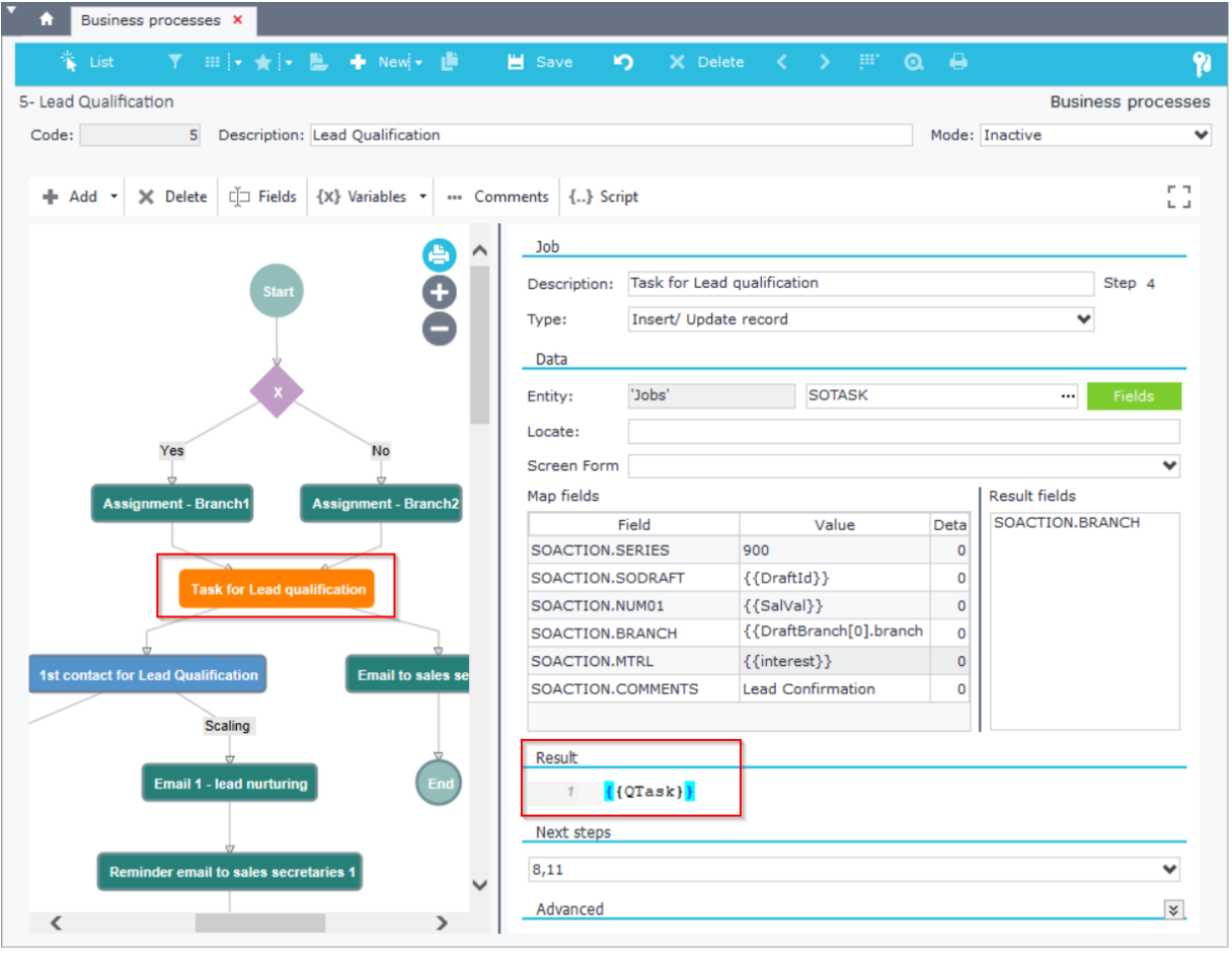


Figure D1.1

It is then used in a “**User Action**” step, to check if this specific task (Locate `{{QTask.id}}`) is completed (Condition `SOACTION.ACTSTATES = 600`) in order to move on to next steps (Figure D1.2).

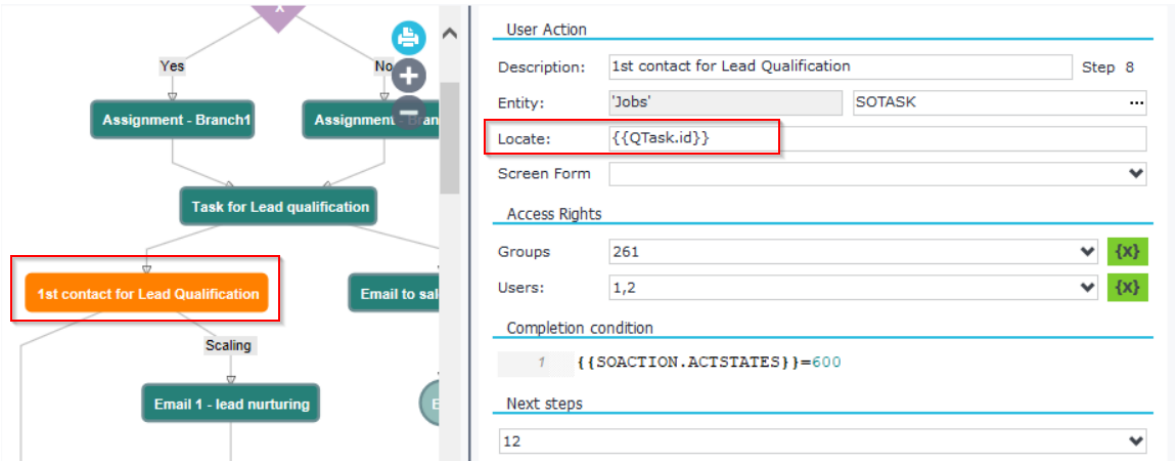


Figure D1.2

The variable "**Qtask**" is initialized inside "**Variables**" as in Figure D1.3.

Name	Type	Formula
QTask	Calculation	0
QTaskCode	SQL command	SELECT soactioncode FROM SOACTION WHERE SOACTION={{QTask.id}}
SystemDate	Calculation	:X.SYS.LOGINDATE
LCustomer	SQL command	select trdr from sodraftlink where sodraft={{DraftId}}

Formula
1 0

Fields Functions Ok Cancel

Figure D1.3

Another use of "Calculation" is for storing field values, that are processed inside conditions or result fields of a step. In the example of Figure D1.4 the variable "**DraftId**" stores the value of the field "**SODRAFT.SODRAFT**", which is initialized in the Start of the workflow (Entity SODRAFT) and then used to map fields in another step (Figure D1.5).

Name	Type	Formula
DraftId	Calculation	SODRAFT.SODRAFT
DraftPname	Calculation	SODRAFT.NAMEF
DraftCname2	Calculation	SODRAFT.NAMEF
DraftPname	Calculation	SODRAFT.NAMEF

Formula
1 SODRAFT . SODRAFT

Fields Functions Ok Cancel

Figure D1.4

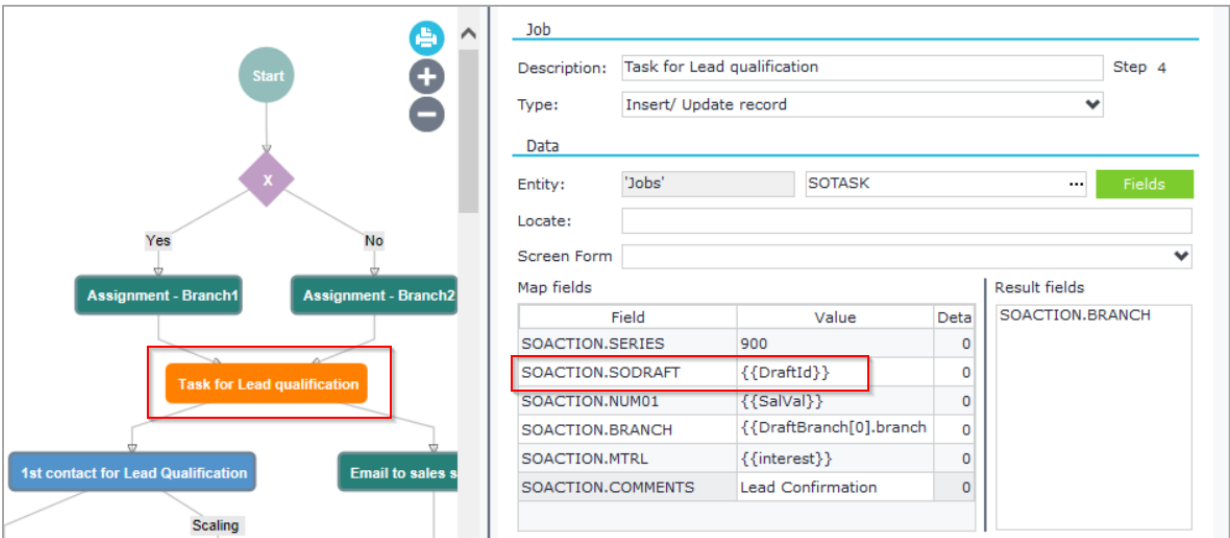


Figure D1.5

D.2 SQL Command

SQL Command is used for retrieving data from database using sql statements and the results are available in any step. The main advantage comparing to the corresponding command in user-defined fields is that you have the option to **select as many fields and records** as you like and not only one. This is very efficient as it does not send many queries to the database.

Sql command data are referenced using the expression: **{{VariableName[recordno].Column}}**

In the example of figure D2.1, the step **"Create sales opportunity"** inserts a new project (PRJC) and maps fields using variables. Field **PRJC.TRDR** is mapped with variable **{{LCustomer[0].trdr}}** which is the result of the query:

```
select trdr from sodraftlnk where sodraft={{DraftId}}
```

Field **PRJC.NAME** is mapped with variable **{{CustName[0].name}}** which is the result of the query (Figure D2.2) :

```
SELECT name, code FROM TRDR WHERE TRDR={{LCustomer[0].trdr}}
```

It's obvious that you can use reference variables inside other variables, if they have been initialized first.

Job

Description: Create sales opportunity Step 5

Type: Insert/ Update record

Data

Entity: 'Projects' PRJC Fields

Locate:

Screen Form

Map fields

Field	Value	Data
PRJC.TRDR	{{LCustomer[0].trdr}}	0
PRJC.NAME	{{CustName[0].name}}	0
PRJC.PRJCRM	2	0
PRJC.CODE	Y*	0
PRJC.BRANCH	{{QTask[0].SOACTION.B	0
PRJC.INVAL	9000	0

Result fields

PRJC.CODE

Result

1 {{Lprjc}}

Next steps

9,36

Advanced

Figure D2.1

Create Variable

Variables

Name	Type	Formula
LCustomer	SQL command	select trdr from sodraftlnk where sodraft={{DraftId}}
CustName	SQL command	SELECT name, code FROM TRDR WHERE TRDR={{LCustomer[0].trdr}}

Formula

1 select trdr from sodraftlnk where sodraft={{DraftId}}

Fields Ok Cancel

Figure D2.2

D.3 Soft1 360

This command is used for creating references to SoftOne web and mobile applications. An example of a web function is DeepLink, that has the following syntax:

DEEPLINK:ObjectName[attributes],RecordID, FormName

The example of Figure D3.1 defines a variable that creates a deep link for a specific record of an object. Variable "linkforprjc" is used in a Job step that sends an email. The body of the email, contains a link of the record for Soft1 360 site (Figure D3.2).

Create Variable

Name	Type	Formula
TEST1	Calculation	1
ISPPP	Calculation	CUSTOMER.ISPROSP
linkforprjc	Soft1 360	DEEPLINK:PRJC["FORCEFILTERS=PRJCRM:2"],{{Lprjc.id}},SFA
interest	Calculation	SODRAFT.UCTBL02

Formula

1 DEEPLINK:PRJC["FORCEFILTERS=PRJCRM:2"],{{Lprjc.id}},SFA

Fields Ok Cancel

Figure D3.1

Job

Description: Email to salesman Step 10

Type: Send Email

Email

From: Email Account Name: SalesDirector

To: {{SalesSecEmail[0].LIST}} Attachment:

CC: sales@mycompany.com BCC:

Subject: New lead - {{CustName[0].name}}

Normal Verdana 1 B I U ABC

New Lead

Contact: {{CustName[0].name}} / Phone: {{sodraftvalues.data.SODRAFT[0].PHONELOCAL}}

Soft1 360 Link.

Result

Next steps

34

Advanced

Context menu: Undo, Cut, Copy, Paste, Delete, Select all, Opening, Save as, Page code, Report/Printout, Preview

Figure D3.2

E. Process Steps

A step is the part of the process that defines the actions that will take place inside the flow. A new step is created through right click on the left panel, or through the toolbar button "Add". Selected (focused) steps are highlighted on the left panel with orange color. All step types can be used in parallel or sequentially. Sequential (serial) process steps are executed one after another, while parallel ones are executed simultaneously. Parallel steps are used when for example you have an approval system and you need to send separate approval tasks to many approvers and then continue the process based on the actions of one or more of them. The different types of steps you can use inside a business process are discussed in the following sections.

E.1 Start

A business process can be started (triggered) from **Entity** records that are posted by users, **Scheduler** or other **parent business processes**. These different ways are discussed below.

E.1.1 Entity

Records that are inserted or updated using SoftOne objects can start a business process. Start Conditions, that fire these events, are defined through the **"Start Condition"** option (Figure E1.1).

On the right-hand side select **"Entity"** from the **"Start from"** list, choose the desirable event (After Insert, After Update or After Post) and then enter the object that will start the workflow. Any SoftOne object can be used, even custom ones for the start entity of the workflow. "After post" event fires either on insert or on update of the record. The conditions that trigger the start process entity are entered inside the **"Start condition"** panel. You are allowed to use any field of the start entity (Toolbar button **"Fields"**), or created **variables** (e.g. SQL commands) and **X.SYS** parameters. Variables entered here are initialized and can be later used in other step of the process.

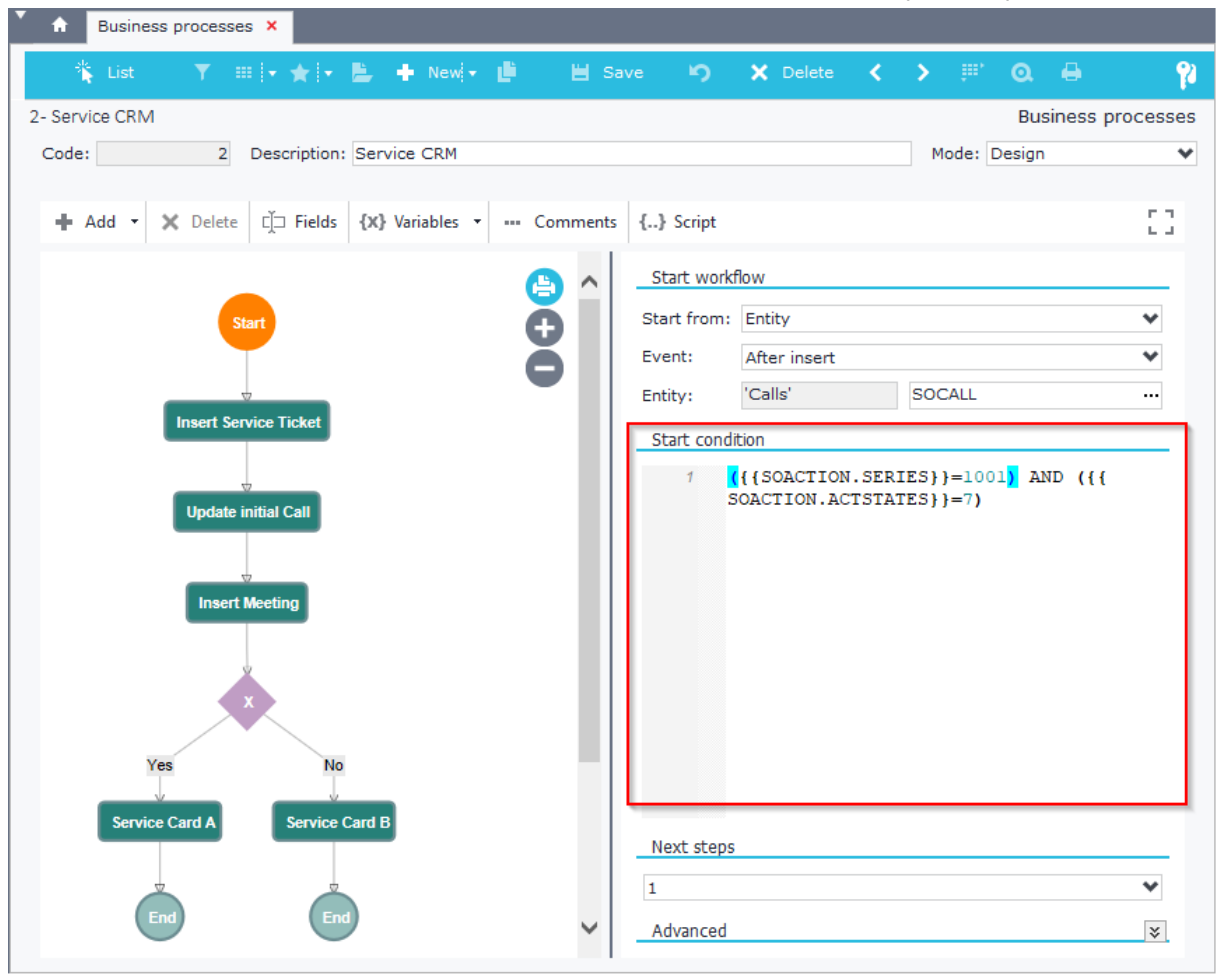
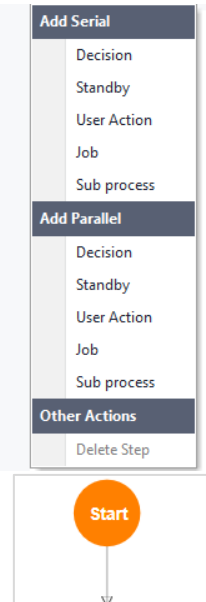


Figure E1.1

E.1.1.1 Before Post Javascript

There is an option of writing **Javascript** code that will run in the “**Before post**” event of the object defined in the Start Entity of the business process. This can be very helpful when you need to select a value (and save it in a variable) before the record is posted and then compare it with posted ones in later steps.

It can also be used when you need to alter a value (e.g. SALDOC.APPRV = 0) and trigger the workflow in the after post event by checking the results that occurred due to the changes of this value along with other conditions. Javascript code can be written inside the “**Advanced**” panel (Figure E1.1.1).

The screenshot shows the SAP Business Process Manager (BPM) interface for the "4- Expense Approval" process. The interface includes a toolbar with "List", "New", "Save", "Delete", and navigation icons. The main area displays a flowchart on the left and configuration panels on the right.

The flowchart starts with an orange circle, goes through a decision diamond, then a task "Task 2 for approval > 1000 euro", another decision diamond, and tasks "email2", "Approval 2", and "Notify CEO for amounts >5000" before ending.

The right panel shows the configuration for the process:

- Start workflow:** Start from: Entity, Event: After insert, Entity: 'Supplier Other transa', LINSUPDOC
- Start condition:** 1: {{{LINSUPDOC.NETAMNT}}} > 0) AND {{{LINSUPDOC.NETAMNT}}} > 1000
- Next steps:** 1
- Advanced:**
 - Before post (Javascript):**

```

1 if (FINDOC.FINDOC<0)
2 {
3   LINSUPDOC.APPRV=0;
4 }

```
 - Initialize Variables:**

Figure E1.1.1

E.1.1.2 Variables Initialize

Before post values can be saved into variables, through the “**Initialize Variables**” memo. Enter here the variables that you need to store data before the start entity record is posted, so as to use them in any other step.

For example you may need to save the customer’s balance before a Sales Document is saved, so as to compare it with the after post balance in other steps and create different types of processes.

E.1.2 Scheduler

SoftOne scheduler can be set to start a business process. This means that you can schedule a business process to run at a specific time and day (or recurrently) without the need of user action.

The commands that must be entered inside the scheduler are (Figure E1.2.1):

```
TYPE=BAM
JOBNAME=BAMRecordID (id of the Business process)
```

Figure E1.2.1 shows an example of a Job Scheduler that is set to be executed every month (1st) and run the BAM process with code 10.

The screenshot displays the 'Time scheduling' window for a 'Business pr...' job. The interface includes a toolbar with options like List, New, Save, Delete, and navigation arrows. The main area shows the job details for '25/05/2018- 1', including the insertion date, user (Administrator), and recurrence string '25/05/2018;20:28;0;M;0;1;1'. A red box highlights the recurrence string, and a red arrow points from it to the 'Scheduling' dialog box. The dialog box shows the start time as '20:28' and the recurrence set to 'Monthly' on the 'First' of each month. The active period of execution starts on '25/05/2018' with 'No End Date' selected. The 'Define command' section shows the command 'TYPE=BAM' and 'JOBNAME=10'.

Figure E1.2.1

In the BAM process Start step the “Start from” is “Scheduler” (Figure E1.2.2). The next step sends an email to the specified recipients and the process ends (Figure E1.2.3) and this happens every month (from Scheduler recurrency).

The screenshot shows the BAM process configuration interface. The title bar indicates the process is 'Business pr...'. The main header shows '10- Start from Scheduler' with a Code of '10' and a Description of 'Start from Scheduler'. The Mode is set to 'Inactive'. Below the header, there are tabs for 'Add', 'Delete Step', 'Fields', 'Variables', and 'Comments'. The left pane displays a workflow diagram with a 'Start' step (orange circle) highlighted by a red box, followed by a 'Send Monthly Report email' step (green rectangle), and an 'End' step (green circle). The right pane shows the configuration for the 'Start workflow'. The 'Start from' dropdown is set to 'Scheduler' and is highlighted by a red box. Below it, the 'Entity' field is empty. The 'Start condition' section shows a single condition '1'. The 'Next steps' section shows a single step '1'. The 'Advanced' section is collapsed.

Figure E1.2.2

The screenshot shows the BAM process configuration interface for the 'Send Monthly Report email' step. The title bar indicates the process is 'Business pr...'. The main header shows 'Send Monthly Report email' with a Step of '1'. The Type is set to 'Send Email'. Below the header, there are tabs for 'Add', 'Delete Step', 'Fields', 'Variables', and 'Comments'. The left pane displays a workflow diagram with a 'Start' step (green circle), followed by a 'Send Monthly Report email' step (orange rectangle), and an 'End' step (green circle). The right pane shows the configuration for the 'Job' and 'Email' sections. The 'Job' section has a Description of 'Send Monthly Report email' and a Type of 'Send Email'. The 'Email' section has fields for 'From' (Email Account), 'Name' (Socrates), 'To' (info@mycompany.com), 'Attachment' (empty), 'CC' (empty), 'BCC' (empty), and 'Subject' (Monthly Report). Below the email fields, there is a text area with the content 'This is a monthly report.....'. The 'Result' section shows a single result '1'.

Figure E1.2.3

E.2 Decision

This step is used to define different sub processes based on a true/false condition. The condition may contain Variables, X.SYS parameters or Fields (toolbar button) from the "Start entity". The process splits in as many "Yes" and "No" branches as the ones defined in the "Next steps" panel.

In the example of Figure E2.1 the flow will continue with Step "Service Card A" if the condition $\{\{vTicket.data.SOACTION[0].CCCITEMTYPE\}=3\}$ is true, otherwise it continue with step "Service Card B".

