

---

# **RFgen Software**

## **All Editions**

### **Release 5.0**

**A Product Of:**  
**The DataMAX Software Group, Inc.**  
**1101 Investment Blvd., Suite 250**  
**El Dorado Hills, CA 95762**  
**(916) 939-4065**

**Copyright 2015, The DataMAX Software Group, Inc.,**  
**All Rights Reserved**



# TABLE OF CONTENTS

<b>Introduction .....</b>	<b>1</b>
Mobile Development Studio Overview .....	1
Mobile Services Manager Overview.....	3
Mobile Services Dashboard Overview .....	3
Windows (Graphical) Clients .....	4
Before You Begin .....	4
Loading the Software .....	4
Basic Implementation Steps.....	6
<b>Configuring RFgen Software .....</b>	<b>8</b>
Configuring Mobile Application Database .....	8
Configuring Application Services.....	13
Configuring Desktop Preferences .....	16
Configuring Environment Properties .....	17
Configuring the Performance Monitoring.....	21
Configuring the Event Logs.....	22
Configuring the Scripting Environment.....	23
Configure Source Control Integration.....	25
Configuring System Queues and Tasks.....	26
Configuring Database Connections.....	27
Configuring ODBC Data Sources.....	35
Configuring a Screen Mapping Connection.....	38
Configuring an ERP Connection .....	38
Configuring a Web Connection .....	40
Configuring Transaction Management .....	42
Download Enterprise Objects.....	42
View Enterprise Objects.....	46
<b>Mobile Development Studio Menu Bar .....</b>	<b>49</b>
Testing .....	51
Application Testing .....	51
Transaction Testing .....	54
SQL Query Testing .....	56

Devices .....	58
Deploy Mobile Applications .....	58
Remote Application Explorer .....	61
Remote Database Explorer .....	63
Remote SQL Explorer .....	63
Remote Log Viewer .....	64
Utilities.....	65
Export or Import Mobile Applications .....	65
Find in Application Scripts .....	67
Replace in Application Scripts .....	68
Undo Save/Delete Action .....	69
Reports.....	70
Application Statistics.....	70
Event Logs.....	72
Performance Log .....	72
Script Validation.....	73
Task List .....	73
Upgrade Log .....	74
View Transaction Queues .....	74
Help.....	76
<b>Mobile Development Studio Tools .....</b>	<b>77</b>
Solutions Tab Overview .....	77
Users List.....	78
Menu List .....	79
Applications List.....	82
VBA Modules List .....	128
Transactions List.....	130
Hosts List.....	132
Resources Tab Overview .....	133
Device Profiles.....	133
Image Resources.....	143
Configuring Mobile Device Themes.....	145
Translations .....	158
Vocollect Tasks.....	159
<b>Screen Mapping .....</b>	<b>161</b>
How to Make Screen Mapping Work .....	161
Theory of Operation .....	161
Programming Philosophy .....	162
Design Considerations .....	162
Screen Mapping Levels .....	163
Logon Security Considerations.....	164
System Integrity Considerations.....	164

Keyboard/Special Key Configurations .....	165
Runtime Environment Variables .....	165
<b>Configuring the Host Connection .....</b>	<b>166</b>
Scheduled Downtime .....	171
VT220 Key Mapping .....	172
<b>Hosts Tree .....</b>	<b>173</b>
<b>Building a Start Menu Macro.....</b>	<b>174</b>
The Send Keys Selection .....	176
Start Menu Macro – Record System Signon .....	176
Start Menu Macro – Identify System Menu .....	178
Start Menu Macro – Record System Signoff .....	179
Start Menu Macro – Record Error Recovery .....	179
Start Menu Macro – Test Scripts .....	180
<b>Building a Host Screen Macro.....</b>	<b>181</b>
Host Screen Macro – Go to the Application Screen .....	182
Host Screen Macro – Identify the Application Screen .....	182
Host Screen Macro – Return to the Main Menu .....	183
Host Screen Macro – Test Scripts .....	183
<b>Building a Host Transaction Macro .....</b>	<b>184</b>
<b>Recording Options .....</b>	<b>190</b>
<b>Building a Screen Mapping Application.....</b>	<b>191</b>
<b>Mobile Enterprise Service Management .....</b>	<b>193</b>
To Permanently Activate the Service .....	195
<b>Mobile Enterprise Dashboard .....</b>	<b>197</b>
Active Clients View.....	198
Active Connections View.....	201
<b>Transaction Management Dashboard .....</b>	<b>204</b>
<b>User Management Console.....</b>	<b>206</b>
User Accounts.....	206
Menu Tab .....	207
<b>Mobile Devices .....</b>	<b>210</b>
Windows Desktop Client .....	210
Thin Client .....	210
Mobile Client .....	210
Mobile Device Management.....	211
Auto Upgrade Clients .....	212
RFgen Mobile Configuration .....	212