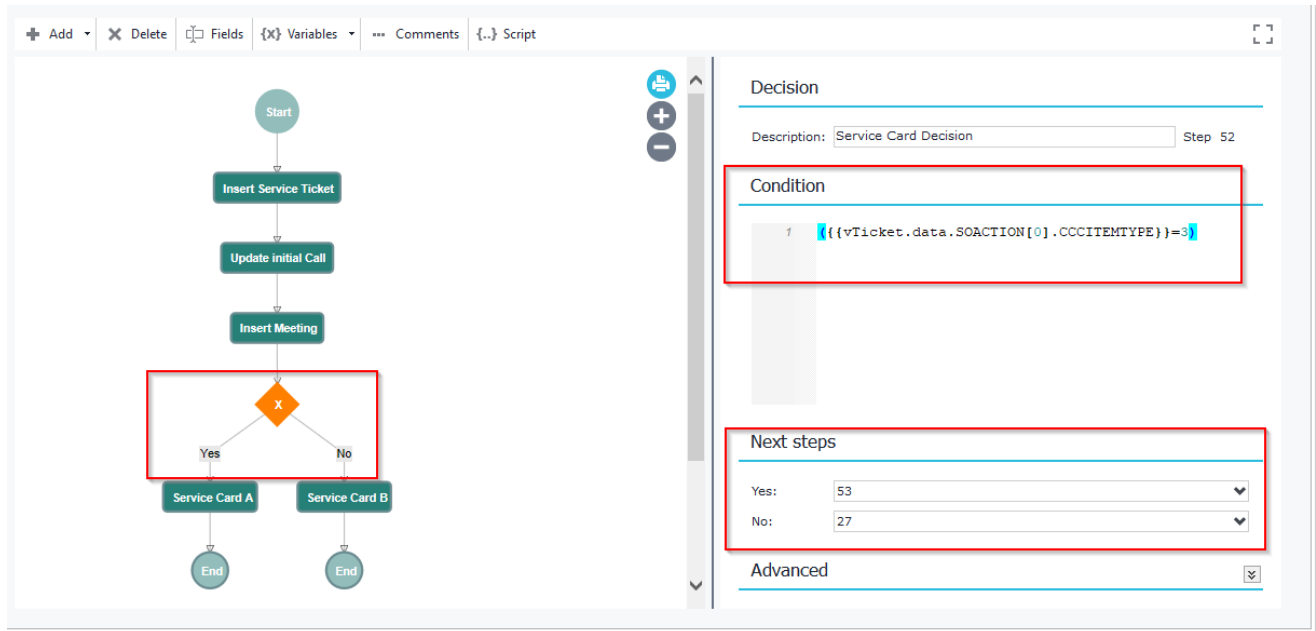
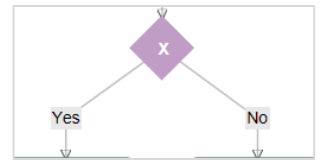


Figure E2.1

The example of Figure E2.2 shows a "Decision" step that splits the process in three different steps. It moves to Step "Task to approve amounts <= 1000 euro" if the Condition $\{\{LINSUPDOC.NETAMNT\}\} \leq 1000$ is true, otherwise it moves to two different steps.

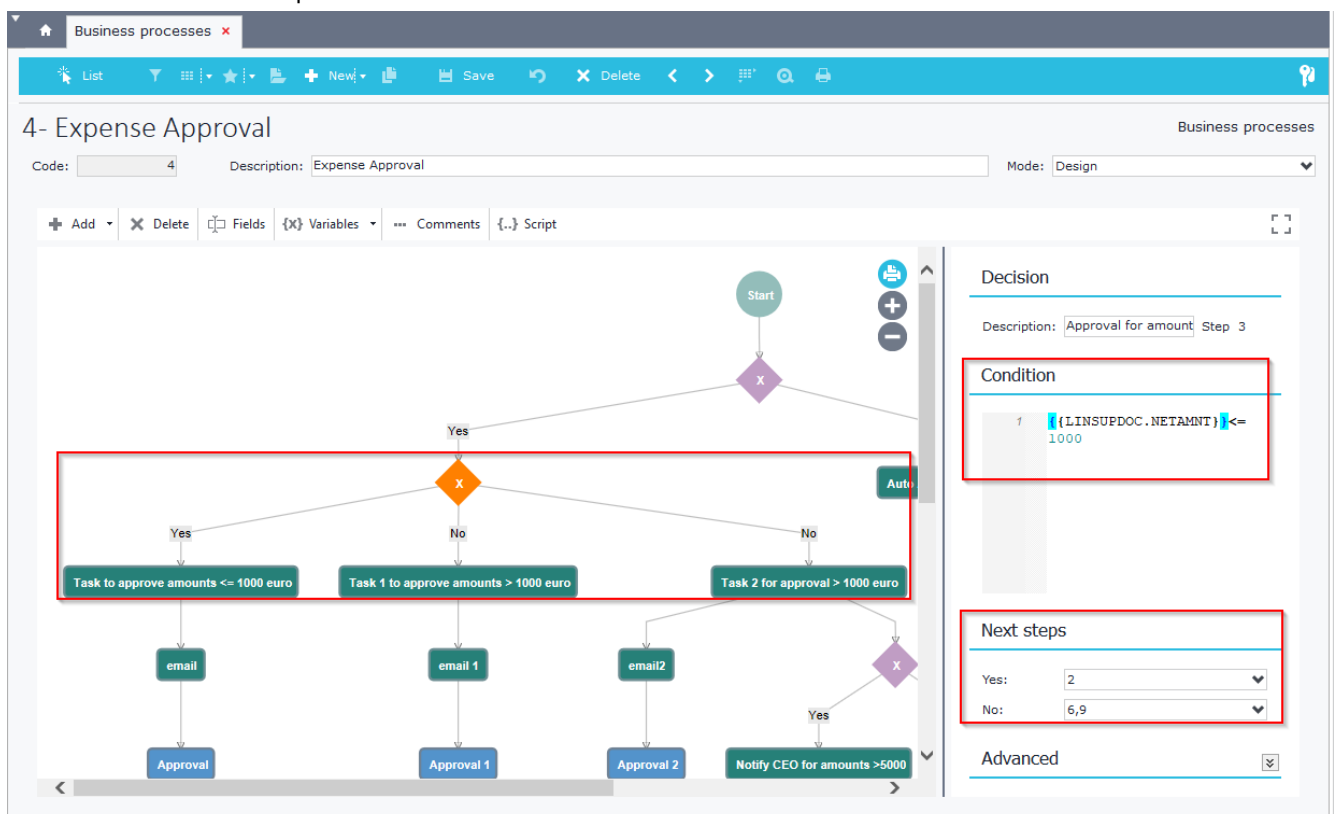


Figure E2.2

Notice that the condition uses the field LINSUPDOC.NETAMNT, which is a field from the “Start” entity (selected through the toolbar button Fields).

The “Start” entity object fields are available for use in the steps of the business process that are above the first “User Action” step. This means that if you need a field value of the “Start” entity in a step that is below a “User Action” step, then you need to create a variable and initialize it in a step above the first “User Action”. This variable will contain the value of the start entity field and will be available for use in any step of the entire process.

In the following example, *SODRAFT* is the “Start” entity of the business process. The step “Task for Lead qualification” initializes the variable `{{interest}}` when used as mapping field of *SOACTION.MTRL* (Figure E2.3).

Variable `{{interest}}` is defined to store the data of the start entity field *SODRAFT.UCTBL02* (Figure E2.4). This way the variable can be later used in any step of the process, even below “User Action” steps.

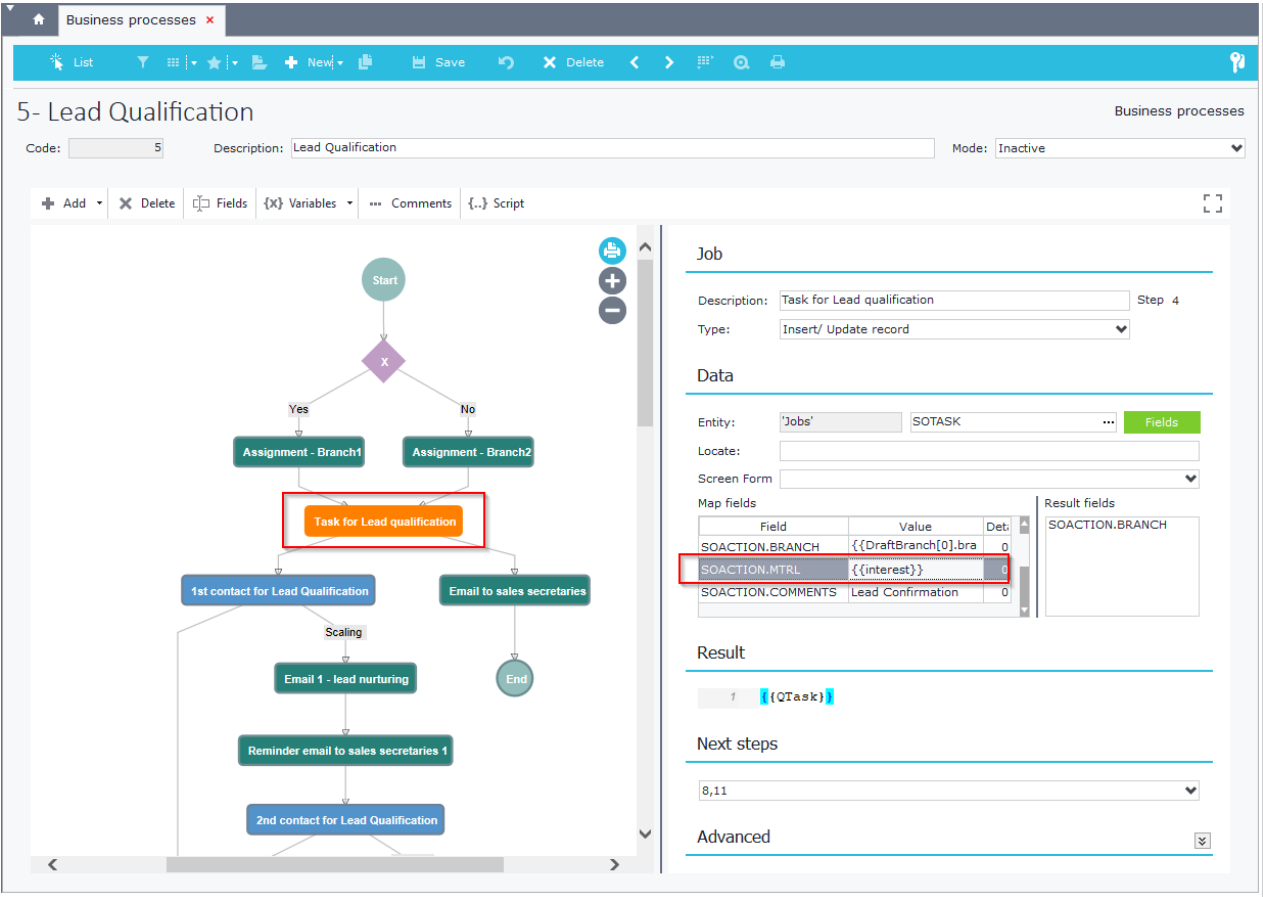


Figure E2.3

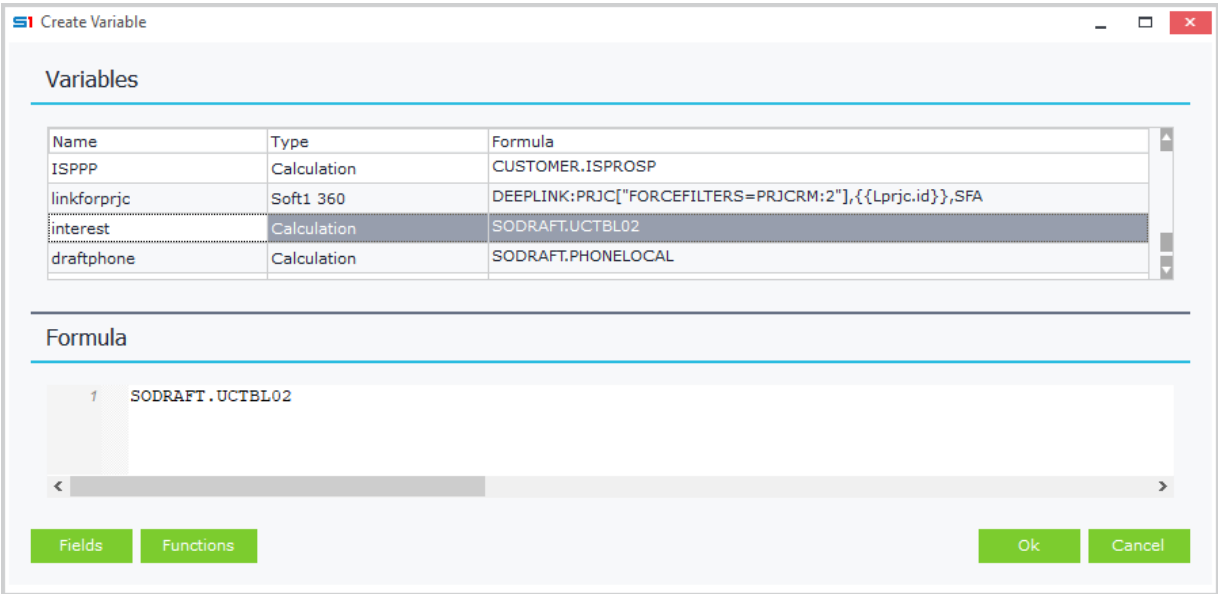


Figure E2.4

E.3 Standby

This type of step waits for other steps to be completed in order to move on to the next step. The steps are entered inside the “Condition” panel as in Figure E3.1.

The following example shows a “Standby” step (18) that waits for steps 15 (Email to sales manager) and 17 (New customer approval) to be completed in order to move to step 19 (Invoicing).

Process 15 (Figure E3.2) is a “Job” type step that sends an email, while process 17 (Figure E3.3) is a sub workflow.

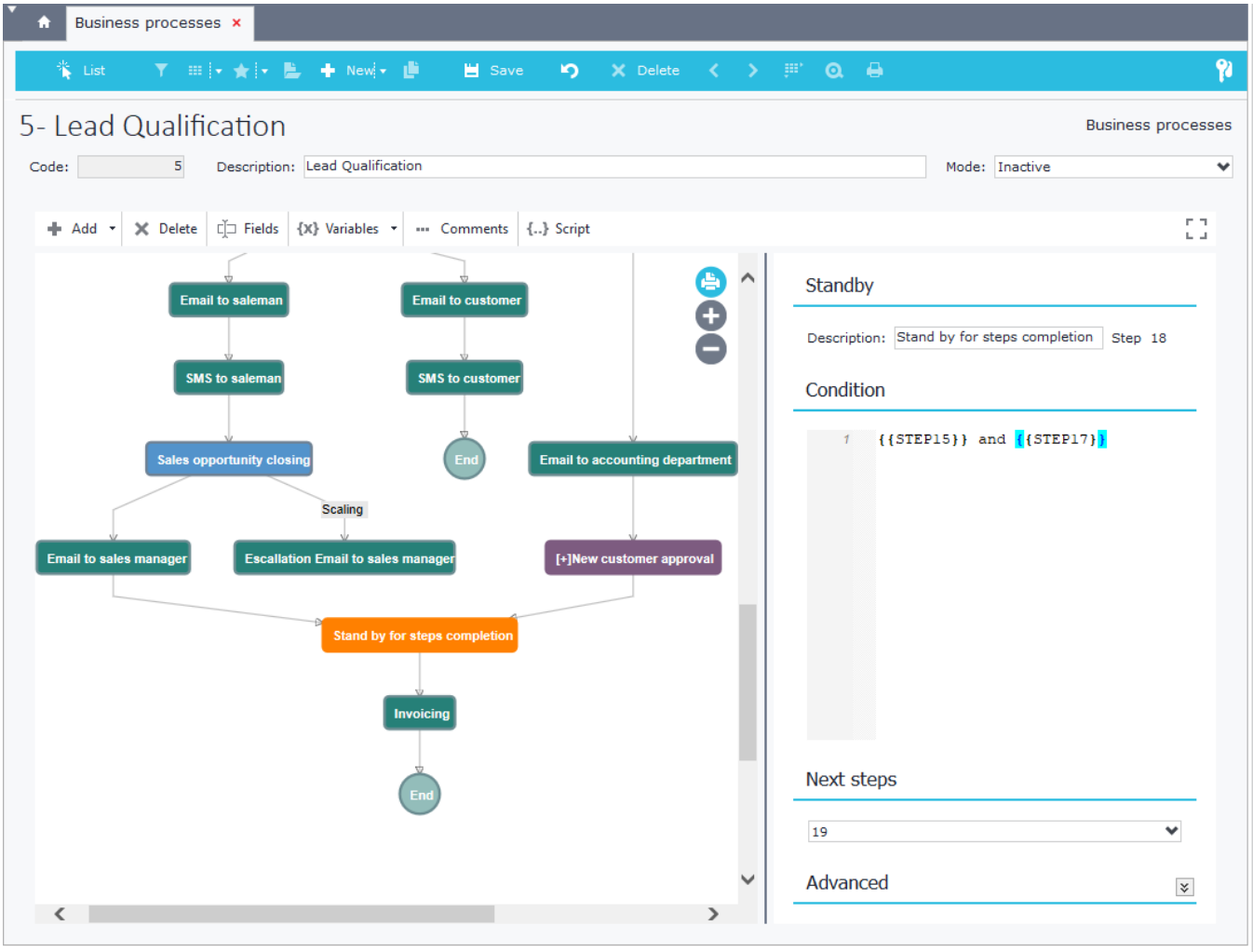


Figure E3.1

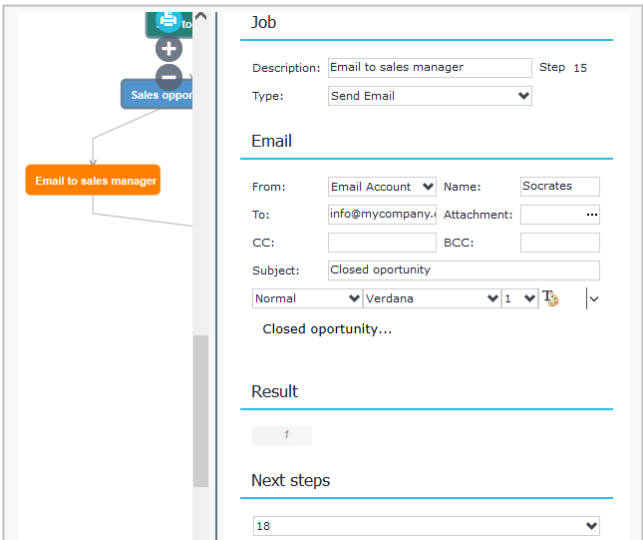


Figure E3.2

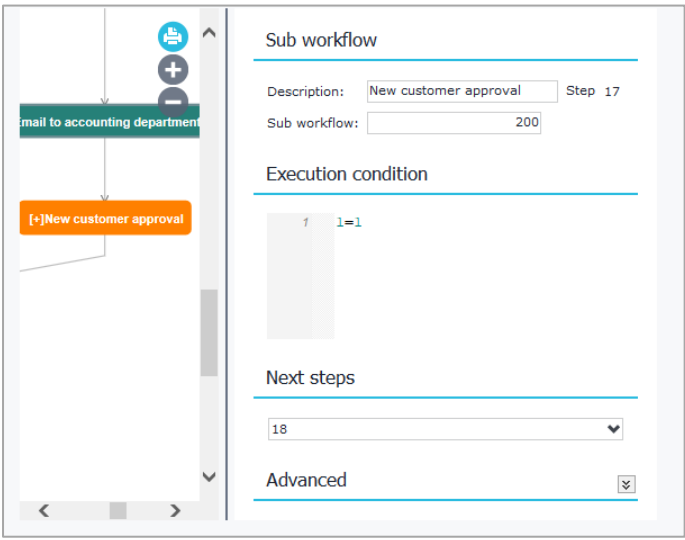



Figure E3.3

E.4 User Action

This step type is triggered when an entity record is posted by specific users or user groups and certain conditions are met. Steps that are created after User Actions offer two branches, one that is continuous and another that is used for escalation.

The id of the record is defined inside the textbox "Locate". Users or Groups are selected from the panel "Completed by" panel. Variables can be used for selecting users or groups if you click on the button  next to them.

The sub business process of Figure E4.1 shows a user action step that is triggered when User 1 or any user of the User Group 200, save the Customer with record id **{{PARENT.LCustomer[0].trdr}}** and the value of the field **{{CUSTOMER.ISPROSP}}**=0. The id **{{PARENT.LCustomer[0].trdr}}** is initialized in a "Job" step of parent business process 5 (Figure E4.2), which uses the variable "LCustomer" that retrieves data using the sql statement :

```
select trdr from sodraftlnk where sodraft={{DraftId}}
```

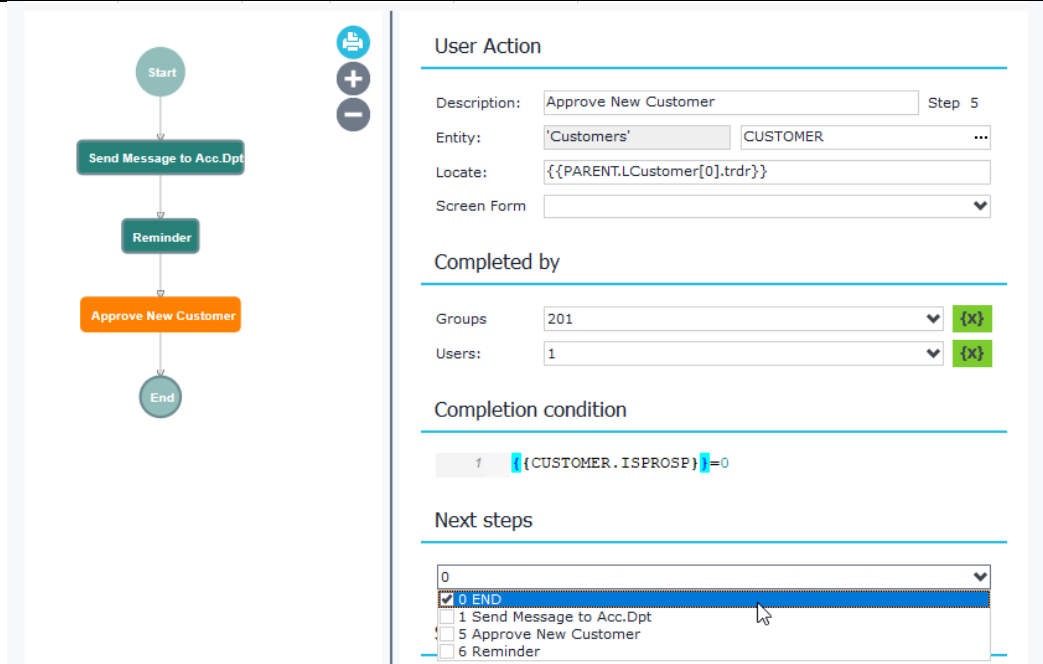


Figure E4.1

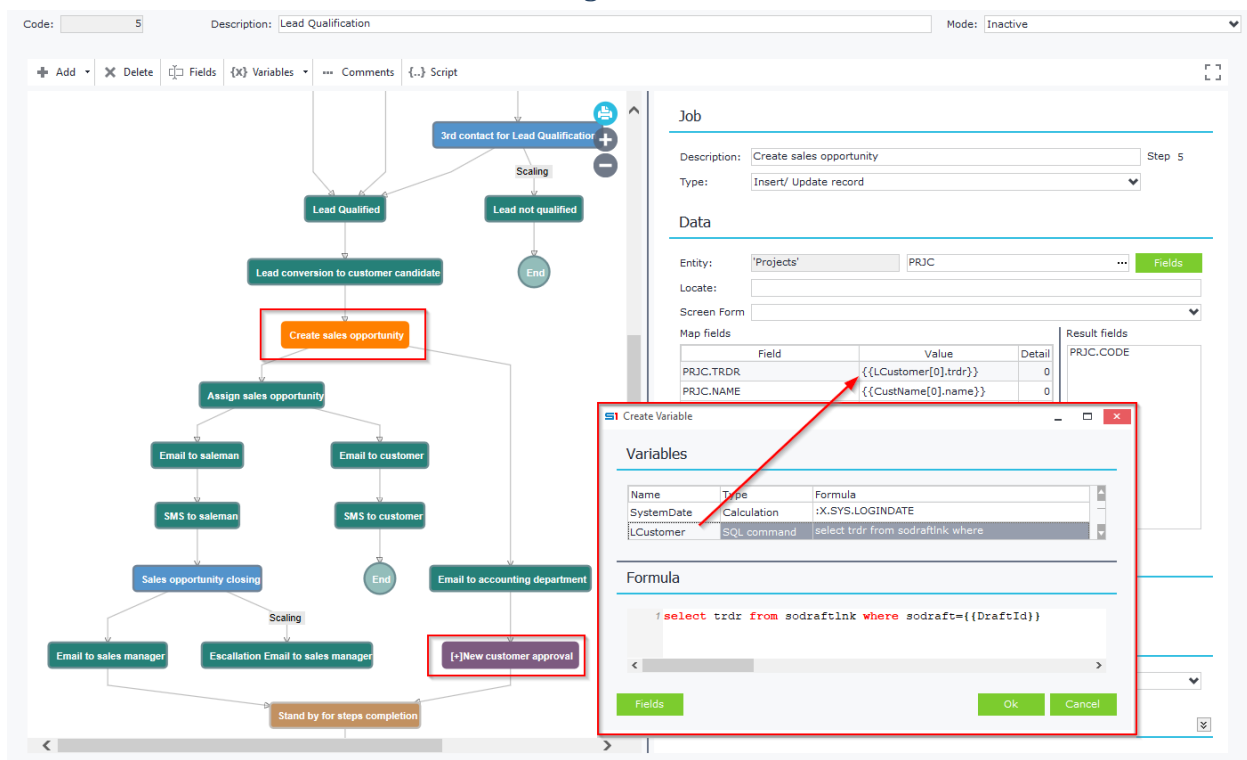


Figure E4.2

Escalation

Some actions (e.g. Order Approval) are time critical and need to be completed in a specific time period. Such actions can be managed through escalation functionality, that allows you to set up different process steps when the deadline is reached. Scaling time can be defined in minutes, hours or days.

In the following example (Figure E4.3) the process waits 1 day for User 262 to complete the Service Folder with id `{{SRVCARD.id}}`. Step is completed when certain condition is met (`SRVCARD.FINSTATES = 3 or 4 or 6`).

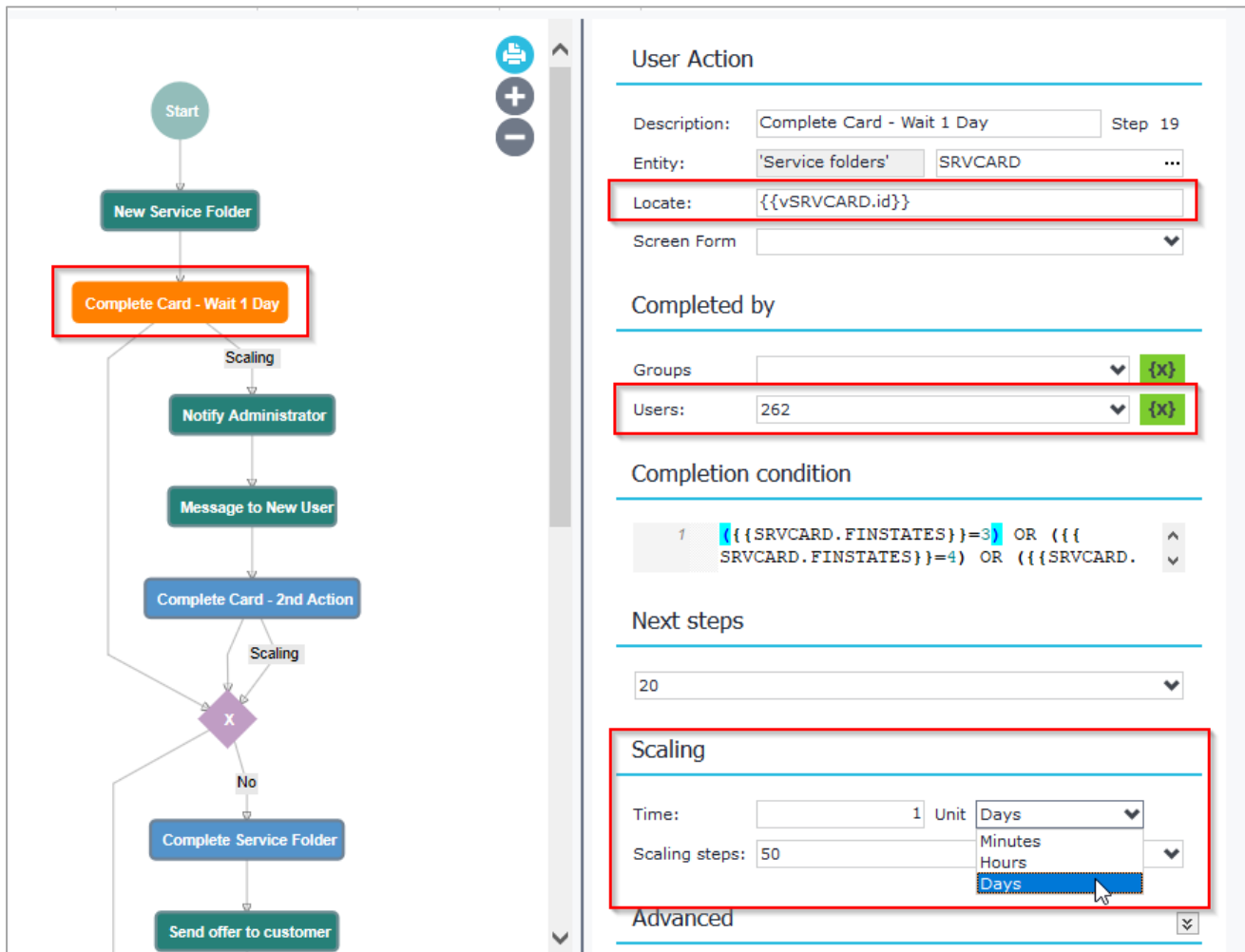


Figure E4.3

When deadline is reached an email is sent to notify a user that the record failed to complete and then sends a message to another user (Figure E4.4) so as to complete the action.

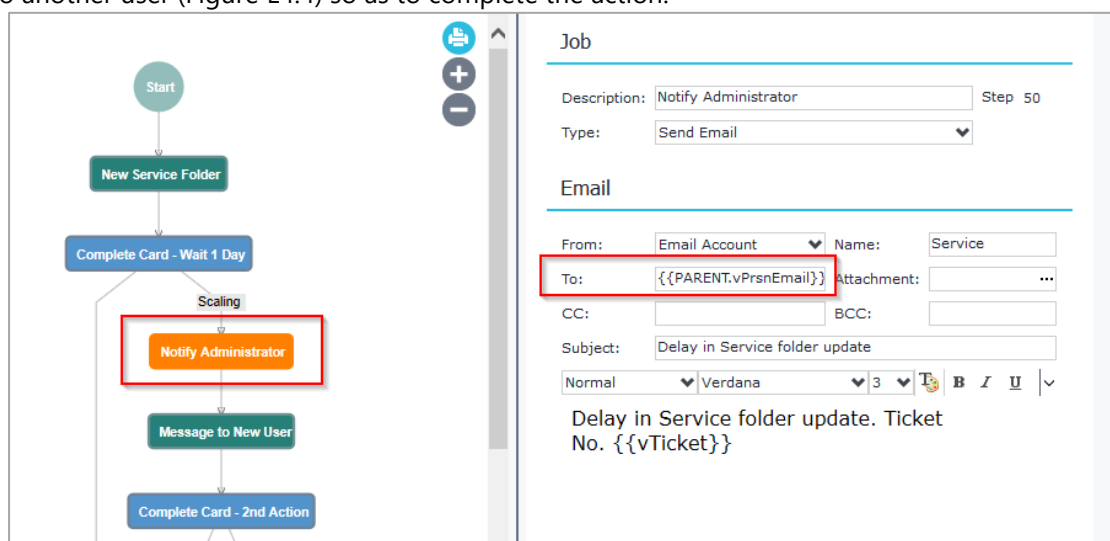


Figure E4.4

E.5 Job

This type of step is used for creating automated jobs that are executed in server-side. The different options that are available according to the type of job are discussed below.



E.5.1 Send email

Sends email to the given recipient using the email settings defined in the following table:

PROPERTY	DESCRIPTION
From	SoftOne e-mail account (Parameters – Users & Rights – Email Accounts).
Name	From Display Name.
To, CC, BCC	Recipient e-mail addresses split by semicolons. They can be hardcoded or selected from Start Entity Fields or defined in Variables.
Attachment	File that will be attached to the e-mail. Variables, that return the filename using the full path, can be used.
Subject	Text that will be added to the subject of the e-mail

The example of Figure E5.1.1 sends an e-mail to recipients that are defined in "SalesSecEmail" Variable.

Business processes

5- Lead Qualification

Code: 5 Description: Lead Qualification Mode: Design

Job

Description: Email to sales secretaries Step 11

Type: Send Email

Email

From: Email Account Name: BAM

To: {{SalesSecEmail[0].LIST}} Attachment: ...

CC: salesadmin@mycompany.com BCC: ...

Subject: New lead for qualification

Normal Verdana 3

New lead for qualification

Relative task: {{QTaskCode[0].soactioncode}}

Result

1

Next steps

0

Advanced

Figure E5.1.1

E.5.2 Send SMS

Sends sms to the given recipient using the following settings:

PROPERTY	DESCRIPTION
From	SoftOne SMS account (Parameters – Users & Rights – Email Accounts).
Number	Recipients’ phone number. It can be hardcoded or selected from Start Entity Fields or defined in Variables.
Message	SMS Text, with use of data defined in variables.

The example of Figure E5.2.1 sends SMS to Number 123456789 and the message contains data from Variables.

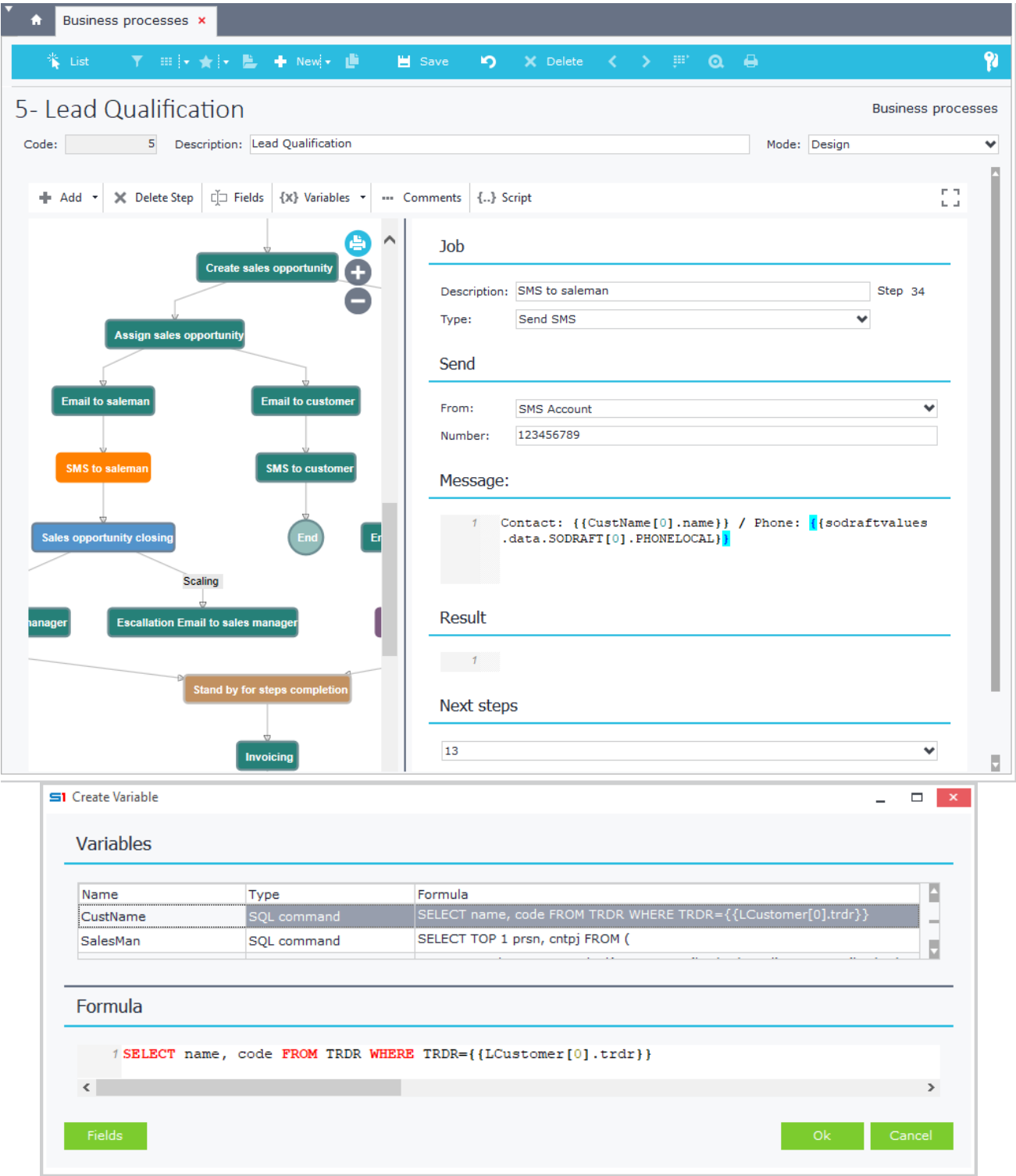


Figure E5.2.1

E.5.3 Run Job / Print

This type of process job is used for running SoftOne object jobs or printing reports.

E.5.3.1 Run Object Job

Select the object job to run, from the corresponding button "Select", enter the desirable template and the record Id. The result message can be assigned to a variable and used in a next step (e.g. send email).

The example of Figure E3.5.1.1 runs job "Convert Draft Entry" using the template named "WorkFlow" for the record `{{DraftId}}` that was previously initialized in the Start entity of the business process. The result is saved in the variable `{{BatchMsg}}` (Figure E3.5.1.2).

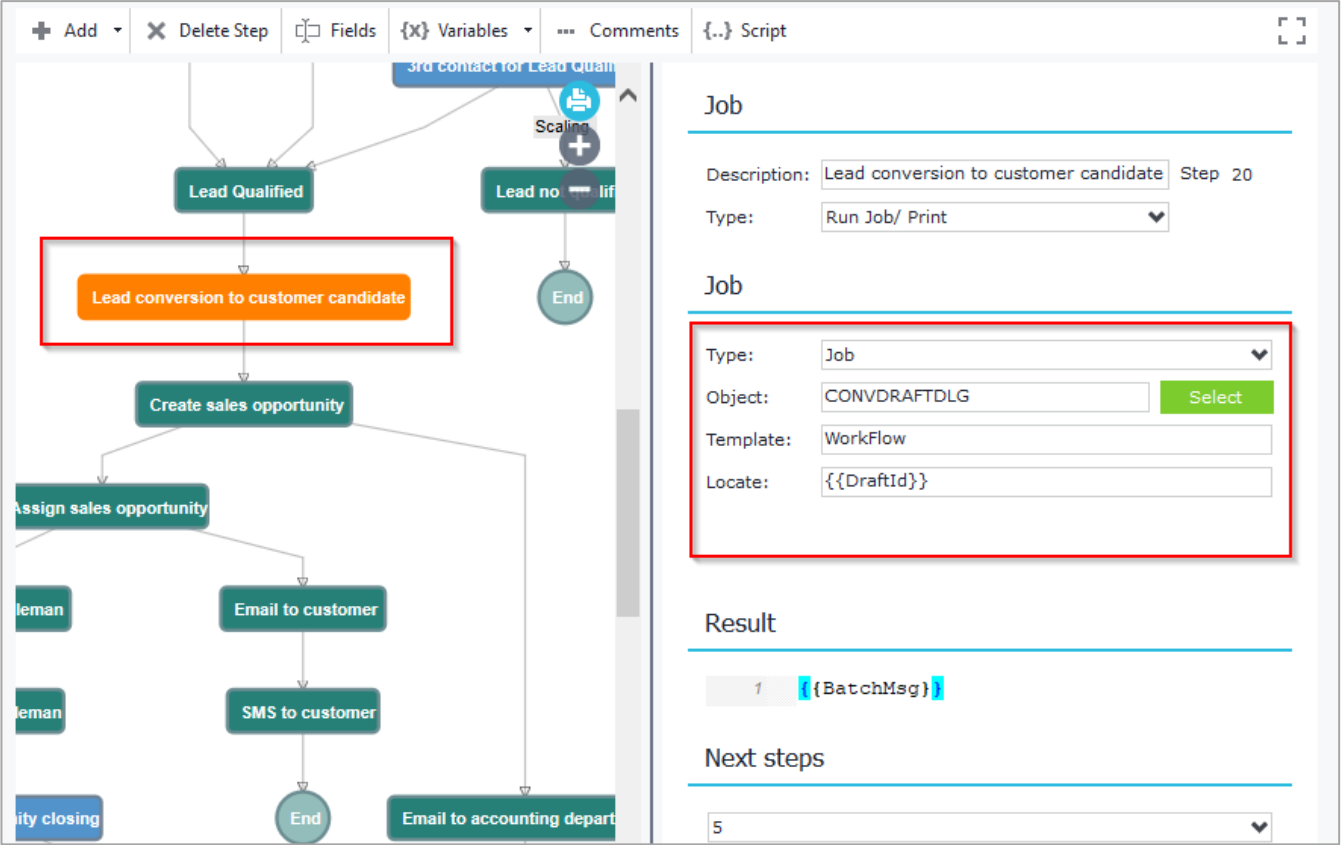


Figure E5.3.1.1

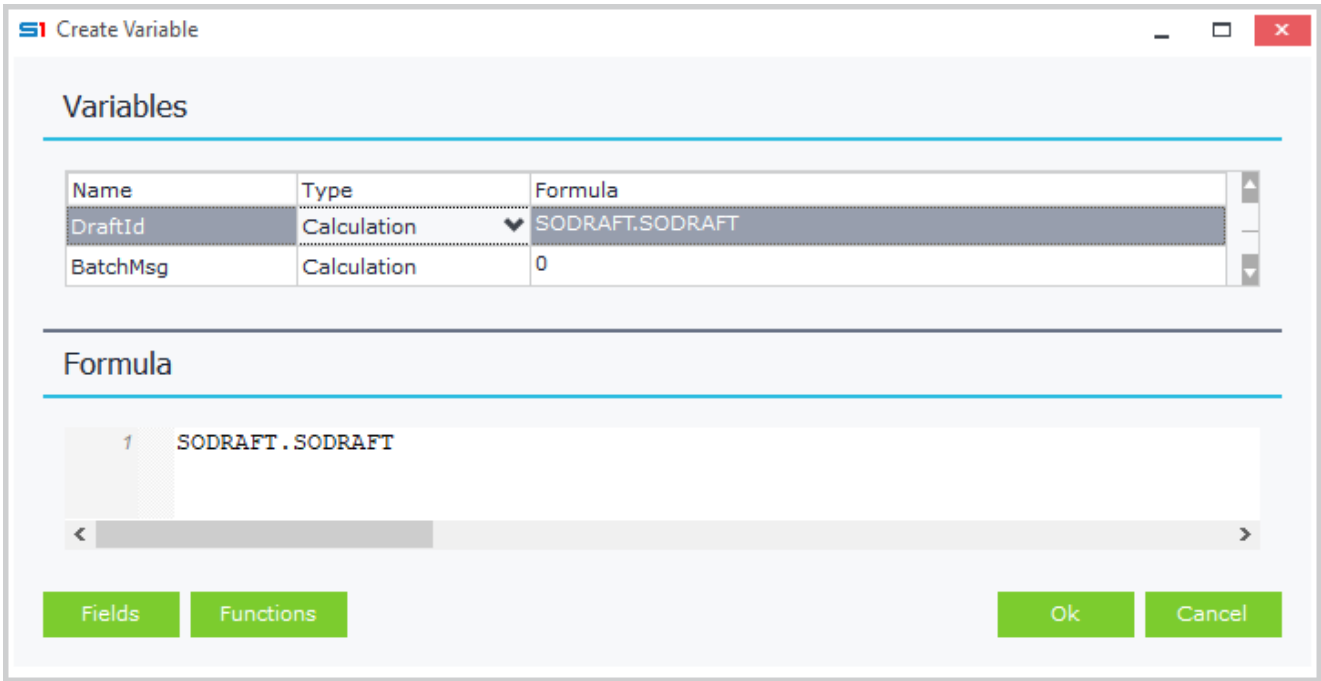


Figure E5.3.1.2

E.5.3.2 Print Report (Browser)

Prints the browser of the located object record. Select or enter the object inside “Object” textbox and then the record Id that will be printed (Locate) and the desirable browser template. Enter the filename (using variables) and select the File type from the list.

The example of Figure E5.3.2.1 prints the browser “Customers Balance” of Customers (CUSTOMER) object for the record {{vCustomerid}}, which is the id of the Sales Document posted in the “Start” step (SALDOC.TRDR) (Figure E5.3.2.2).

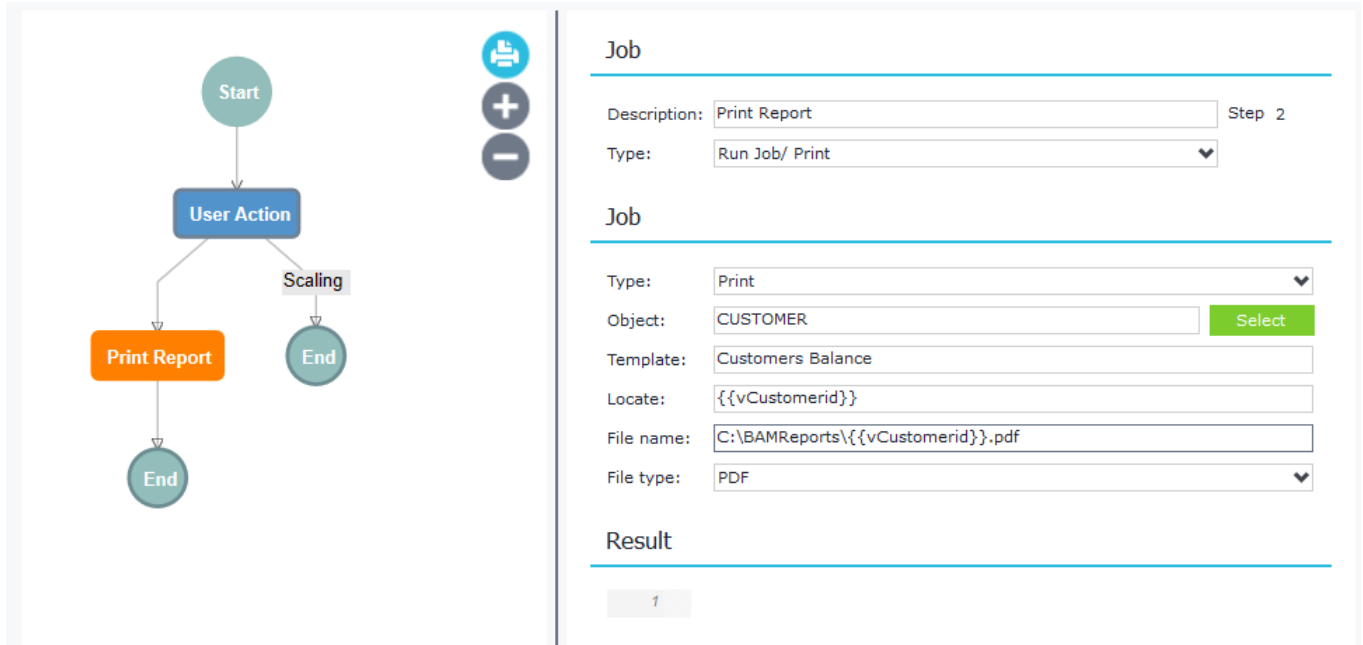


Figure E5.3.2.1

Name	Type	Formula
vCustomerid	Calculation	SALDOC.TRDR

1 SALDOC.TRDR

Figure E5.3.2.2

E.5.4 Insert / Update Record

This type of job uses the [SetData](#) post method for adding or editing object records.

The available options are the following:

PROPERTY	DESCRIPTION
Entity	SoftOne Object that will be used for insert or update. Select or enter the object name.
Locate	ID of the record. Leave it blank for inserting new records or enter here the id of the object that will be updated. Variables entered here are initialized.
Screen Form	Object Form. Select a form, when you need to execute extra business logic (through custom code) that exists in this specific form.
Map Fields	Map the target fields using hardcoded values or Fields from "Start Entity" (Toolbar button Fields) or Variables that were initialized in a previous step. Always keep in mind that "Start Entity" Fields are available for use in any step above the first "User Action" step.
Result Fields	Enter here the fields that will be returned when posting is completed. These fields will be added in the "Result" recordset under the "Data" node so as to use them in later steps of the business process..
Result	<p>Results are in JSON format and contain the data produced from SetData post method. Enter here a variable name that will be populated with the id of the record along with the result message and the result fields defined in the textbox "Result Fields".</p> <pre> { "success": true / false, "id": "xxx" "data": { "ResultTable1": [{ "ResultField1": "...", "ResultField2": "...", ... }] } }</pre> <p>The result fields are available for use in other steps, using the expression: {{ResultVariableName.data.TableName[0].FieldName}}.</p>

The example of Figure E5.4.1 updates a record of LINSUPDOC, setting the value of the field LINSUPDOC.APPRV to 1. The id that is used to locate the record is LINSUPDOC.FINDOC, which is the id of "Start Entity". In order to use the "Start Entity" id in other steps (even the ones below "User Action" we assign it to a variable (startfindoc) and initialize it here in the Locate statement (Figure E5.4.2).