RFgen Mobile Client.....	222
<b>Visual Basic Scripts .....</b>	<b>224</b>
Global User Defined Subroutines and Functions .....	224
Using External ActiveX files in the VBA Environment .....	225
VBA Global Variables/Objects.....	225
VBA Declarations .....	226
VBA Events .....	226
Click .....	226
GotFocus .....	226
KeyPress .....	226
Initialize .....	227
Load.....	227
Lost Focus .....	227
OnBackup .....	227
OnConnect.....	228
OnCursor .....	228
OnDisconnect .....	228
OnEnter .....	229
OnEscape .....	229
OnFkey .....	229
OnInRange .....	230
OnLocale .....	230
OnMenu .....	230
OnOutOfRange.....	231
OnReadData.....	231
OnRefresh .....	231
OnReturn .....	232
OnRowColChange.....	232
OnScan .....	232
OnSearch.....	232
OnTimer.....	233
OnUpdate .....	233
OnVocollect .....	233
Terminate.....	234
Unload .....	234
<b>VBA Language Extensions .....</b>	<b>235</b>
Prompt Specific Extensions.....	235
Align.....	235
Autosize .....	235
BackColor1 .....	236
BackColor2 .....	236
BackGradient .....	237

BorderStyle .....	237
Caption.....	237
Checked.....	238
Defaults.....	238
DisplayOnly.....	238
Edits .....	239
ErrMsg .....	239
FieldId .....	239
Font.Bold.....	240
Font.Italic .....	240
Font.Size.....	240
Font.Underline .....	241
ForeColor .....	241
Format.....	241
Height.....	242
Image.Bitmap.....	242
Image.DisplayMode .....	243
Image.GetBitmap .....	243
Image.LoadResource .....	244
Image.Path.....	244
Index .....	244
Label.Align .....	245
Label.AutoSize .....	245
Label.BackColor1 .....	245
Label.BackColor2 .....	246
Label.BackGradient .....	246
Label.BorderStyle .....	246
Label.Font.Bold.....	247
Label.Font.Italic .....	247
Label.Font.Size.....	248
Label.Font.Underline .....	248
Label.ForeColor .....	248
Label.Height.....	248
Label.Left .....	249
Label.Theme .....	249
Label.Top .....	250
Label.Width .....	250
Left .....	250
List .....	251
List.AddItem .....	251
List.AddItemEx.....	252
List.Cell .....	252
List.Cell(x,y).BackColor1 .....	253
List.Cell(x,y).BackColor2 .....	253
List.Cell(x,y).BackGradient .....	254

List.Cell(x,y).Bold.....	254
List.Cell(x,y).ForeColor .....	255
List.Cell(x,y).Italic.....	255
List.Cell(x,y).Underline .....	255
List.Cell(x,y).Value.....	256
List.Clear.....	256
List.Column.....	256
List.Column(x).Align .....	257
List.Column(x).AutoSize.....	257
List.Column(x).BackColor1 .....	257
List.Column(x).BackColor2 .....	258
List.Column(x).BackGradient.....	258
List.Column(x).Bold .....	259
List.Column(x).Caption .....	259
List.Column(x).DisplayOnly .....	259
List.Column(x).ForeColor .....	260
List.Column(x).Format .....	260
List.Column(x).ImageMode .....	261
List.Column(x).Italic .....	261
List.Column(x).ScaleDecimals.....	262
List.Column(x).TrimSpaces .....	262
List.Column(x).Underline .....	262
List.Column(x).Visible .....	263
List.Column(x).Width .....	263
List.Columns.....	263
List.Count.....	264
List.Data.....	264
List.Index .....	264
List.InsertItem .....	265
List.InsertItemEx .....	265
List.PageDown.....	266
List.PageUp .....	266
List.RemoveItem .....	267
List.ScrollDown .....	267
List.ScrollUp.....	267
List.SetColumn .....	268
List.Sorted.....	268
List.Value(x) .....	268
PageNo .....	269
Password .....	269
Required .....	269
ScrollBars .....	270
SelLength .....	270
SelStart .....	270
Text.....	271