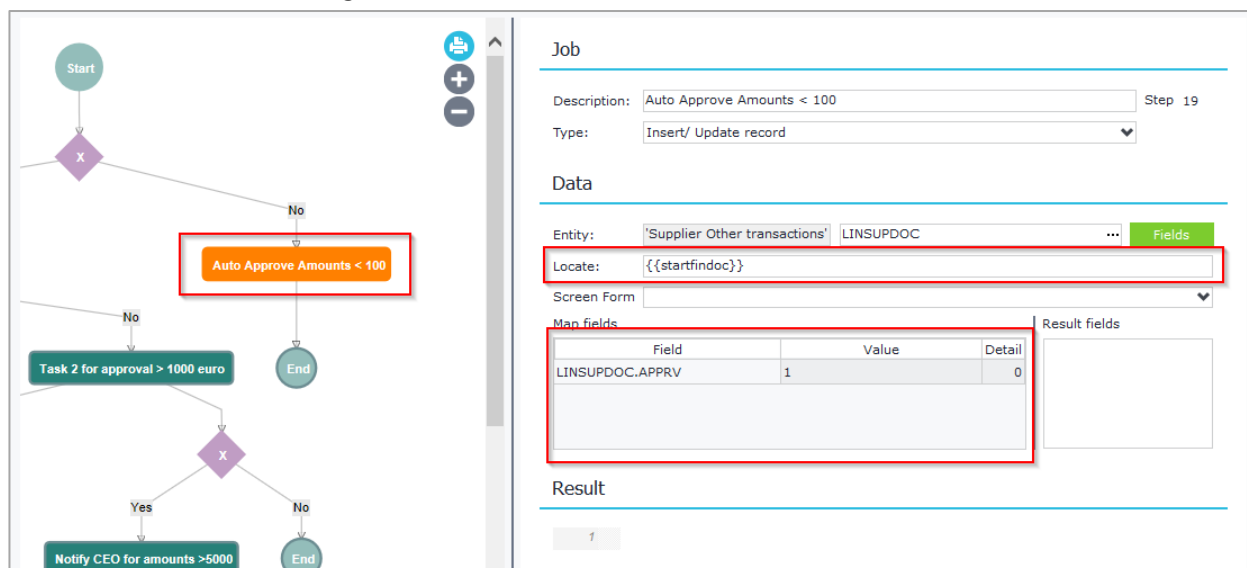


Figure E5.4.1

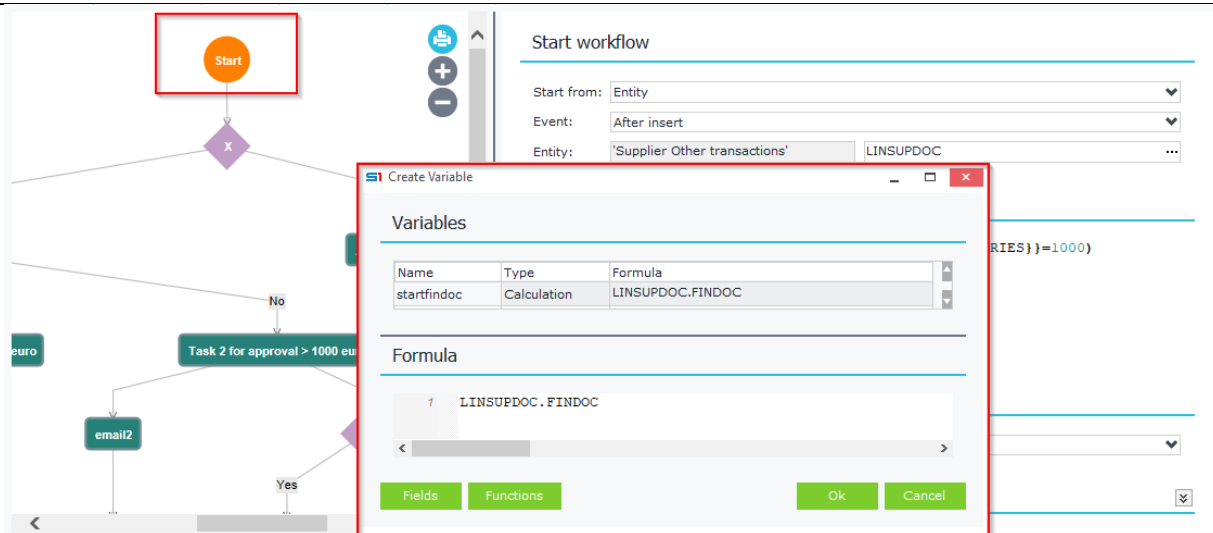


Figure E5.4.2

The example of Figure E5.4.3 insert a new task (Locate statement is blank) and the results are saved in the variable *Qtask*. It also adds the field *SOACTION.BRANCH* to the result variable, which is used in another step (Figure E5.4.4).

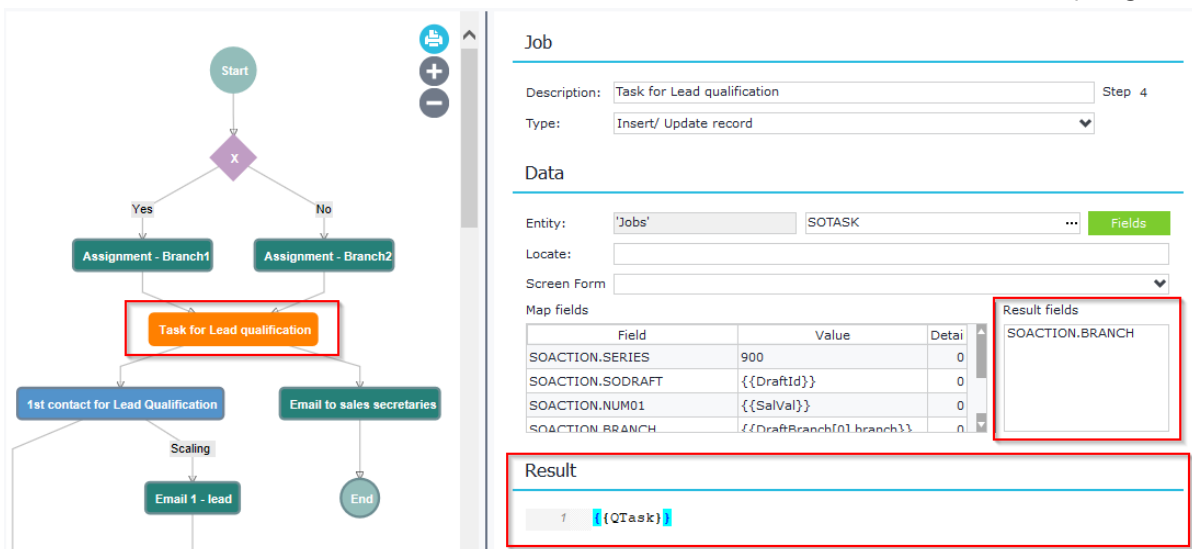


Figure E5.4.3

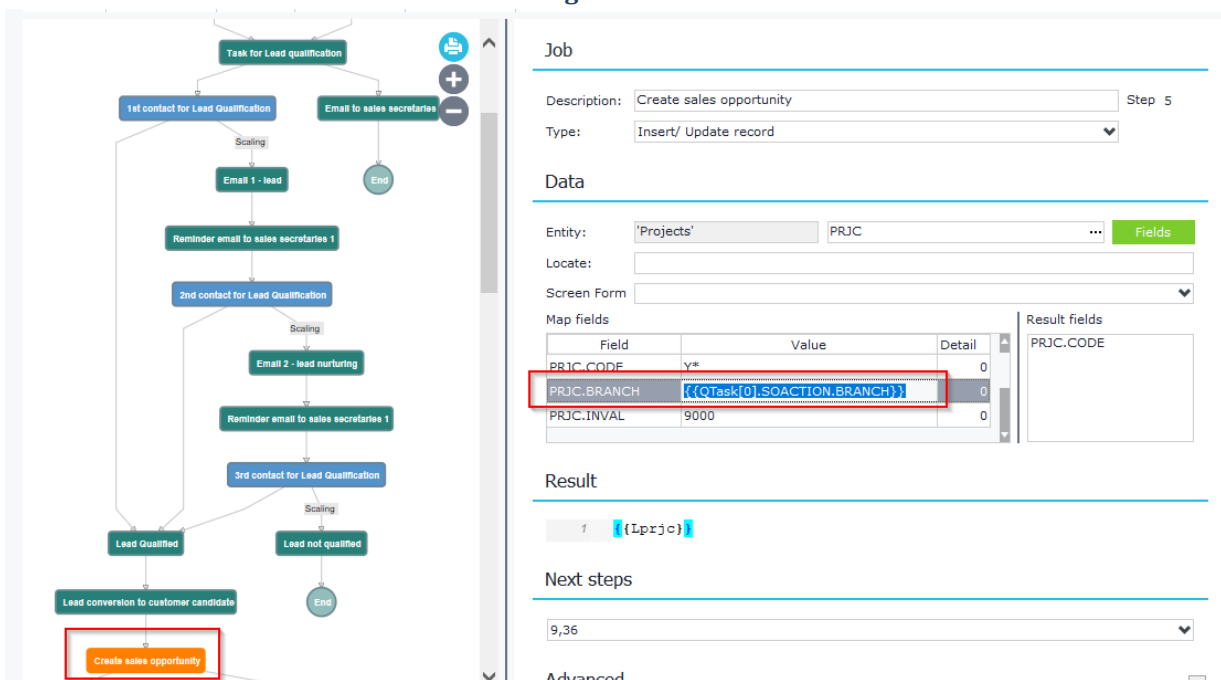


Figure E5.4.4

E.5.5 SQL command

This type of job step is used for executing sql select statements and storing the results in variables for later use in other steps. It also allows you to execute sql insert, update and delete statements.

It identifies the type of sql statement by the first word used, which means that it (internally) runs GetSQLDataset when the sql statement begins with "SELECT ..." (Figure E5.5.1), while on the other hand it runs EXECSQL when it begins with "INSERT" or "UPDATE" or "DELETE" (Figure E5.5.2).

Business processes

51- Work Permit Approval

Code: 51 Description: Work Permit Approval Mode: Inactive

Job

Description: Get Employee Data Step 1

Type: SQL command

SQL command

```
1 SELECT CODE, NAME, NAME2 FROM PRSN
2 WHERE PRSN={{EmployeeId}}
```

Result

1 {{EmployeeData}}

Next steps

6

Advanced

Figure E5.5.1

Job

Description: Update Approval user Step 7

Type: SQL command

SQL command

```
1 Update soaction set SOACTION.APRVUSER = {{
AprvPrsnData[0].users}} where soaction = {{
AprvTask1.id}}
```

Result

1 {{next}}

Next steps

1

Advanced

Figure E5.5.2

E.5.6 Print form

This type of step is used for printing object forms. Select or enter the object inside "Entity" textbox and then the record Id that will be printed (Locate) and the desirable print-out form template. Job returns the filepath of the printed file and you can assign it in a variable for use in other steps (Result).

The example of Figure E5.6.1 prints the form 1001 of Sales Documents (SALDOC) for the record {{newsaldoc}}, which is the id of the Sales Document posted in the "Start" step (SALDOC.FINDOC). The result filename is assigned to variable {{formfilepath}} (Figure E5.6.2).

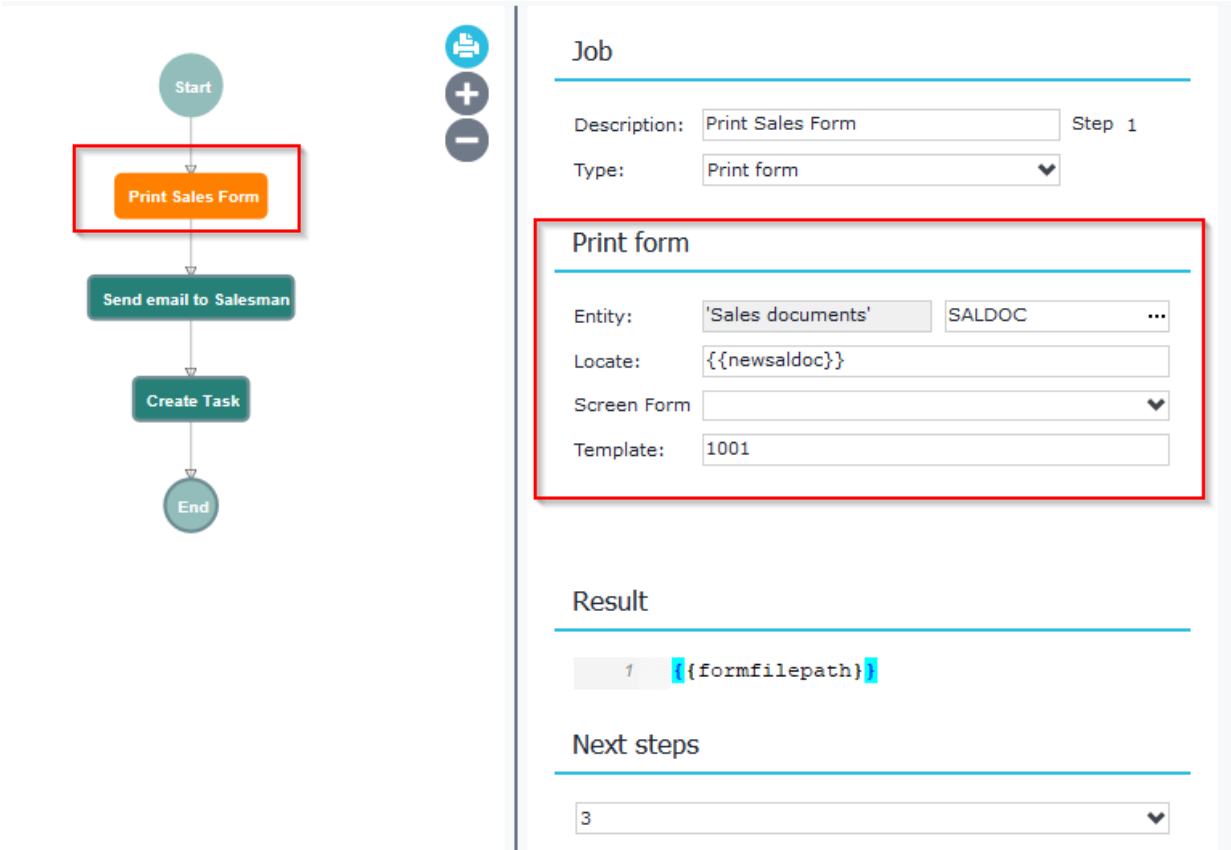


Figure E5.6.1

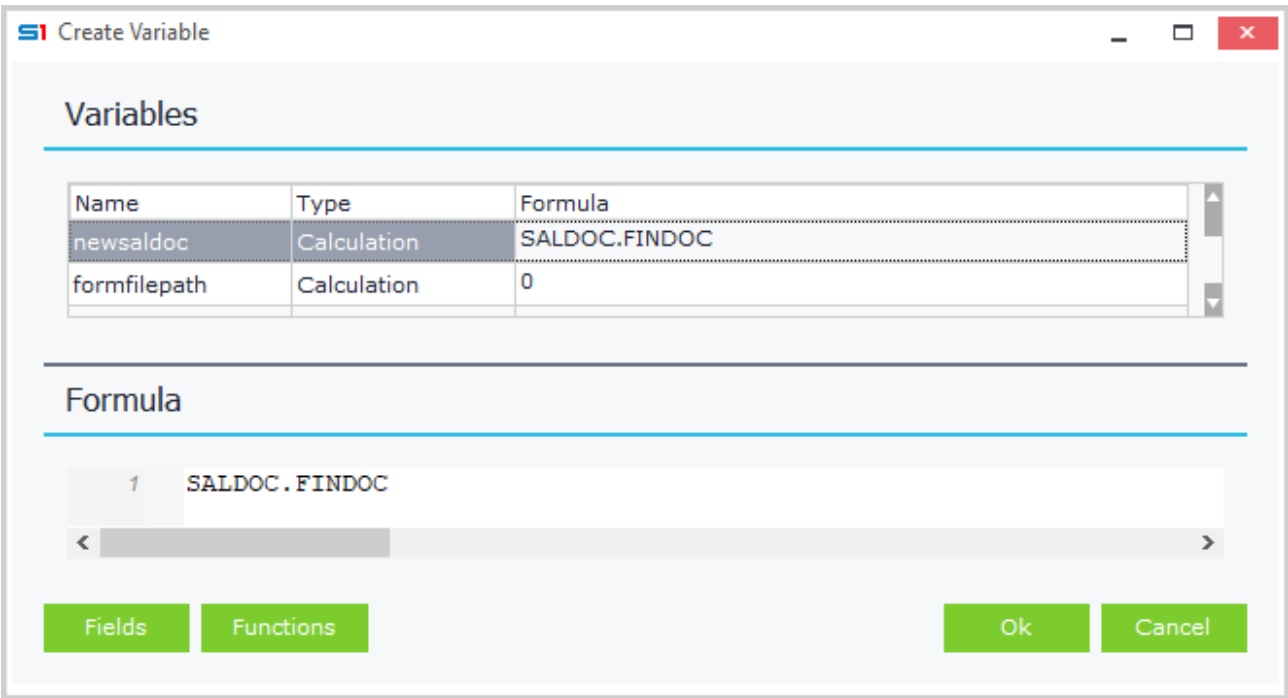


Figure E5.6.1

E.5.7 Send message

Sends a SoftOne message to the given users or groups using the following settings:

PROPERTY	DESCRIPTION
To	Users or Groups
Select	Select users or groups or click on button "{x}" to enter a variable that defines them.
Message	Message text, where you can use Variables.
Attachments	Adds linked records of given entity.

Figure E5.7.1 shows an example of message sent to user 2 that contains message using data from parent process. It also adds an attachment that contains the redirecton link of a customer record (Figure E5.7.2).

Start

Message to Accounting Dpt

Reminder

Approve Customer

End

Job

Description: Message to Accounting Dpt Step 1

Type: Send message

Send

To: Users

Select: 2 {x}

Message

Attachments

1 New Customer for approval.

2 {{PARENT.CustName[0].code}} {{PARENT.CustName[0].name}}

Figure E5.7.1

Start

Message to Accounting Dpt

Reminder

Approve Customer

End

Job

Description: Message to Accounting Dpt Step 1

Type: Send message

Send

To: Users

Select: 2 {x}

Message

Attachments

Description	Entity	Locate	Company
New Customer	CUSTOMER	{{PARENT.LCustomer[0].trdr}}	1000

Figure E5.7.2

E.5.8 Create reminder

Creates a SoftOne reminder using the following settings:

PROPERTY	DESCRIPTION
To	Users or Groups
Select	Select users or groups or click on button "{x}" to enter a variable that defines them.
Reminder Time	Reminder time in hours, minutes or days.
Related entry	Adds a link to the reminder that contains the record of the "Locate" property of the object defined in "Entity" property.

The example of Figure E5.8.1 creates a reminder that adds a link of the Service folder record {{SRVCARD.id}}, which is the result of the Service folder added in step 1 - "New Service Folder" (Figure E5.8.2).

The figure shows a job configuration interface. On the left is a flowchart: Start → New Service Folder → Complete Card - Wait 1 Day → Scaling → Notify Administrator → **Reminder to Users** (highlighted in a red box) → Complete Card - 2nd Action. On the right is the configuration panel for the 'Reminder to Users' job (Step 59). The 'Job' section has Description: 'Reminder to Users' and Type: 'Create Reminder'. The 'Send' section has To: 'Users' and Select: '1,2,3,211,212' with a '{x}' button. The 'Reminder' section has Time: '2' and Unit: 'Hours'. The 'Related entry' section (highlighted in a red box) has Entity: 'SRVCARD' and Locate: '{{vSRVCARD.id}}'.

Figure E5.8.1

The figure shows a job configuration interface. On the left is a flowchart: Start → **New Service Folder** (highlighted in a red box) → Complete Card - Wait 1 Day → Scaling → Notify Administrator → Reminder to New User → Complete Card - 2nd Action → Scaling. On the right is the configuration panel for the 'New Service Folder' job (Step 1). The 'Job' section has Description: 'New Service Folder' and Type: 'Insert/ Update record'. The 'Data' section has Entity: 'Service folders' and SRVCARD, with a 'Fields' button. The 'Map fields' table is as follows:

Field	Value	Detz
SRVCARD.NUM01	{{PARENT.vEmailAnsw	0
MTRDOC.MTRL	{{PARENT.vItem}}	1
MTRDOC.SNCODE	{{PARENT.vSN}}	1

The 'Result' section (highlighted in a red box) shows a table with one row: 1 | {{vSRVCARD}}.

Figure E5.8.2

E.5.9 HTTP Request

This type of Job is used for making HTTP calls using either get or post method. Select the method and enter the URL. Then add the Headers, if any, and the Body of the call for POST requests. The answer can be saved in a variable for later use in other steps of the business process.

In the following example (Figure E5.9.1) we establish a login request to SoftOne demo database using a POST method. The results are saved in {{vResults}} variable (Figure E5.9.2).

Job

Description: Step 2 Step 2

Type: HTTP Request

Http Request

Method: POST URL: http://demo.oncloud.gr/s1services

Headers

KEY	VALUE
Content-type	application/json
Accept	application/json

Body

```
1 {
2   "service": "login",
3   "username": "MyUser",
4   "password": "123456",
}
```

Result

1 {{vResults}}

Next steps

0

Figure E5.9.1

Create Variable

Variables

Name	Type	Formula
vResults	Calculation	0

Formula

1 0

Fields Functions Ok Cancel

Figure E5.9.2

E.5.10 Data Flow Rule

This type of job step is used for running SoftOne data flow rules that don't have interface (dialog filters and form display). It's very useful when you need to create more than one records in a single step using the data of one record of an object. This is accomplished because data flow rules allow multiple create of records using grouped data (e.g. create as many records as the number of the different lines of a single Sales Document).

Design is very easy, as you only need to define the object name inside "Entity" textbox, the record Id that will be used to run the rule (Locate) and finally the name of the rule that will be executed.

The following example shows a business process that is activated when a meeting with Series 105 is inserted (Figure E5.10.1). In the next step it waits for users to complete the meeting using a specific condition (Figure E5.10.2) and then it automatically creates a Sales Document using the data flow rule "Create Sales from Meeting" (Figure E5.10.3). The locate statement of the data flow rule {{SOACTION.SOACTION}} is the id of the meeting of the first step.

Figure E5.10.1

Figure E5.10.2

Job

Description: Step 2

Type:

Data Flow Scenario

Entity: ...

Locate:

Rule:

Result

Next steps

Advanced

Figure E5.10.3

The Data Flow rule is created as in Figure E5.10.4. The source entity is Meetings (SOMEETING) and the target the Sales Documents (SALDOC).

1094- Create Sales from Meeting Data flow rules

Description:

Data **Run**

Source entity

Name:

Description:

View:

Data tables **Modifications**

General Actions

1 of 1

Target entity

Name: View:

Description: Tables:

Transactions (Sales) **Lines of services**

Data source:

Assign field values

Field	Expression	Aggregation
Series	7021	None
Customer	{SOACTION.TRDR}	None

Figure E5.10.4

E.6 Sub Process

This type of step is used to trigger a sub business process. Apart from the design that it is the same as any other business process, parameters can be sent from parent to child processes. Nested business processes are also displayed in different graphical modelling view at runtime.

Figure E6.1 shows an example that triggers the Sub process 200, which uses a value from the parent process `{{PARENT.LCustomer[0].trdr}}` to locate the Customer record in a User Action step (Figure E6.2).

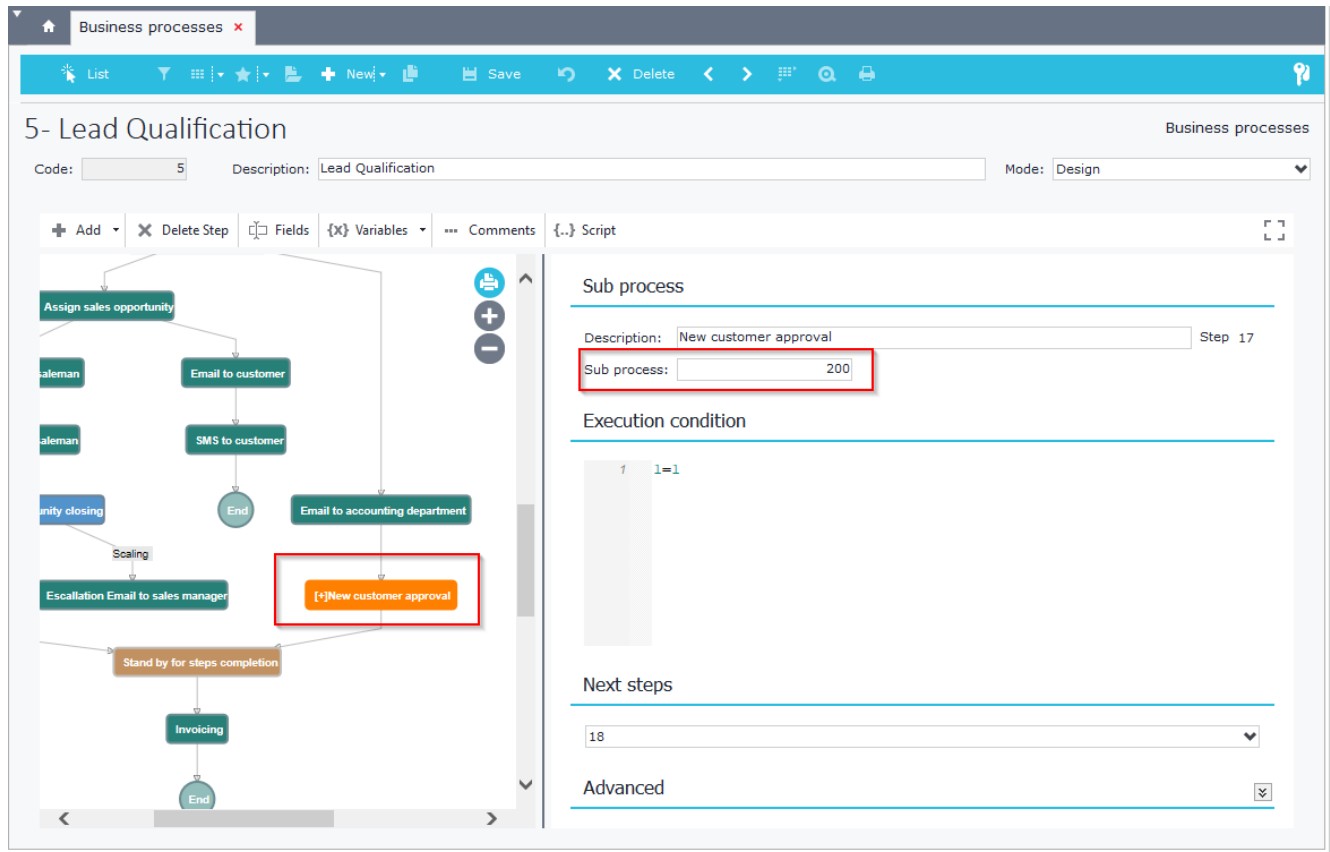
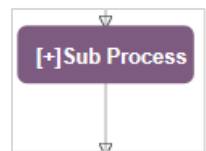


Figure E6.1

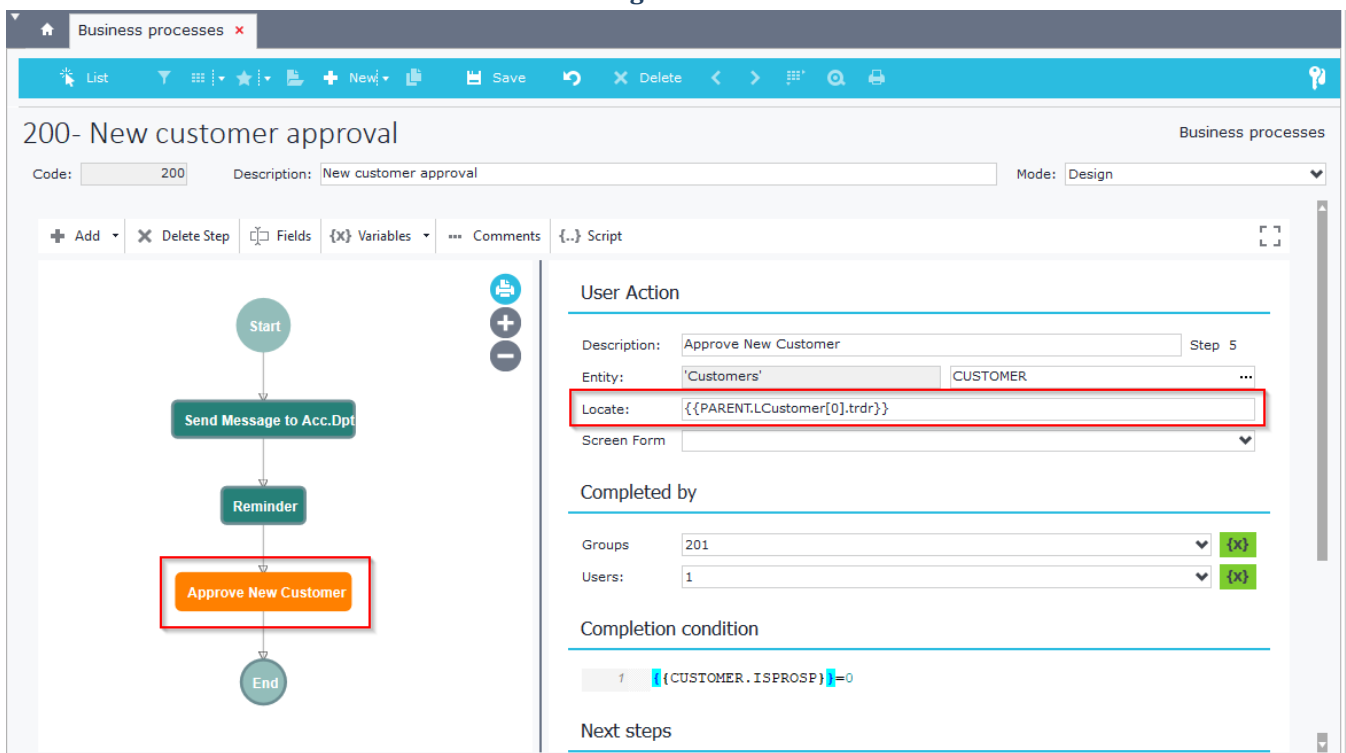


Figure E6.2

F. Runtime

At runtime, BAM offers visual view of the active processes per object record. This is available in all SoftOne objects through the related job “Business Process” (Figure F1). BAM steps are processed asynchronously from server, so there is no delay when users process records that trigger workflow steps.

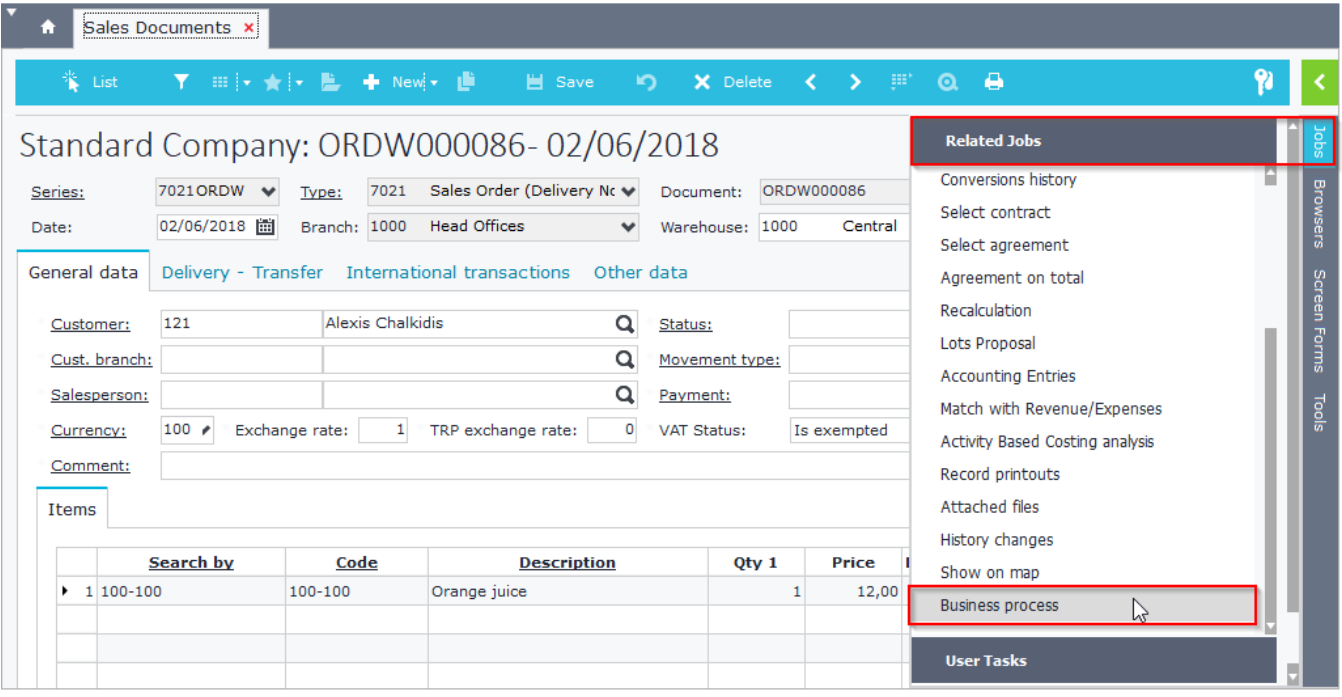


Figure F1

The business process diagram of an object record shows the steps that are completed in green color and the pending ones in orange (Figure F2). Jobs that are executed while you open the Business process Status diagram can be refreshed using the “Refresh” button. The grid of the right panel shows detail info about the workflow.

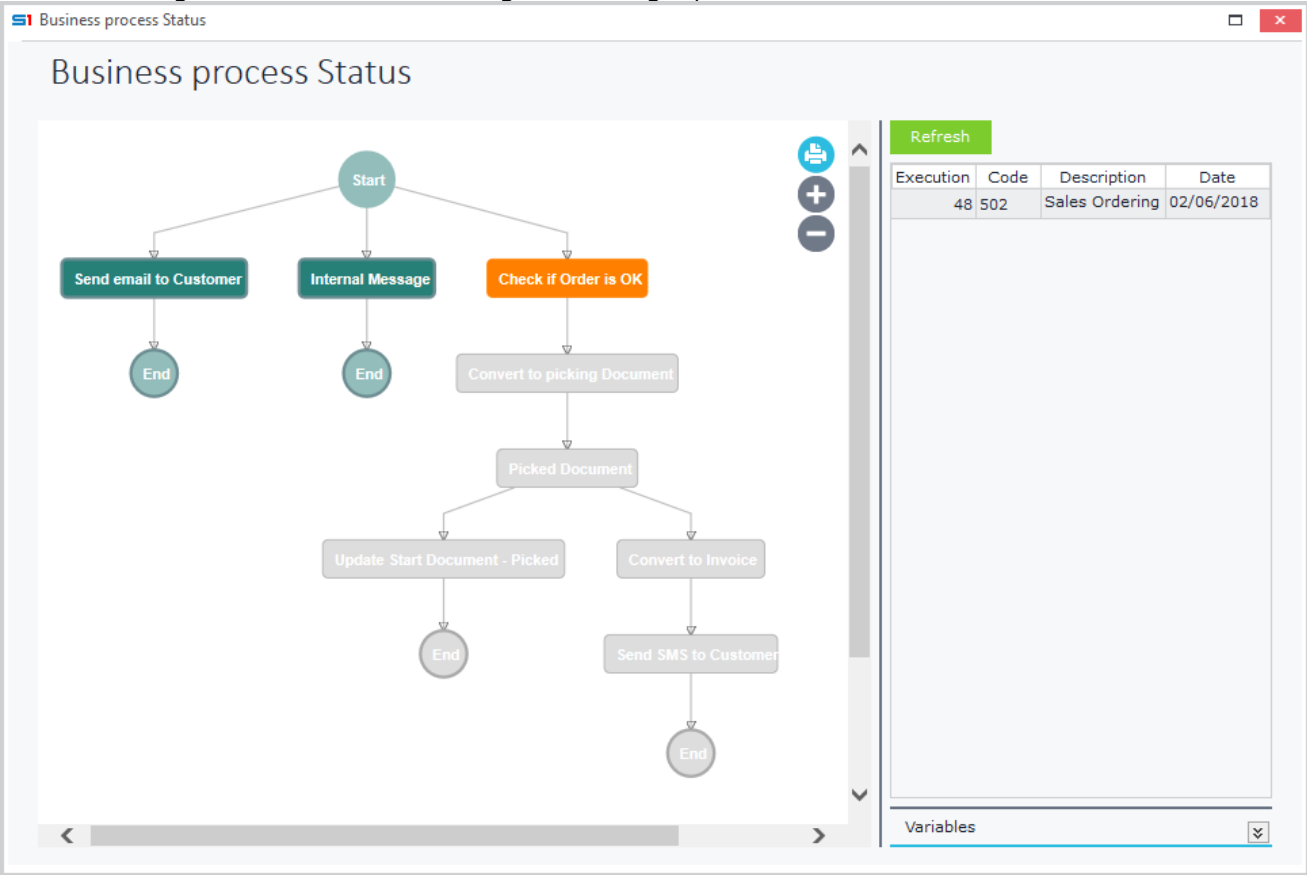


Figure F2

Variables panel shows analysis of the calculated variables of the selected step (Figure F3). This is very useful for debugging the business process and checking that the variables created inside the design of the workflow have the correct values. Steps that fail to process are colored red, and the fail reason is inserted inside **"Variable"** panel. The Step "Picked Document" of Figure F4 failed because in Design mode the locate statement of the job was blank and the application could not locate the sales document in order to convert it and complete the step.

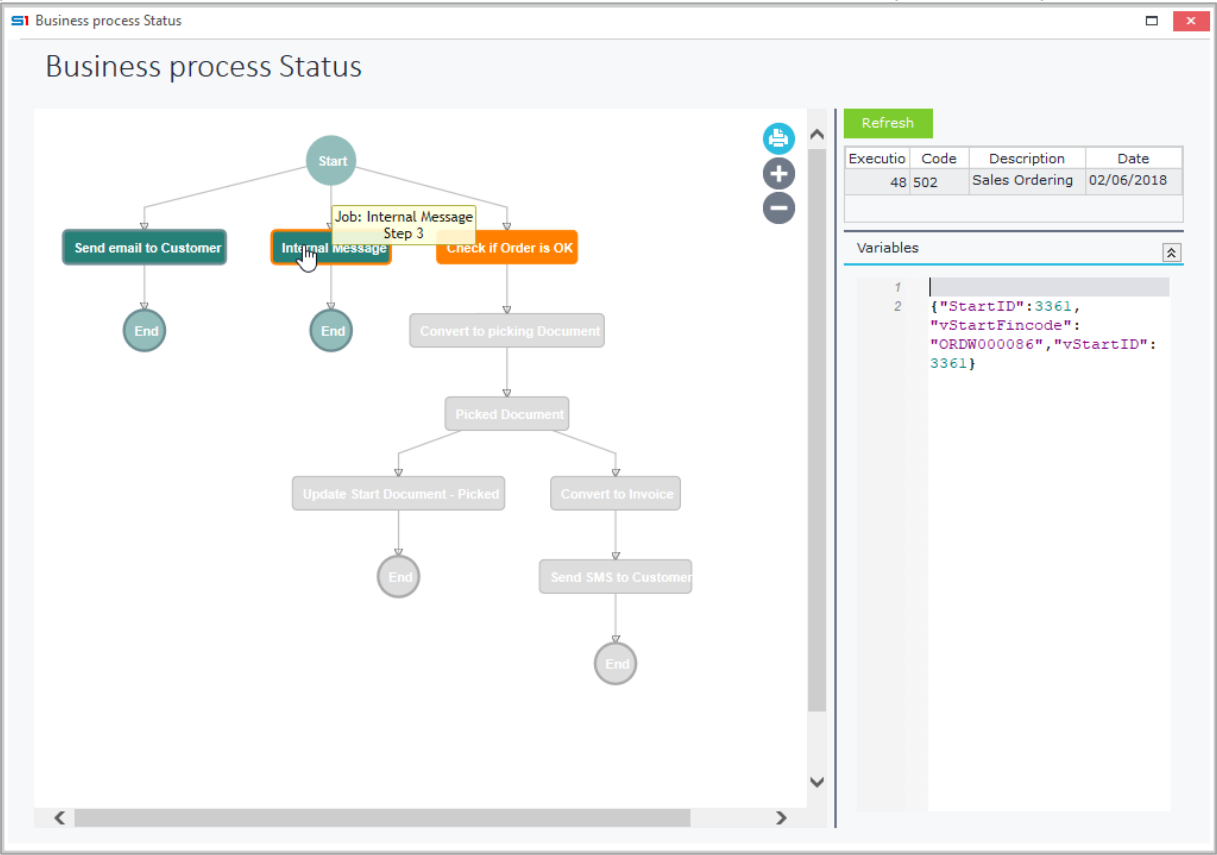


Figure F3

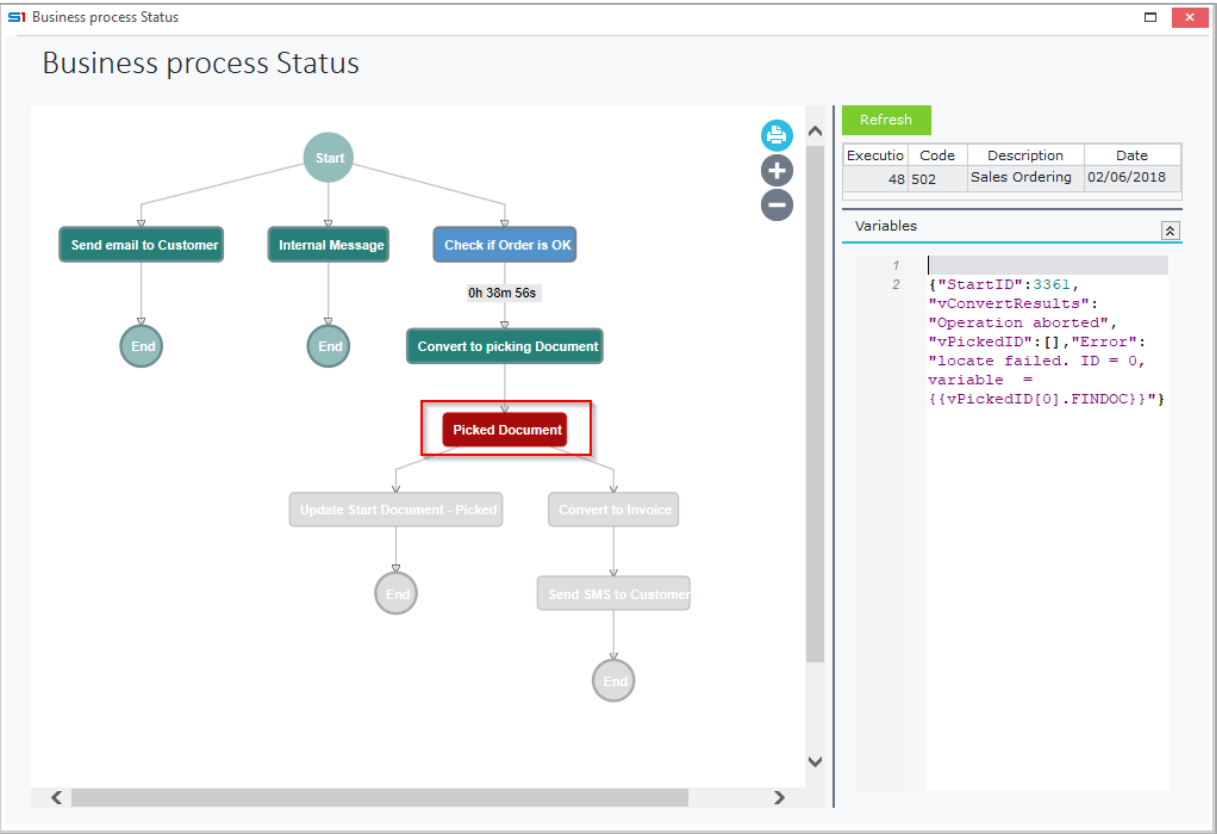


Figure F4

G. Audit & Reports

G.1 Business Process Management

This entity keeps a log file of all business process records that were triggered and are pending or executed. It lists the steps, their execution time, the user and the insert/end date per business process. It also shows the status of the step, that is also available in Browser filters (Figure G1.2).

Right Click on a record (Figure G1.1) gives you the following options:

- Drill-down to the **related entry** of the step
- **Retry** the **step execution** if the process of the selected step has failed to run
- View the whole **business process** in the modelling view Window.

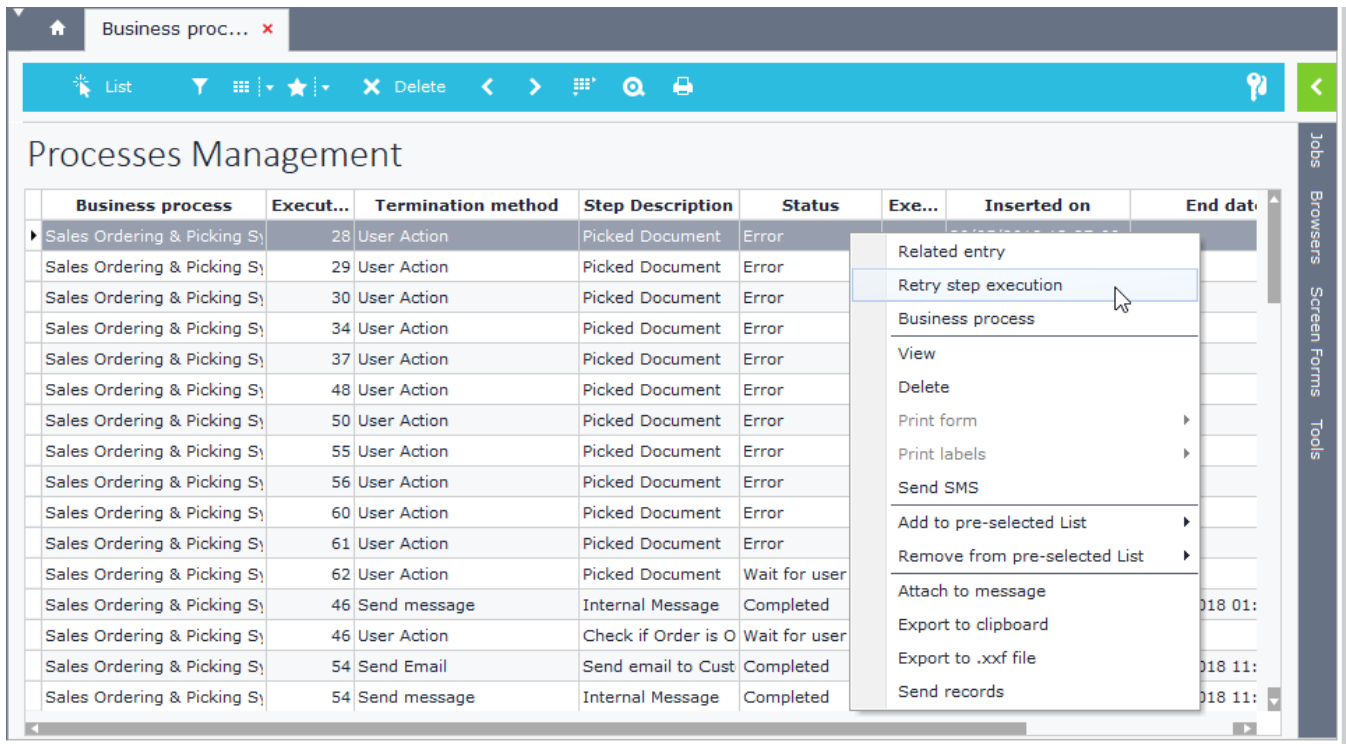


Figure G1.1

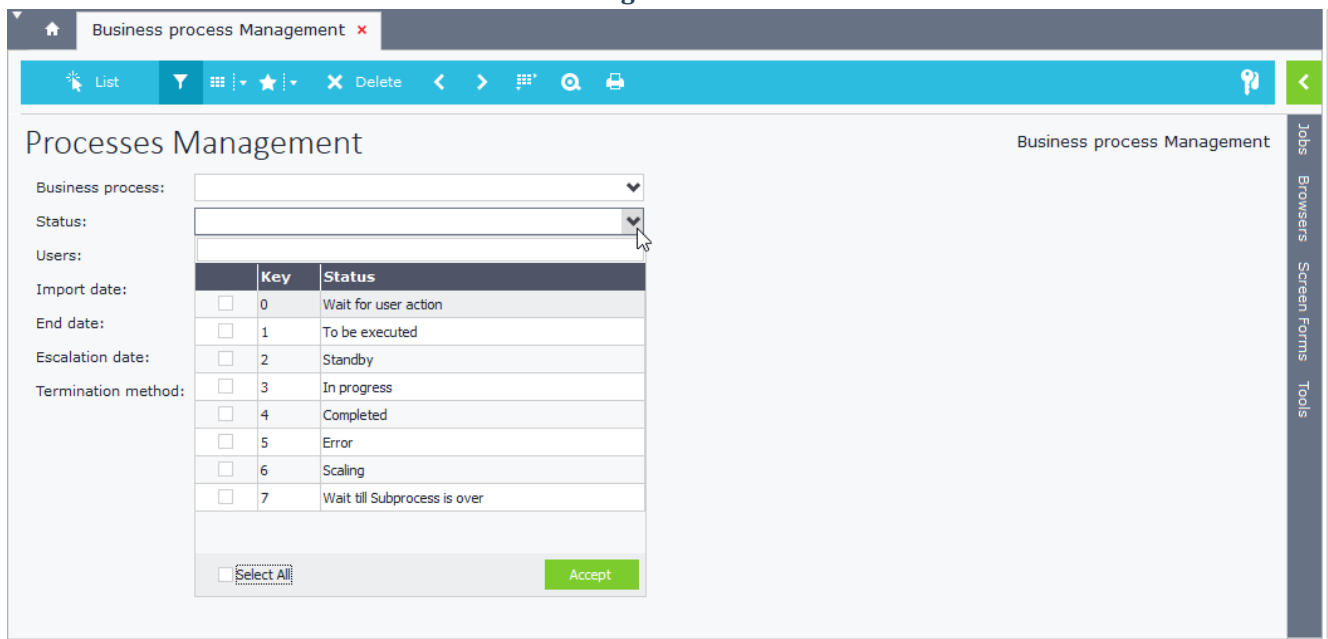


Figure G1.2

G.2 Business Process Analysis

This Dashboard shows graphical analysis of business processes execution time per job and users (Figures G2.1 & G2.2).

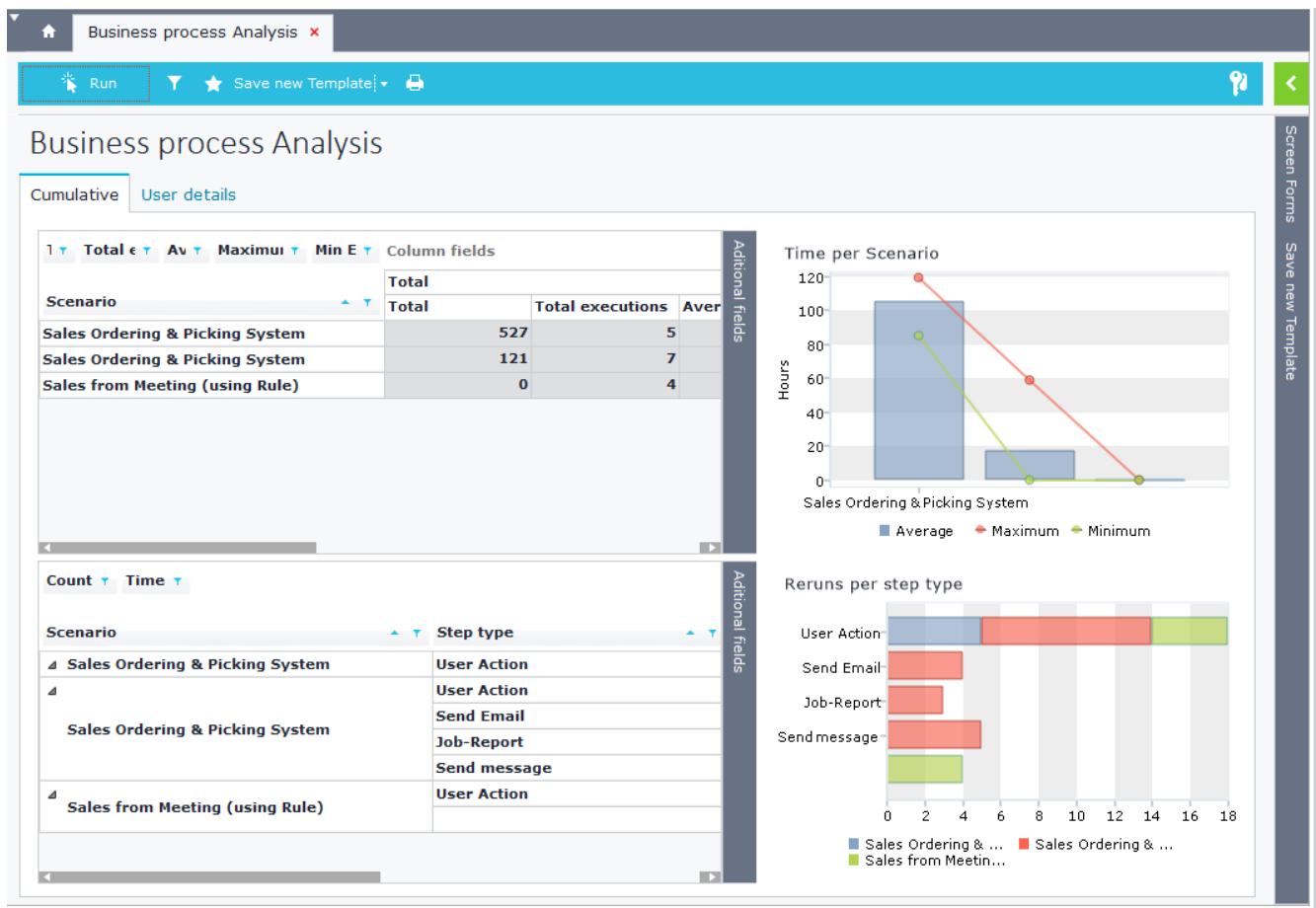


Figure G2.1

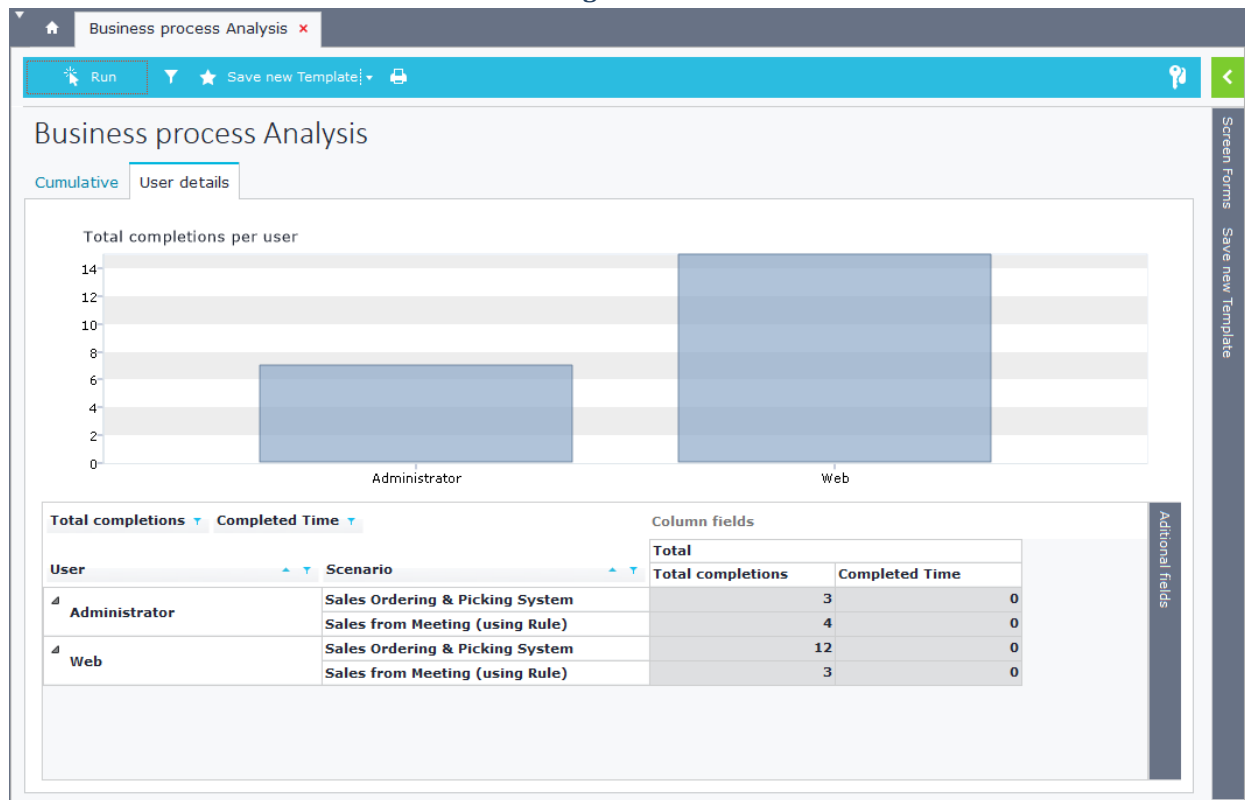


Figure G2.2

APPENDIX

SYSTEM PARAMETERS & COMMANDS

- A. [SODTYPE – Entity Values](#)
- B. [SOSOURCE – Module Values](#)
- C. [ORIGIN – Transaction Source Values](#)
- D. [CSTTYPE \(CSTINFO Table\)](#)
- E. [X.SYS – System Parameters](#)
- F. [ACMD Commands – System Tools](#)
- G. [Editor Commands](#)
- H. [Editor Attributes](#)
- I. [Built-in Editors](#)
- J. [Related Job Commands](#)
- K. [Browser Job Commands](#)
- L. [Command Line Switches](#)
- M. [XCO: On-Premise Connection Settings](#)
- N. [PARAMS.CFG: Cloud Connection Settings](#)
- O. [Internal Object Parameters](#)

A. SODTYPE – Entity Values

The following table contains all the values that can be found in the internal string list SODTYPE.

SODTYPE List			
Code	Module	Code	Module
10	Internal use only	44	Warranties
11	Company	51	Inventory
12	Suppliers	52	Services
13	Customers	53	Debits/Credits
14	Cash accounts	54	Fixed assets
15	Debtors	61	Revenue/Expense accounts
16	Creditors	70	Item sets
20	Company employees	71	Bills of materials
21	Contacts	72	Debits/Credits set
22	Draft Entry	73	Service sets
25	Resources	81	Cheques
28	CRM Questionnaires	82	Costing
29	HR Questionnaires	83	Gift vouchers
30	Salespersons	84	Bonus cards
31	Collectors	89	General ledger
32	Buyers	90	Cost accounting
33	Technicians	91	ABCCosting
40	Projects	95	Payroll
41	Installations	96	Actions
42	Special contracts	97	Call centers
43	Geographic points	98	B2B service

B. SOSOURCE – Module Values

The following table contains all the values that can be found in the internal string list SOSOURCE.

SOSOURCE List			
Code	Module	Code	Module
1010...1019	Custom business documents	1352	Service folders
1054	Depreciation documents	1353	Special transactions – Customers
1089	G.L. entries	1361	Revenue documents
1090	C.A. entries	1381	Cash transactions - Customers
1096	Alternative packages	1382	Exports costing
1097	Cash registers	1412	Supplier remittances
1098	Financial status report (Bulgaria)	1413	Customer remittances
1099	KEPYO	1414	Cash accounts remittances
1100	Accountants info documents	1415	Debtor remittances
1120	Time sheets	1416	Creditor remittances
11351	Retail	1453	Cash accounts special
1140	Construction accounting codes	1481	Cash account cash transactions
1151	Inventory documents	1553	Special from debtors
1154	Fixed asset documents	1581	Cash transactions from debtors
1171	Production orders	1653	Special from creditors
1181	Cheque documents	1681	Cash transactions from creditors
1212	Supplier reconciliation	1717	Suppliers reconciliation
1251	Purchase documents	2021	Actions
1253	Special transactions -Suppliers	2052	Service calls
1261	Expense documents	2095	Payroll results
1281	Cash transactions - Suppliers	5151	Inventory composition
1282	Imports costing	5171	Production order
1312	Reconcile customer with suppliers	7151	Production documents
1313	Customers reconciliation	8100	Cheques
1351	Sales Documents	9500	D.P.S.

C. ORIGIN – Transaction Source Values

The table below contains all the values that can be found in the internal string list ORIGIN. Field ORIGIN, that uses this string list is under FINDOC table.

ORIGIN List			
Code	Transaction source	Code	Transaction source
1	Normal	22	Production documents from subcontract
2	Costing Imports	23	From cash register
3	Costing Exports	24	Exchange rate gains and losses (evaluation)
4	Integrated financial & cost accounts	25	Exchange rate gains and losses (open-item)
5	Automatic pay-off	26	Previous fiscal years revenue
6	Reversal transactions	27	Inter-co. movement (from production)
7	Physical stock taking	28	Counter-balance account based costing
8	From receiving note (Service)	29	Card installment payment document
9	From folder (Service)	30	Forecast / Reversing document
10	Initialize fixed assets	31	Fixed assets revaluation
11	From call (Service)	32	Opportunity interest
12	Update physical stock taking documents	33	Contract document
14	Production documents from order-prod.orders	34	Production slips from orders
15	Composition doc. from sale/purchase/inventory	35	Bonus points from bonus cards
16	Revenue/Expenses	36	Export document (Ifaistos)
17	Credit notes due to turnover	37	Reversing of sales cost (Cyprus)
18	Production orders from sales order	38	Credit note due to turnover (purchases)
19	Semi-finished products production doc.	39	VAT Records (Art. 39b)
20	Production document from work slips	40	Contract Instalment Invoices
21	From import		

D. CSTTYPE (CSTINFO Table)

The following table contains all the values that can be found in the internal string list CSTTYPE, that appears in CSTINFO table records. Records marked with red can be imported / exported through Custom Administration tool.

CSTTYPE List	
Code	Control
0	Browsers / Reports
1	Screen Forms
2	Templates (Prototypes)
3	User-Defined Global (General) Fields
5	Imports (ASCII / Excel)
6	Alerts (EDA)
7	Personal User Menu
8	S1 Scripts
9	Custom SDK Code Files
10	Classic Menu
12	Data Flow Rules
13	Data Flow Scenarios
15	Gadgets
16	Advanced JavaScript
18	SQL Scripts
19	Business Processes (BAM)
20	Retail Designer
21	S1 Smart Scripts
92	SelectTop (Max Entries)
96	Trace (Objects Log File)
97	Custom Designers
98	Database Version
99	User Rights
98	Database Version
99	User Rights

E. X.SYS – System Parameters

The values in the table below apply to SoftOne tools such as browsers, reports, Java & VB script in custom forms, SBSL scripts and EDA (alerts). Use them by entering **:X.SYS.CName**, where CName is the name of each command (e.g. for login company code use the command :X.SYS.COMPANY).

System Parameters (:X.SYS.CName)		
Command (CName)	Description	Type
ACNSHEMA	Model of the chart of accounts	Small Int
BRANCH	Login branch	Small Int
CLIENTDATE	Login Client Date and Time	Datetime
CMPCURR	Company currency code	VarChar(5)
COMPANY	Login company	Small Int
COUNTRY	Country	Small Int
DBTYPE	1=ORACLE, 3=MSSQL	Small Int
FISCPRD	Login fiscal period	Small Int
FISCPRDPAY	Payroll year	Small Int
FOLDER	Portfolio	Small Int
FPDATE	First date of the login period	Date
FPERIOD	First fiscal period of the login fiscal year	Small Int
FYDATE	First date of the login fiscal year	Date
GROUPS	Login user group code	Small Int
ISADMIN	Returns 1 if login user is administrator	Small Int
LANGEXT	Login language	VarChar(5)
LOCKFISCPRD	Locked fiscal period	Small Int
LOGINDATE	Login date	Date
LPCDATE	Last date of the login period	Date
LPERIOD	Last fiscal period of the login fiscal year	Small Int
LYDATE	Last fiscal date of the login fiscal year	Date
MAINCOMPANY	Parent Company in a Group of Companies schema	Small Int
NFISCPRD	Next fiscal year	Small Int
PAYCPDATE	Date for the calculation of payroll period	Date
PAYFPDATE	First date of the payroll period	Date
PAYLPDATE	Last date of the payroll period	Date
PAYPERIOD	Payroll period	Small Int
PAYPRDTYPE	Payroll period type	Small Int
PERIOD	Login fiscal period	Small Int

PFISCPRD	Previous fiscal period	Small Int
PFYDATE	First date of the previous year	Date
PLYDATE	Last date of the previous year	Date
RELATEDCMPS	Relative Companies in Group of Companies schema	VarChar(1024)
SERIALNUM	Soft1 Serial number	VarChar(15)
SOCASH	Cash	Small Int
SOCASHUSER	Cash from user	Small Int
SOCURRENCY	Currency	Small Int
SYSDATE	System date	Date
USER	Login user	Small Int
USERBRANCHES	Login user branches	VarChar(1024)
USERNAME	Login user name	VarChar(30)
WHOUSES	Login warehouse	VarChar(4000)
WRKSTN	Workstation	Small Int

F. ACMD Commands – System Tools

The table below displays all the available commands that can be executed through the SoftOne menu using the job type "System Tools". The command syntax is: "**ACMD:xxx**" where xxx is the name of the Job.

acCommands Table	
Command	Job Name
acAbout	SoftOne About window
acAgentConfigure	SoftOne Agent Configuration
acAgentUninstall	SoftOne Agent Uninstall
acBackup	Backup database
acBuildHtmlDoc	Create accounting report model
acBuildOfflineFile	Create file for local synchronization
acCalcErrors	Calculation errors of local fields
acChangePWD	Change password
acChat	Support call
acCheckCS	Check client / server connection
acCheckDB	Update database window
acCheckForUpdates	Check for updates
acComputerSoft1Address	View computer soft1 address
acConnections	Connections administration
acCreateDB	Create database tables
acCutCompanyData	Export company data
acDesktop	SoftOne Desktop home page
acEditAccess	Define user access rights
acEditUserMenu	Design user menu
acExpBarClose	Hide menu panel
acExport	Export database tables
acFavorites	Favorite jobs
acFullTextSearch	Enable / Disable database full text search
acHelpContents	Help Contents and index
acHistory	Jobs history window
acImpFromClipboard	Import XXF from Clipboard
acImpFromFile	Import XXF from file
acImport	Import database tables
acKnowBase	Program knowledge base
acLicUse	Activate product license
acMainMenu2	Display main menu

acMemory	SoftOne Messages window
acmMind	Game Master mind
acModifMenu	Display group / user menu
acMP3Player	MP3 Player
acMySoft1	My installation window
acNewLicense	Newlicense
acNewSynchOffline	Sync offline (client) database (ver. 3.12.508 and newer)
acProgramNews	Program news
acRelogin	Relogin using differentcredentials
acReminder	Reminder window
acRemoteCommands	Scheduled commands
acRemoteConfig	Design remote commands
acRemoteServer	Remote Server
acReportGen	Custom reportsdesigner
acRestore	Restore database
acS1AddOns	SoftOne Add Ons
acSaaSServerLogin	SaaS Server Login
acScheduler	Scheduler
acSDKFiles	Custom SDK Files
acSelectTop	Select Top Entries per Object
acSendCustomerMessage	Send e-mail message to customer
acSettings	Settings
acSetupOffline	Off-Line installation parameters
acSetupPrinters	Printer settings
acSQLMonitor	SQL monitor
acSrvAccess	Define Server access rights
acStandard	Display old menu
acStartupSQL	Startup SQL queries
acStatusBar	Show / Hide status bar
acSyncDB	Update database window
acSynchOffLine	Sync offline (client) database
acSynchOnline	Sync online (server) database
acTetris	Game Tetris
acToggleMnu	Remove menu panel
acUpgradeLicense	Renew licence
acUserMenu	User menu

acWebPage	SoftOne site
acWMMModules	Web & Mobile Licences
acZReportSignB	Z Report of the type B electronic signature device

G. Editor Commands

The table below summarizes all the available editor commands and their use.

EDITOR COMMANDS		
Command	Operation / Usage	Data Type
\$DT	Displays Date & Time in two different fields	Datetime
\$TIME	Displays Time in HH:MM format (Saves the current Date)	Datetime
\$TIMESEC	Displays Time in HH:MM:SS format (Saves the current Date)	Datetime
\$PASSWORD	Password Mask (Displays text with asterisks)	Varchar
\$PASSWORDH	Hash Password (Displays text with asterisks)	Varchar
\$Y	Boolean Field (Displays checkbox control)	Small Integer
\$YN	Boolean Field in datagrids (Displays "Yes/No" combobox control)	Small Integer
\$WEBPAGE	Web Page Fields - Link to Internet Browser	Varchar
\$EMAIL	Email Fields - Link to Email Program	Varchar
\$PRINTERS	Displays Printers Window Dialog	Varchar
\$IMAGE	Displays image control (usage in Image fields)	Image
\$DATERANGE	Date from to	Datetime
\$TIMERANGE	Time from to	Datetime
\$COLOR	Displays Softone Color Selection window	Varchar
\$ONECOLOR	Displays Windows Color Selection window = 256 ^ 2Red * 256 ^ 1Green * 256 ^ 0Blue	Varchar
\$FILENAME	Displays File Selection Window and returns its path	Varchar
\$FOLDERNAME	Displays folder selection window and returns its path	Varchar
\$MASKEDIT	Code Mask Creation Window	Varchar
\$RECURRENCE	Displays Softone Scheduler Window	Varchar
\$MEMOEDIT	Corrects word wrap in datagrid lines for memo textbox fields	Varchar
\$LIST:OBJECT	Lookup from preselection lists created from browsers. Tables used are SOLIST and SOLISTLNS.	Varchar
\$XRUN	Runs external program \$XRUN(D[0,C:\Xplorer.exe,"/" + CUSEXTRA.VARCHAR01]) The above example runs SoftOne application using the switch parameters added in field CUSEXTRA.VARCHAR01.	Varchar

H. Editor Attributes

Inside Editors, that retrieve data from other Tables, Views or Strings (e.g. ITEM), it is possible to enter extra commands that have various functions, such as entries filtering, fields updating, change links etc.

The following table summarizes all the available commands as well as examples of their syntax.

Note: Line breaks and Quotes are not permitted in editors, instead you can use the sql function char().

EDITOR ATTRIBUTES		
Command	Operation / Usage	Syntax Example
W (Master Table Selectors)	Filters table data using extra where clause in SQL statement. Usage in selector fields linked to master tables.	SALDOC(W[A.COMPANY=:X.SYS.COMPANY]) Lookup data from sales documents table using as filter the Company Login. Login system parameters (as Login Company) can be used by entering a colon : and then the name of the param(:X.SYS.COMPANY). CUSTOMER(W[A.CODE LIKE CHAR(97)+CHAR(37)]) Customers that code starts with letter 'a' (... AND A.CODE LIKE 'a%')
W (Memory Table Selectors)	Filters memory table data using the operators AND, OR, <>, >, <. Usage in selector fields linked to memory tables.	SERIES(F[COMPANY;SOSOURCE=:X.SYS.COMPANY;1351],W[FPRMS=7000 OR FPRMS=7001]) Display sales series from login company that FPRMS=7000 or FPRMS=7001
W (String Lists)	Filters String List data. Usage in fields linked to string lists.	\$CCCMYSTR (W[1,2,3]) Lookup data from the string list CCCMYSTR displaying only the values that the key is 1 or 2 or 3.
F	Filters memory table data. Usage in fields linked to memory tables.	MTRMARK(F[CCCMYFIELD;SODTYPE=1;51]) Lookup from the table MTRMARK filtering data using CCCMYFIELD = 1 and SODTYPE = 51 fields and values.
M, R	Defines the field that will be returned. Usually used in string fields.	CUSTOMER(W[A.CCCMYFIELD=1],M,R[CODE]) Lookup data from customers (inherited table) using specific filter from CCC field and return the customer's code. (Use the Editor of the above example in a varchar field)
U	Updates fields after selection. Usage in selector fields linked to tables.	SERIES(U[FPRMS;BUSUNITS;=FPRMS;BUSUNITS@]) Lookup data from series table and update the FPRMS and BUSUNITS fields with the values of the fields FPRMS and BUSUNITS of the selected series. The @ symbol at the end indicates that the fields on the left, will always be updated, no matter if they already have a value or not. If you do not enter the @ symbol, then the fields will be updated only if they do not already have a value.
H	Redirection to Object, using the hyperlink of the field label.	CCCTABLE(H[CCCOBJECT]) or CCCTABLE(H[#ObjectID;RecordID]) Lookup data from table CCCTABLE. Redirection to Object CCCOBJECT.
E,W[]	Exclusion Filter in String Lists. Usage in fields linked to string lists.	\$CCCMYSTR (E,W[5]) Lookup data from the string list CCCMYSTR displaying only the values that the key is not 5.
J	Join - Link to specific field. Usage in Custom Design Printouts.	SALDOC(J[A.FINDOC])
I	Defines the field that will be updated when user clicks the button "Input" in fields of datagrids with editors.	ITEM(I[QTY1]) In sales documents, when user selects from the available items and clicks on the button "Input" then the data entered will update the QTY1 field.
L	Local Editor that lookups data from a table, db view or virtual table that is inside an object.	1. CCCMYVTTABLE(L,F[CCCMYVTTABLEFIELD=:OTHERTABLE.FIELD]) 2. #WHOUSE(L) (example in object Company → BRANCH.WHOUSE)

C	Removes the SQL expression " AND A.COMPANY=<LoginCompany> " from the where clause. Usage in selector fields linked to master tables.	SALDOC(W[A.CCCField in (1,2,3)],C) Lookup data from sales documents of all the companies, using also an extra where clause: "AND A.CCCField in (1,2,3)".
A	Removes the SQL expression " AND A.ISACTIVE=1 " from the where clause. Usage in selector fields linked to master tables.	CUSTOMER(A) Lookup data from Customer inherited table (Active and Inactive records)
Z	Applies to Soft1 Designer String Lists only. Displays both Key and Value of the string list.	\$SODTYPE(Z) Displays data in format "Key – Value"

I. Built-in Editors

The table below contains the analysis of the application internal editors.

Built-in Editors	
Command	Editor
@ACN	ACNT(F[ACNSCHEMA;SODTYPE=:X.SYS.ACNSCHEMA;;CSEL.SODTYPE])
@ACNAL	ACNTAL(F[ACNSCHEMA;SODTYPE=:X.SYS.ACNSCHEMA;90])
@ACNALM	ACNTAL(M,R[CODE],F[ACNSCHEMA;SODTYPE=:X.SYS.ACNSCHEMA;90],H[ACNTAL])
@ACNCTGR	ACNCATEGORY(F[ACNSCHEMA;SODTYPE=:X.SYS.ACNSCHEMA;;CSEL.SODTYPE])
@ACNGL	ACNTGL(F[ACNSCHEMA;SODTYPE=:X.SYS.ACNSCHEMA;89])
@ACNGLM	ACNTGL(M,R[CODE],F[ACNSCHEMA;SODTYPE=:X.SYS.ACNSCHEMA;89],H[ACNTGL])
@ACNGRP	ACNGROUP(F[ACNSCHEMA;SODTYPE=:X.SYS.ACNSCHEMA;;CSEL.SODTYPE])
@ACNM	ACNT(M,R[CODE],F[ACNSCHEMA=:X.SYS.ACNSCHEMA],H[ACNTGL])
@ACNMDL	ACNMODEL(F[ACNSCHEMA=:X.SYS.ACNSCHEMA])
@ACNMDLAL	ACNMODELAL(F[ACNSCHEMA;SODTYPE=:X.SYS.ACNSCHEMA;90])
@ACNSLG	ACNISOLOG(F[ACNSCHEMA;SODTYPE=:X.SYS.ACNSCHEMA;;CSEL.SODTYPE])
@ACNT	ACNT(F[ACNSCHEMA=:X.SYS.ACNSCHEMA])
@ALLCLSFLDR	FINDOC(M,R[FINCODE],W[SOSOURCE IN (1282, 1382) AND EXISTS (SELECT 1 FROM CSTFLDR CS WHERE CS.CSTFLDR=A.FINDOC AND CS.CFOCLOSEOK<>1)])
@ARS	AREAS(F[SODTYPE=:CSEL.SODTYPE])
@CKAD	KAD(W[EXISTS (SELECT 1 FROM COMPANYKAD WHERE COMPANY=:X.SYS.COMPANY AND A.KAD=COMPANYKAD.KAD UNION SELECT 1 FROM COMPANY WHERE COMPANY=:X.SYS.COMPANY AND A.KAD=COMPANY.KAD)])
@CLSFLDR	FINDOC(M,R[FINCODE],W[SOSOURCE=&SOSOURCE AND EXISTS (SELECT 1 FROM CSTFLDR CS WHERE CS.CSTFLDR=A.FINDOC AND CS.CFOCLOSEOK<>1)])
@COL	PRSNIN(W[EXISTS (SELECT 1 FROM PRSEXT WHERE A.PRSN=PRSEXT.PRSN AND PRSEXT.COMPANY=:X.SYS.COMPANY AND PRSEXT.SODTYPE=31 AND PRSEXT.ISACTIVE=1)])
@COLM	PRSNIN(M,R[CODE],W[EXISTS (SELECT 1 FROM PRSEXT WHERE A.PRSN=PRSEXT.PRSN AND PRSEXT.SODTYPE=31 AND PRSEXT.COMPANY=:X.SYS.COMPANY AND PRSEXT.ISACTIVE=1)])
@COMM	COMMENTS(M,R[NAME],F[SOSOURCE=:CSEL.SOSOURCE])
@CRCNTRL	CRCONTROL(F[SODTYPE=:CSEL.SODTYPE])
@DSCPLC	VCRDRULE(F[SODTYPE;SOTYPE=:CSEL.SODTYPE;2])
@IRSM	IRSDATA(M,R[NAME])
@JOB	JOBTYPE(F[SODTYPE=:CSEL.SODTYPE])
@MATEXP	EXPN(F[USEBYITEM=1])
@MATOMD	MTRGROUP(F[SODTYPE=:CSEL.SODTYPE])
@PAYT	PAYMENT(F[SODTYPE=:CSEL.SODTYPE])
@PERIODACN	PERIOD(W[COMPANY = :X.SYS.COMPANY AND FISCPRD = :X.SYS.FISCPRD])

@PERIODPAY	PAYPRD(W[COMPANY = :X.SYS.COMPANY AND FISCPRDPAY = :X.SYS.FISCPRDPAY])
@PERIODMTR	PERIOD(W[COMPANY = :X.SYS.COMPANY AND FISCPRD = :X.SYS.FISCPRD AND PERIOD<1000])
@PERIODTRD	PERIOD(W[COMPANY = :X.SYS.COMPANY AND FISCPRD = :X.SYS.FISCPRD AND PERIOD>0 AND PERIOD<1000])
@PUR	PRSNIN(W[EXISTS (SELECT 1 FROM PRSEXT WHERE A.PRSN=PRSEXT.PRSN AND PRSEXT.SODTYPE=32 AND PRSEXT.COMPANY=:X.SYS.COMPANY AND PRSEXT.ISACTIVE=1)])
@PURM	PRSNIN(M,R[CODE],W[EXISTS (SELECT 1 FROM PRSEXT WHERE A.PRSN=PRSEXT.PRSN AND PRSEXT.SODTYPE=32 AND PRSEXT.COMPANY=:X.SYS.COMPANY AND PRSEXT.ISACTIVE=1)])
@RPCPLC	PRCPOLICY(F[SODTYPE;SOTYPE=:CSEL.SODTYPE;1])
@SAL	PRSNIN(W[EXISTS (SELECT 1 FROM PRSEXT WHERE A.PRSN=PRSEXT.PRSN AND PRSEXT.SODTYPE=30 AND PRSEXT.ISACTIVE=1)])
@SALM	PRSNIN(M,R[CODE],W[EXISTS (SELECT 1 FROM PRSEXT WHERE A.PRSN=PRSEXT.PRSN AND PRSEXT.SODTYPE=30 AND PRSEXT.ISACTIVE=1)])
@SXACNTM	SXACNT(M,R[CODE],H[SXACNT])
@SXACNTSM	SXACNT(W[A.MTRTYPE IN (1,3)],M,R[CODE],H[SXACNT])
@SXACNTXM	SXACNT(W[A.MTRTYPE IN (2,4)],M,R[CODE],H[SXACNT])
@TEC	PRSNIN(W[EXISTS (SELECT 1 FROM PRSEXT WHERE A.PRSN=PRSEXT.PRSN AND PRSEXT.SODTYPE=33 AND PRSEXT.COMPANY=:X.SYS.COMPANY AND PRSEXT.ISACTIVE=1)])
@TEMPLATESABC	TEMPLATESABC(F[SOSOURCE;GLTEMPLATESMD=:SOSOURCE;2])
@TEMPLATESGL	TEMPLATESGL(F[SOSOURCE;GLTEMPLATESMD=:SOSOURCE;1])
@TEMPLATESSX	TEMPLATESSX(F[SOSOURCE;GLTEMPLATESMD=:SOSOURCE;3])
@TFPRMS	TFPRMS(F[SOSOURCE=:CSEL.SOSOURCE])
@TRDBRN	TRDBRANCH(F[TRDR=:CSEL.TRDR])
@TRDBSN	TRDBUSINESS(F[SODTYPE=:CSEL.SODTYPE])
@TRDCTGR	TRDCATEGORY(F[SODTYPE=:CSEL.SODTYPE])
@TPGROUP	TRDPGROUP(F[SODTYPE=:CSEL.SODTYPE])
@TRDRS	TRDR(W[SODTYPE IN (12,13,15,16)])
@UAT1	UATBL01(F[ACNSHEMA;SODTYPE=:CSEL.ACNSHEMA;;CSEL.SODTYPE])
@UAT2	UATBL02(F[ACNSHEMA;SODTYPE=:CSEL.ACNSHEMA;;CSEL.SODTYPE])
@UAT3	UATBL03(F[ACNSHEMA;SODTYPE=:CSEL.ACNSHEMA;;CSEL.SODTYPE])
@UAT4	UATBL04(F[ACNSHEMA;SODTYPE=:CSEL.ACNSHEMA;;CSEL.SODTYPE])
@UAT5	UATBL05(F[ACNSHEMA;SODTYPE=:CSEL.ACNSHEMA;;CSEL.SODTYPE])
@UCT1	UCTBL01(F[SODTYPE=:CSEL.SODTYPE])
@UCT2	UCTBL02(F[SODTYPE=:CSEL.SODTYPE])
@UCT3	UCTBL03(F[SODTYPE=:CSEL.SODTYPE])
@UCT4	UCTBL04(F[SODTYPE=:CSEL.SODTYPE])
@UCT5	UCTBL05(F[SODTYPE=:CSEL.SODTYPE])

@UT1	UTBL01(F[SODTYPE=:CSEL.SODTYPE])
@UT2	UTBL02(F[SODTYPE=:CSEL.SODTYPE])
@UT3	UTBL03(F[SODTYPE=:CSEL.SODTYPE])
@UT4	UTBL04(F[SODTYPE=:CSEL.SODTYPE])
@UT5	UTBL05(F[SODTYPE=:CSEL.SODTYPE])
@ZIPM	VZIPADDRESS(M,R[CODE])

J. Related Job Commands

The table below contains all the commands that are used to access related jobs per module.

Transaction Related Jobs – Table FINDOC		
Related Job	Reference Modules	SoftOne command
Payment terms	Sales – Purchases – Collections – Remittances	XCMD:SFPAYTERMS
Open-item	Sales – Purchases – Collections – Remittances	XCMD:1031
Pay-off	Sales	XCMD:CFNCUSDOC,EKSOFL:1,S
Pay-off	Purchases	XCMD:CFNSUPDOC,EKSOFL:1,S
Costing folder	Sales – Purchases – Collections – Remittances	XCMD:CSTFINLNS,S;FINDOC.FINDOC
Recalculation	Sales – Purchases – Retail – Collections – Remittances	XCMD:10006 XCMD:100061 (no confirmation messages)
Originated from	Sales – Purchases – Collections – Remittances - Item Docs – Production Docs – Service Folders	XCMD:SFFINFROM
Converted into	Sales – Purchases – Collections – Remittances – Service Folders	XCMD:SFFINTO
Lots proposal	Sales – Purchases – Item Docs – Collections – Remittances	XCMD:10112
Agreement on total	Sales – Purchases – Retail – Collections – Remittances – Service Folders	XCMD:SFFINPRCRULE
Select contract	Sales – Purchases – Collections – Remittances	XCMD:SFCNTR
Select agreement	Sales – Purchases – Retail – Collections – Remittances	XCMD:SFPRCPOLICY
Match with revenue/expenses	Sales – Purchases – Retail Collections – Remittances – Item Docs	XCMD:10113
Activity Based Costing Analysis	Sales – Purchases – Retail – Collections – Remittances – Item Docs	XCMD:ABCTRNLINES,S[ABCTABLE=FINDOC];FINDOC.FINDOC
Entry	Sales – Purchases – Retail – Collections – Remittances – Item Docs – Production Docs	XCMD:10110
Related files	Sales – Purchases – Retail – Collections – Remittances – Item Docs – Production Docs – Service Folders / Calls	XCMD:XTRDOCDATA,S
Update sales prices	Purchases	XCMD:10107
Advance payment / Order	Retail	XCMD:SFADVANCE
Credit card installments	Retail	XCMD:908
Production cost (subcontracted)	Special Transactions	XCMD:CSTPRDLNS,S;FINDOC.FINDOC

Pay-off	Special Transactions	XCMD:&EKSOFL
Settlement	Revenues – Expenses	XCMD:&CMDOBJ
Related documents	Service Calls (APPMNT)	XCMD:SFFINTO
Conversion history	Sales – Purchases – Item Docs	XCMD:10114
Record printouts	Sales – Purchases	XCMD:10115

Customer / Supplier Related Jobs – Table TRDR		
Relative Job	Reference Modules	SoftOne command
Balance per company	Customers – Suppliers – Debtors – Creditors	XCMD:SFCMPREM
Register of non-natural persons	Customers	XCMD:GsisCmpAfmData

Item Related Jobs– Table MTRL		
Relative Job	Reference Modules	SoftOne command
Balance per company	Items	XCMD:SFCMPREM

Accounting Related Jobs – Table ACNT		
Relative Job	Reference Modules	SoftOne command
Entry model	Accounting Entries	XCMD:10009
Related files	Accounting Entries	XCMD:XTRDOCDATA,S

Contract Related Jobs– Table SRVC		
Relative Job	Reference Modules	SoftOne command
Related document	Contracts	XCMD:1001

Budget Related Jobs– Table BUDGET		
Relative Job	Reference Modules	SoftOne command
Update initial data	Budget – Budget data	XCMD:InitBudgetData,s,BUDGET.BUDGET
Update actual data	Budget – Budget data	XCMD:CalcBadgetData;BUDGET.BUDGET
Delete data	Budget – Budget data	XCMD:DELETEBUDGETDATA;BUDGET.BUDGET
Copy data	Budget	XCMD:COPYBUDGETDATA;BUDGET.BUDGET

File / Document Related Jobs – Table XDOCS		
Relative Job	Reference Modules	SoftOne command
Open file	Documents file	XCMD:10003
Save to file	Documents file	XCMD:10004
Save file to DB	Documents file	XCMD:10011
Delete file from DB	Documents file	XCMD:10012
Upload file to Azure	Documents file	XCMD:11011
Delete file from Azure	Documents file	XCMD:11012
Download from URL	Documents file	XCMD:11013

CRM Related Jobs – Table SOACTION		
Relative Job	Reference Modules	SoftOne command
Next step	Actions – Calls – Meetings – Tasks – e-mails	XCMD:FOLLOWUPDLG
Action history	Actions – Calls – Meetings – Tasks – e-mails	XCMD:ACTHISTDLG
Repeat action	Actions – Calls – Meetings – Tasks – e-mails	XCMD:ACTREPEATDLG
Sales opportunity	Actions – Calls – Meetings – Tasks – e-mails	XCMD:1002
Previous step	Actions – Calls – Meetings – Tasks – e-mails	XCMD:SFPREVSTEP
Related files	Actions – Calls – Meetings – Tasks – e-mails – Draft entries	XCMD:XTRDOCDATA,S
Attachments	e-mails	XCMD:1001
Send	e-mails	XCMD:1000
Reply	e-mails	XCMD:1003
Reply to all	e-mails	XCMD:1004
Forward	e-mails	XCMD:1005
Templates	e-mails	XCMD:1006
Convert draft entry	Draft entries	XCMD:1001
Register of non-natural persons	Draft entries	XCMD:GsisCmpAfmData
Reports recording	Actions – Calls – Meetings – Tasks – e-mails – Draft entries	XCMD:10115
Follow up	Actions – Calls – Meetings – Tasks – e-mails – Draft entries	XCMD:1200
Data	Actions – Calls – Meetings – Tasks – e-mails – Draft entries	XCMD:SFLEAD
Attachment	e-mails	XCMD:10006
Send and Save	e-mails	XCMD:1007

K. Browser Job Commands

The table below contains the commands that can be found in the right click context menu of object browsers.

Document Browser Jobs – Table FINDOC		
Relative Job	Reference Modules	SoftOne command
Conversion	Sales	XCMD:CONVERTDLG,SOSOURCE:1351
Conversion	Item Docs	XCMD:CONVERTDLG,SOSOURCE:1151
Conversion	Purchases	XCMD:CONVERTDLG,SOSOURCE:1251
Convert document	Fixed assets docs	XCMD:ASSDOC,N,CNV:1
Convert lines	Sales – Purchases	XCMD: CONVERTDLG,SOSOURCE:1351,LN:1
Remove pending	Sales – Purchases – Retail – Item Docs.	XCMD:REMOVERESTMODE
Restore pending	Sales – Purchases – Retail – Item Docs.	XCMD:UNDOREMOVEREST
Remove pending (lines)	Sales – Purchases – Retail – Item Docs.	XCMD:REMOVERESTMODE,LN:1
Restore line pending	Sales – Purchases – Retail – Item Docs.	XCMD:UNDOREMOVEREST,LN:1
Documents processing	Sales	XCMD:PROCESSDLG,SOSOURCE:1351
Documents processing	Item Docs	XCMD:PROCESSDLG,SOSOURCE:1151
Documents processing	Purchases	XCMD:PROCESSDLG,SOSOURCE:1251
Copy document	Sales	XCMD:SALDOC,N,CNV:3
Copy document	Retail	XCMD:RETAILDOC,N,CNV:3
Copy document	Purchases	XCMD:PURDOC,N,CNV:3
Copy document	Item Docs	XCMD:ITEDOC,N,CNV:3
Copy document	Fixed assets docs	XCMD:ASSDOC,N,CNV:3
Cancel transactions	Sales – Purchases – Retail - Item docs – Fixed assets – Remittances – Collections – Special transactions	XCMD:1001
Cancel by reversal	Sales	XCMD:SALDOC,N,CNV:100
Cancel by reversal	Purchases	XCMD:PURDOC,N,CNV:100
Cancel by reversal	Item Docs	XCMD:ITEDOC,N,CNV:100
Cancel by reversal	Collections – Special Transactions	XCMD:1003

Re-issue by cancelling with reversal	Sales – Retail	XCMD:10016
Discount amount (forecast)	Invoice discounting documents	XCMD:Docspmcash
Discount amount (disbursement)	Invoice discounting documents	XCMD:Docsmcash
Balance disbursement	Invoice discounting documents	XCMD:Docsrcash
Discount expenditure	Invoice discounting documents	XCMD:DocsPayExpences
Management expenditure	Invoice discounting documents	XCMD:DocsExpences
Create credit notes	Credit notes results	XCMD:CreditDoc
Order planning	Sales	XCMD:IteOrdersServ
Transfer to purchase documents	Sales	XCMD:CRTPURORDS
Transfer to production documents	Sales	XCMD:CRTPRDDOCS
Transfer to production orders	Sales	XCMD:CRTPRDORDDOCS
Recalculate	Sales	XCMD:PRCRECALCDLG
Discount credit notes calculation	Sales	XCMD:CULCTMPCREDIT

Customers / Projects Browser Jobs – Tables TRDR / PRJC		
Relative Job	Reference Modules	SoftOne command
Save actions	Customers – Suppliers – Debtors – Creditors – Projects	XCMD:CRSACTIONDLG
Send email	Customers – Suppliers – Debtors – Creditors	XCMD:SENDEMAILDLG
Insert file	Customers – Suppliers – Debtors – Creditors – Projects	XCMD:CreateXDoc

Contacts Browser Jobs – Table PRSN		
Relative Job	Reference Modules	SoftOne command
Save actions	Contacts	XCMD:CRSACTIONDLG
Send email	Contacts	XCMD:SENDEMAILDLG
Export to Outlook	Contacts	XCMD:EXPOUTCONTACTS
Export to Google	Contacts	XCMD:EXPGCONTACTS
Insert file	Contacts	XCMD:CreateXDoc

Calendar	Contacts	XCMD:102
-----------------	----------	----------

Item Browser Jobs – Table MTRL		
Relative Job	Reference Modules	SoftOne command
Mass price modification	Items	XCMD:IteCalcMarkUp
Transfer among W/h	Items	XCMD:IteWhouseRest
Order planning	Items	XCMD:IteOrdersServ
Insert file	Items – Fixed assets - Services	XCMD:CreateXDoc
Transfer to “Web items” file	Items	XCMD:IteToWeb
Transfer to “Touch items” file	Items	XCMD:IteToRetail

Accounting Browser Jobs – Table ACNT		
Relative Job	Reference Modules	SoftOne command
Cancel entry	General ledger entries	XCMD:1001
Cancel by reversal	General ledger entries	XCMD:ACCGLHEADER,N,CNV:100
Cancel entry	Cost accounting entries	XCMD:1001
Cancel by reversal	Cost accounting entries	XCMD:ACCALHEADER,N,CNV:100

Cheque Browser Jobs – Table CHEQUE		
Relative Job	Reference Modules	SoftOne command
Cash transaction	Cheques	XCMD:ChqCfnDoc
Insert file	Cheques	XCMD:CreateXDoc

Services Browser Jobs – Table SRVC		
Relative Job	Reference Modules	SoftOne command
Related documents	Service calls	XCMD:100001
Cancel call	Service calls	XCMD:100002
Cancel transactions	Service folders	XCMD:1001
Revoke pending status	Service folders	XCMD:1002
Spare parts documents	Service folders	XCMD:SALDOC,N,FROMSRVCARD:1
Service documents	Service folders	XCMD:SALDOC,N,FROMSRVCARD:2

Service – spare parts document	Service folders	XCMD:SALDOC,N,FROMSRVCARD:0
Shipment to customer	Service folders	XCMD:SALDOC,N,SENDSRVCARD:1

Installations Browser Jobs – Table INST		
Relative Job	Reference Modules	SoftOne command
Insert file	Installations	XCMD:CreateXDoc

Warranties Browser Jobs – Table Guaranty		
Relative Job	Reference Modules	SoftOne command
Save file	Warranties	XCMD:CreateXDoc

File Documents Browser Jobs – Table XDOC		
Relative Job	Reference Modules	SoftOne command
Open document	Documents file	XCMD:10003
Export file	Documents file	XCMD:10004
Send by e-mail	Documents file	XCMD:SendDocWithMail

CRM Browser Jobs – Table SOACTION		
Relative Job	Reference Modules	SoftOne command
Next step	General actions–Calls – Meetings – Tasks – e-mails – CRM action statistics	XCMD:FOLLOWUPDLG,BROWSER:1
Action history	General actions – Calls – Meetings – Tasks – e-mails – CRM action statistics	XCMD:ACTHISTDLG,BROWSER:1
Repeat action	General actions – Calls – Meetings – Tasks – e-mails – CRM action statistics	XCMD:ACTREPEATDLG,BROWSER:1
Save action	CRM action statistics	XCMD:1100
Insert action	Draft entries	XCMD:CRTACTIONDLG
Send email	Draft entries	XCMD:SENDEMAILDLG
Export to Google	Draft entries	XCMD:GLCONTACTS

L. Command Line Switches

The table below contains the switches that can be used when executing SoftOne (Xplorer.exe file).

Command Line Switches		
Switch	Operation / Usage	Example
/server	Runs SoftOne Application Server. Defaults ports for SoftOne are: 1) Application Port=22001, 2) WebPort=22002	X:\...\Xplorer.exe /server
/server:XXX	Runs SoftOne Application Server using XXX ip address or HostName.	X:\...\Xplorer.exe /server:XXX
/host:XXX	Runs SoftOne application as client and connects to application server XXX (HostName or ip address).	X:\...\Xplorer.exe /host:XXX
/port:XXX	Changes the TCP/IP port of Application Server to XXX	X:\...\Xplorer.exe /server /port:XXX
/install	Installs SoftOne Application Server as Windows service. Defaults ports for SoftOne are: 1) Application Port=22001, 2) WebPort=22002 Always run it as Administrator.	X:\...\Xplorer.exe /install
/uninstall	Uninstalls SoftOne Application Server from windows services.	X:\...\Xplorer.exe /uninstall
/regserver	Installs the registry keys for the use of com server. It also adds SoftOne file associations for the following file types: .auv, .cst, .imp, .imp2, .impc, .impc2, .xmd, .xpr, .xxf Always run it as Administrator.	X:\...\Xplorer.exe /regserver
/unregserver	Uninstalls the registry keys installed from regeserver.	X:\...\Xplorer.exe /unregserver
/xco:filename.XCO	Auto runs the xco (connection) file. Autologin in Softone works when there is a section [LOGIN] in the xco file.	X:\...\Xplorer.exe /xco:filename.XCO
/sxco	Executes the configuration commands using the connection file " default.xco " that exists in the application server folder. Useful when clients do not use different parameters (offline, autologin, etc) in their connection file (xco) than the ones that application server uses.	X:\...\Xplorer.exe /host:XXX /sxco
/balance:X	Runs SoftOne Balancer for X number of users. It runs SoftOne Application Server when X SoftOne users login. SoftOne Application Server must not be installed as windows service.	X:\...\Xplorer.exe /balance:10
/srvrestart:XXX	Restarts application server in XXX time intervals	X:\...\Xplorer.exe /srvrestart:10
/forceoffline:1	Forces offline connection mode	X:\...\Xplorer.exe /forceoffline:1
/np "command"	Executes the shell command line followed.	<u>Insert xxf file on login</u> X:\...\Xplorer.exe /np "X:\...\myfile.xxf"
/lang:language	Defines SoftOne application language	X:\...\Xplorer.exe /lang:eng

Switch	Operation / Usage	Example
/xcmd:MenuJob	Runs and opens the MenuJob (usually SoftOne Object) after login. It can be combined with any menu command, such as FORM, LIST, AUTOEXEC, CUSTOM, FORCEFILTERS, FORCEVALUES etc. Usually used with autologin xco.	1. X:\...\Xplorer.exe /xcmd:CUSTOMER 2. X:\...\Xplorer.exe /xcmd:SALDOC[AUTOEXEC=2] 3. X:\...\Xplorer.exe /xcmd:ITEM[LIST=MYLIST,FORM=MYFORM]
/hide	Minimizes SoftOne application in Windows tray. Usually combined with XCMD command, in order to run only one object/entity and not the entire application.	X:\...\Xplorer.exe /hide
/execute:filename	<p>Runs "filename", using commands similar to Remote Server or Scheduler commands. Mostly used in Windows scheduled Tasks.</p> <p>Note: For Azure Installations, inside the local xco file, always use the plain text password not the encrypted one.</p> <p>Examples</p> <p>1. Auto execute a SoftOne script</p> <p>a) Create a file (S1Job.txt) using the following commands: JOBNAME=MyJob TYPE=BATCH OBJECT=FORMIMPORT,SCRIPTNAME:TestScript XCOFilename=MYXCOFILE.XCO</p> <p>b) Create windows shortcut X:\...\Xplorer.exe /execute:X:\...\S1Job.txt</p> <p>2. Auto execute a SoftOne report and save it as ASCII file</p> <p>a) Create a file (S1Report.txt) using the following commands: TYPE=Report OBJECT=CUSTOMER JOBNAME=CustomersBalance OUTPUT=928 FILENAME=X:\...\Myreport.txt XCOFilename=MYXCOFILE.XCO</p> <p>b) Create windows shortcut X:\...\Xplorer.exe /execute:X:\...\S1Report.txt</p>	
/pad	Deprecated – SoftOne interface for Windows tablets	X:\...\Xplorer.exe /pad
/noversion	Does not update client files from Application / Cloud Server	X:\...\Xplorer.exe /noversion
/refresh	Forces client update from Application / Cloud Server files	X:\...\Xplorer.exe /refresh
/noparams	Does not load PARAMS.CFG file	X:\...\Xplorer.exe /noparams
/db:xdbxdrv.bpl	Used for connections with Oracle database	X:\...\Xplorer.exe /db:xdbxdrv.bpl
/webreport:0	Closes the WebServer TCP/IP port (22002)	X:\...\Xplorer.exe /webreport:0
/webreport:XXX	Changes the WebServer TCP/IP port to XXX	X:\...\Xplorer.exe /webreport:XXX
/webfeed:XXX	Runs SoftOne as web feeder using the defined ip address. Default port is 22099.	X:\...\Xplorer.exe /webfeed:test.oncloud.gr
/bam:LOCAL	Runs BAM locally, for testing BAM scenarios. Local webfeeder needs to be running (/webfeed:LOCAL)	X:\...\Xplorer.exe /BAM:LOCAL

Switch	Operation / Usage	Example
/xextui	Runs SoftOne using the New UI	X:\...\Xplorer.exe /xextui
/azconsole	Runs Azure Console	X:\...\Xplorer.exe /azconsole
/cloudusrtimeout:X	Specifies the hours that the client application will be terminated due to user inactivity. Default is 3 hours.	X:\...\Xplorer.exe /cloudusrtimeout:6
/usewebview2	Uses Microsoft Edge as default SoftOne Web browser	X:\...\Xplorer.exe /usewebview2

M. XCO: On-Premise Connection Settings

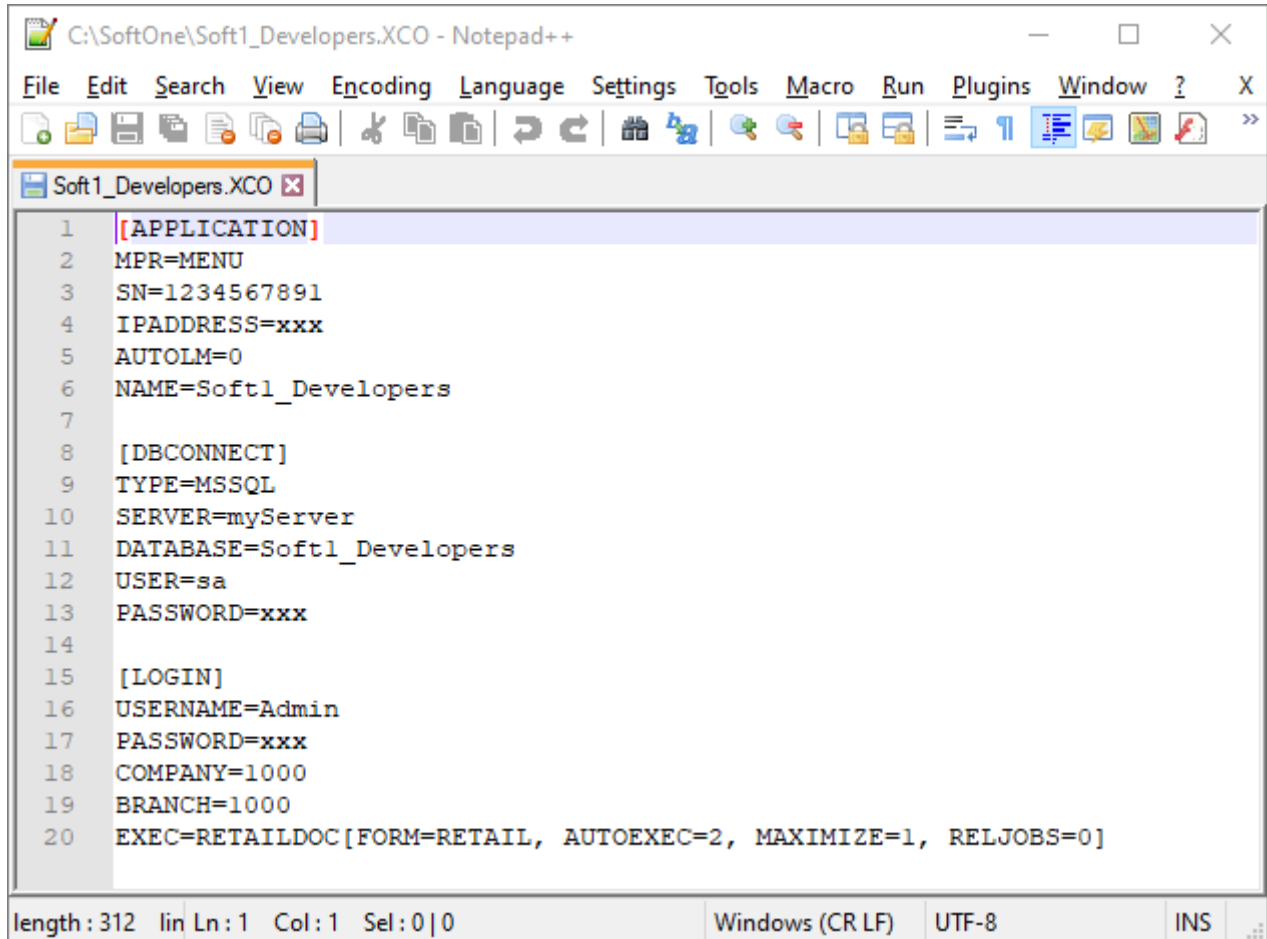
The tables below contain the different sections and arguments that can be used in SoftOne connection file XCO.

XCO Parameters		
Section Application		
Command	Operation / Usage	Example
MPR=MENU	Shows SoftOne Menu	MPR=MENU
SN=SoftOne serial number	Current connection's serial number	SN=0012345678
IPADDRESS=XXX	Licence Manager IP address (Combined with command AUTOLM=0)	IPADDRESS=172.23.3.1
AUTOLM=0 or 1	Autosearch the network for Licence Manager (0=No, 1=Yes)	AUTOLM=0
NAME= <Connection Name (XCO)>	Connection (XCO) name, that will be displayed in login "Connection" list. XCO File (MyConnFile.XCO) might have a different name.	NAME=MyConnectionName
NETLIB=SoftOne.Lib.dll	SoftOne Library for custom .NET inprocess or outprocess applications. Note: Libraries S1Interop.dll and S1System.dll are deprecated.	
NETDLL= <.NET dll Filename>	Name of the custom .NET inprocess file	NETDLL=MyNETdll.dll
PATH= <Application path>	Application path	PATH=C:\Softone
ADDON= <Delphi dll Filename>	Name of the custom Delphi inprocess file	ADDON=MYDLL.dll
FORCEOFFLINE= 1	Forces offline connection mode. Usually used from outprocess applications.	
FORCENETDLL= 1	Login is not allowed if the inprocess net dll can not be initialized.	
RDEFAULTS=0 or 1	Display last login data on Login Dialog (Default=1) (0=No, 1=Yes)	
LANGUAGE=xxx	Login Language (GRE, ENG)	LANGUAGE=GRE
LOCALE=xxxx	Locale Identifier (LCID) (usually used in WEB.XCO) Greek=1032, Romanian=1048, Bulgarian=1026	LOCALE=1032

XCO Parameters		
Section DBConnect		
Command	Operation / Usage	Example
TYPE=MSSQL or ORACLE	<p>Database type (MSSQL, ORACLE).</p> <p>ORACLE connections need the use of commands: LIBRARYNAME=dbexpoda.dll VENDORLIB=dbexpoda.dll DRIVERFUNC=getSQLDriverORA ORACLE connections also need SoftOne to run with the switch /DB:XDBXDRV.BPL e.g. C:\Soft1\Xplorer.exe /DB:XDBXDRV.BPL</p>	<p>1. MSSQL Connection [DBCONNECT] TYPE=MSSQL SERVER=SQLServerName DATABASE=DBName USER=sa PASSWORD=xxx</p> <p>2. ORACLE Connection [DBCONNECT] TYPE=ORACLE DRIVER=XDBXDRV.BPL SERVER=MyPCName:1021:oraclesvc USER=DBName PASSWORD=xxx LIBRARYNAME=dbexpoda.dll VENDORLIB=dbexpoda.dll DRIVERFUNC=getSQLDriverORA</p>
SERVER= <Database Server>	Name of the Database Server.	<p>1. MSSQL Connection SERVER=SQLServerName</p> <p>2. ORACLE Connection SERVER=MyPCName:1021:oraclesvc "MyPCName"=Name of the workstation "1021"=Port that Oracle Service is available "oraclesvc"=Name of Oracle Service</p>
DATABASE= <DB Name>	Database Name	DATABASE=DBName
USER= <Database UserName>	<p>For MSSQL connections it stands for Database User Name.</p> <p>For ORACLE connections it stands for Database Name.</p>	<p>1. MSSQL Connection USER=sa</p> <p>2. ORACLE Connection USER=DBName</p>
PASSWORD= <DB Password>	Database User Password	PASSWORD=xxx
SQLTIMEOUT= <sec>	Configures the remote query timeout. Specifies how long (in seconds) a remote operation can take before SQL Server times out. Default value in MSSQL is 600 (10 minutes).	SQLTIMEOUT=100
COMMANDTIMEOUT= <sec>	Sets the wait time (in seconds) before terminating the attempt to execute a command and generating an error.	COMMANDTIMEOUT=100
DRIVER=XDBXDRV.BPL	Used in ORACLE connections	
LIBRARYNAME=dbexpoda.dll	Used in ORACLE connections	
VENDORLIB=dbexpoda.dll	Used in ORACLE connections	
DRIVERFUNC=getSQLDriverORANET	Used in ORACLE connections	

XCO Parameters		
Section [Login]		
Command	Operation / Usage	Example
USERNAME= <SoftOne UserName>	Autologin SoftOne User Name	USERNAME=S1UserName
PASSWORD= <SoftOne UserPassword>	Autologin SoftOne User Password	PASSWORD=xxx
COMPANY= <SoftOne Company Code>	Autologin SoftOne Company Code	COMPANY=1000
BRANCH= <SoftOne Branch Code>	Autologin SoftOne Branch Code	BRANCH=1000
HIDEBAR=1	Hides the Toolbar	
HIDEMENU=1	Hides SoftOne User Menu	
HIDEXPLORER=1	Minimizes SoftOne application in Windows tray. Usually combined with command EXEC, in order to display only one object/entity and not the entire application.	
EXEC= <SoftOne Command>	Executes the SoftOne command followed (usually SoftOne Object). It can be combined with any menu command, such as FORM, LIST, AUTOEXEC, CUSTOM, FORCEFILTERS, FORCEVALUES etc.	EXEC=SALDOC[AUTOEXEC=2] EXEC=ACMD:acRemoteServer

XCO Parameters	
SQL SERVER – Windows Authentication	
For Windows Authentication Connections use the following commands inside the xco file.	
[DBCONNECT]	
TYPE=MSSQL	
SERVER= <servername>	
DATABASE= <dbname>	
CONNECTIONSTRING= Provider=SQLOLEDB.1;Integrated Security=SSPI;Data Source=%s;Initial Catalog=%s;%sAuto Translate=False;Application Name=SOFT1	



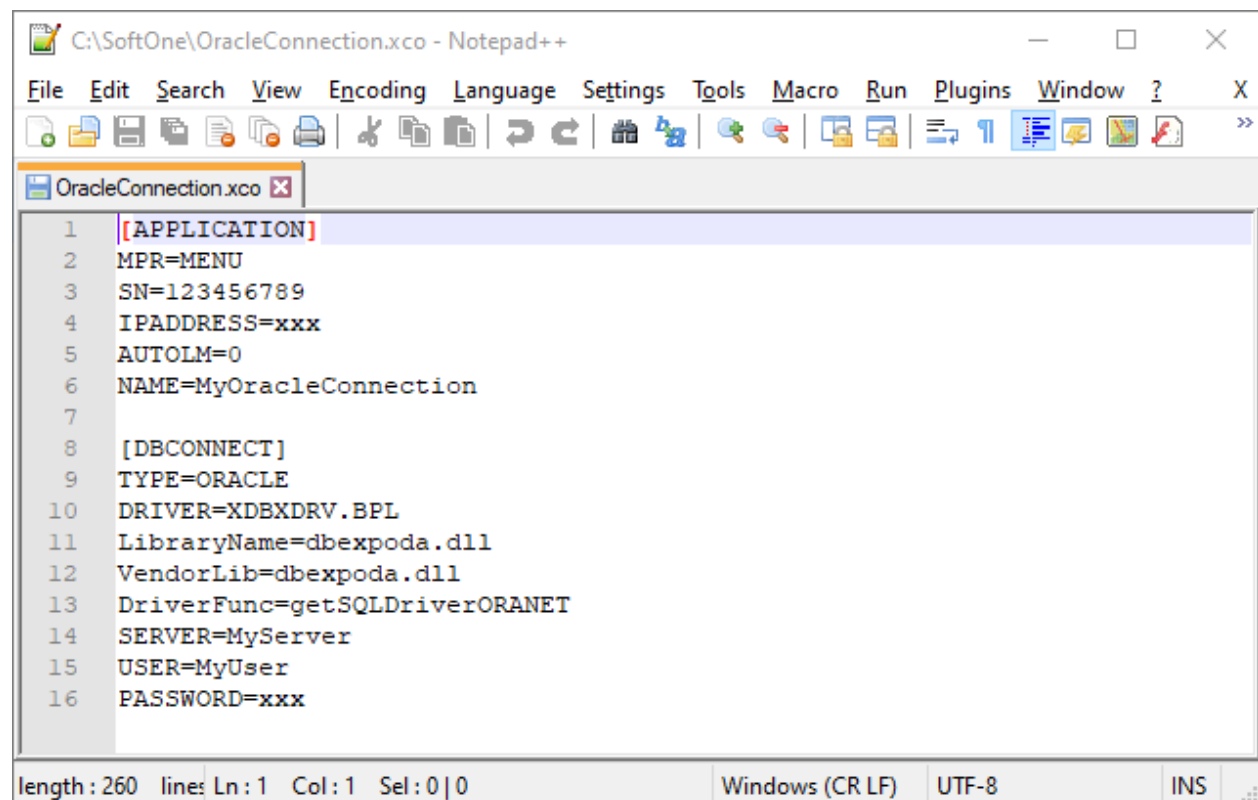
The screenshot shows a Notepad++ window titled "C:\SoftOne\Soft1_Developers.XCO - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, Window, and Help. The toolbar contains various icons for file operations and editing. The file "Soft1_Developers.XCO" is open, and its content is displayed in a monospaced font. The status bar at the bottom indicates "length: 312", "lin Ln: 1", "Col: 1", "Sel: 0 | 0", "Windows (CR LF)", "UTF-8", and "INS".

```

1  [APPLICATION]
2  MPR=MENU
3  SN=1234567891
4  IPADDRESS=xxx
5  AUTOLM=0
6  NAME=Soft1_Developers
7
8  [DBCONNECT]
9  TYPE=MSSQL
10 SERVER=myServer
11 DATABASE=Soft1_Developers
12 USER=sa
13 PASSWORD=xxx
14
15 [LOGIN]
16 USERNAME=Admin
17 PASSWORD=xxx
18 COMPANY=1000
19 BRANCH=1000
20 EXEC=RETAILDOC [FORM=RETAIL, AUTOEXEC=2, MAXIMIZE=1, RELJOBS=0]

```

XCO Example – SQL Connection (Autologin & Open RETAILDOC object)



The screenshot shows a Notepad++ window titled "C:\SoftOne\OracleConnection.xco - Notepad++". The menu bar and toolbar are identical to the previous screenshot. The file "OracleConnection.xco" is open, and its content is displayed in a monospaced font. The status bar at the bottom indicates "length: 260", "lines Ln: 1", "Col: 1", "Sel: 0 | 0", "Windows (CR LF)", "UTF-8", and "INS".

```

1  [APPLICATION]
2  MPR=MENU
3  SN=123456789
4  IPADDRESS=xxx
5  AUTOLM=0
6  NAME=MyOracleConnection
7
8  [DBCONNECT]
9  TYPE=ORACLE
10 DRIVER=XDBXDRV.BPL
11 LibraryName=dbexpoda.dll
12 VendorLib=dbexpoda.dll
13 DriverFunc=getSQLDriverORANET
14 SERVER=MyServer
15 USER=MyUser
16 PASSWORD=xxx

```

XCO Example – Oracle Connection

N. PARAMS.CFG: Cloud Connection Settings

The tables below contain the different sections and arguments that can be used in SoftOne connection file Params.cfg either for cloud or on-premise installations.

Params.cfg		
Section [PARAMS]		
Command	Operation / Usage	Example
LANG:xxx	Defines SoftOne application language	LANG:ENG
SERVER:XXX	Runs SoftOne Application Server using XXX IP address or HostName	X:\...\Xplorer.exe /server:XXX
HOST:XXX	Runs SoftOne application as client and connects to application server XXX (HostName or IP address).	X:\...\Xplorer.exe /host:XXX
PORT:XXX	Changes the TCP/IP port of the Application Server to XXX	X:\...\Xplorer.exe /server /port:XXX
SAAS:saas.azure.oncloud.gr	Connection URL. Always use it for out-process applications	
NETLIB=SoftOne.Lib.dll	SoftOne Library for custom .NET in-process or out-process applications. Note: Libraries S1Interop.dll and S1System.dll are deprecated.	
NETDLL=<.NET dll Filename>	Name of the custom .NET in-process file It's recommended to add it inside the filename.xco in Custom SDK Files	NETDLL=MyNETdll.dll
ADDON=<Delphi dll Filename>	Name of the custom Delphi in-process file	ADDON=MYDLL.dll
CLOUDUSRTIMEOUT=(hrs)	Specifies the hours that the client application will be terminated due to user inactivity. Default value is 3 hours.	CLOUDUSRTIMEOUT=8
TCPPING:(ms)	Defines the interval ping time (ms) that clients check if application server is alive. Used only in client/server installations . Default value is 40000.	TCPPING=60000
TCPNONEWS:(ms)	Specifies the time (ms) that clients will be terminated (killed) from application server due to user inactivity. Used only in client/server installations and must always be greater than the value of TCPPING. This functionality is disabled by default and only enabled if the parameter is defined inside PARAMS.cfg. This means that by default, clients are not terminated due to user inactivity (on-premise installations).	TCPNONEWS=90000

Params.cfg		
Section [SAAS]		
Command	Operation / Usage	Example
SectionName=DisplayName	<p>SectionName: Name of the section that will be used to define the login params (see next table). More than one sections can be defined inside a Params.cfg file and will be displayed in SoftOne Login window (like .XCO files).</p> <p>DisplayName: Name that is displayed inside SoftOne Connection drop down list (SoftOne Login window)</p>	MyInstallation=MyInstallation

Params.cfg		
Section [InstSectionName] (Example [MyInstallation]) or [SAASSYSTEM] for single connection		
Command	Operation / Usage	Example
USERNAME= <SoftOne UserName>	SoftOne installation serial number	USERNAME=Admin
PASSWORD= <SoftOne UserPassword>	SoftOne installation password	PASSWORD=xxx
DONTSHOWAGAIN=1	Hides the Cloud Login Window	

```

1  [PARAMS]
2  SAAS:saas.azure.oncloud.gr
3
4  [SAAS]
5  S1MyDB1=MyDBName1
6  S1MyDB2=S1MyDBName2
7
8  [S1MyDB1]
9  USERNAME=00000000001
10 PASSWORD=xxx
11 DONTSHOWAGAIN=1
12
13 [S1MyDB2]
14 USERNAME=00000000002
15 PASSWORD=xxx
16 DONTSHOWAGAIN=1

```

length: 216 lines Ln: 13 Col: 10 Sel: 0 | 0 Windows (CR LF) UTF-8 INS

Params.cfg Example

O. Internal Object Parameters

The table below contains internal parameter commands that can be used inside SoftOne objects.

They can be added in "General Settings" tab under SoftOne Objects' "Configuration" or executed through JavaScript code. **Examples:** X.SETPARAM("KEEPHANDPRC","1") or obj.SETPARAM("WARNINGS","OFF").

Internal Object Parameters		
Object / Module	Command	Operation / Usage
All	NOMESSAGES=1	Disables the display of warning messages that rise from SoftOne operational parameters (e.g. Zero Quantity warning).
All	WARNINGS=OFF	Disables the display of warning messages, that rise from "WARNING" fields (as in Customer, Item objects).
SALDOC	NOPROCDEF=1	Disables the execution of credit control scenarios.
SALDOC	NOCHECKLIMITS=1	Disables item quantity checks.
SALDOC	ASKTOCHANGEPOLICY=1	Displays "Change policy" confirmation message.
SALDOC	KEEPHANDPRC=1	Maintains manually added item price (Field "ITELINES.PRICE") in cases of document recalculation.
SALDOC	KEEPHANDSRVPRC=1	Maintains manually added service price (Field "SRVLINES.PRICE") in cases of document recalculation.
SALDOC	KEEPHANDDISCPRC1=1	Manually added item discount (DISC1PRC) is not overwritten in case of recalculation.
SALDOC	KEEPHANDDISCPRC2=1	Manually added item discount (DISC2PRC) is not overwritten in case of recalculation.
SALDOC	KEEPHANDDISCPRC3=1	Manually added item discount (DISC3PRC) is not overwritten in case of recalculation.
SALDOC	NOBONUSPOINT=1	Disables bonus points calculation (on document post).
SALDOC	AUTOCOVERUSE=1	Enables "Auto Coverage mode" using Company Branch (AND F.BRANCH="+FINDOC.BRANCH)
SALDOC	AUTOCOVERUSE=2	Enables "Auto Coverage mode" using Customer Branch (AND F.TRDBRANCH="+FINDOC.TRDBRANCH). If Customer Branch is null then it covers the corresponding documents that have null TRDBRANCH (AND F.TRDBRANCH is null).
SALDOC	AUTOCOVERUSE=3	Enables "Auto Coverage mode" using Company branch and Customer branch.
SALDOC	AUTOTRANSKEEPEXP=1	Sales documents that are created from auto conversion (Series flag) "keep" the expense data (EXP) of the converted document.
SALDOC, PURDOC	NOSERIESREEVAL=1	Disables document recalculation when changing the data of "SERIES" field.