Top.....	271
Type.....	272
ValField.....	272
ValTable.....	273
Visible .....	273
Width.....	274
<b>Application-Based Extensions.....</b>	<b>274</b>
Balloon.....	274
CallForm .....	275
CallMacro.....	275
CallMenu.....	276
ChangeLoginForm .....	276
ChangeUserPassword.....	277
ClearValues .....	277
ClientType.....	277
ConnAvailable.....	277
ExecuteMenuSelection.....	278
ExitForm.....	278
ExitSession.....	278
GetInput .....	278
GetString.....	279
GetValue.....	280
IpAddress.....	280
Locale .....	280
LogError.....	281
LogErrorEx.....	281
MakeList.....	282
MsgBox .....	282
PromptCount.....	284
PromptNo.....	284
SendChar.....	284
SendKey .....	284
SetDisplay.....	285
SetFocus.....	286
SetMenu.....	287
SetValue .....	287
ShowList .....	287
Sleep.....	288
TimerEnabled .....	288
TimerInterval.....	288
User .....	289
UserProperty.....	289
SQLNum .....	289
<b>Menu Strip Extensions.....</b>	<b>290</b>
AppendItem .....	290

Clear .....	290
Count .....	291
Item .....	291
RemoveItem .....	291
RemoveItemByName .....	291
Reset .....	292
SetItem .....	292
Show .....	292
Screen Display Extensions.....	293
Bell.....	293
Clear .....	293
ClearEOL.....	293
ClearEOP.....	294
DrawLine.....	294
Height .....	294
Print .....	295
Refresh .....	295
ResetCursor.....	295
ReverseOff.....	295
ReverseOn.....	296
Width.....	296
Soft Input Panel Extensions .....	296
GetCurrentType .....	296
GetTypes .....	296
Mode .....	297
SetType .....	298
Show.....	298
Server-Based Extensions.....	298
CallMacro.....	298
CommandTimeout .....	299
Connect .....	299
Disconnect.....	300
ExecuteSQL.....	300
GetTable .....	300
IsConnected.....	301
MakeList .....	302
Ping.....	302
QueueMacro .....	303
ReadFile .....	303
SendQueue.....	304
SendTable .....	304
SetCredentials .....	304
SetHost .....	305
SetVPN .....	305
SetWAN.....	306

ShowProgress .....	306
SyncApps.....	306
WriteFile.....	307
<b>System Error Extensions.....</b>	<b>308</b>
AddError.....	308
AppName .....	308
Clear .....	308
Count .....	308
Description .....	309
DevGUID.....	309
IpAddress.....	309
NativeError.....	310
Number .....	310
UserName.....	310
<b>System Level Extensions .....</b>	<b>311</b>
ConnectionProperty .....	311
DeleteProperty .....	316
DisableTimeout.....	316
EnvironmentProperty .....	316
GetConnection.....	317
GetProperty.....	317
SendMessage.....	318
SetProperty .....	318
UsePixels .....	319
UserList.....	319
ValidateWinUser.....	319
<b>Database Related Extensions .....</b>	<b>320</b>
BeginTrans.....	320
CommitTrans .....	321
Count .....	321
Execute .....	321
Extract.....	322
MakeList.....	322
OpenResultset .....	323
RedirectDataSource .....	325
RollbackTrans .....	325
SaveBitmap .....	326
UseDataSource .....	326
<b>Mobile Device Extensions .....</b>	<b>327</b>
ClickAndSkipPrompts .....	327
ClickCoordinates.....	327
EnableGPS .....	328
ForceLocal .....	328
GetGPSInfo .....	328
GoOffline.....	329

GoOnline.....	330
IsOffline.....	331
IsOnline.....	331
Platform .....	331
PlaySound .....	332
PrinterOff .....	332
PrinterOn .....	332
ReadFile .....	333
Send .....	333
SendCommPort .....	333
SetCameraOption.....	334
SetCommMode.....	334
SetCommPort .....	334
TakePicture.....	335
WriteFile.....	336
DeviceObject .....	336
Create .....	337
Execute.....	337
LastError.....	338
Name .....	338
Release.....	338
ReturnValue.....	338
<b>Transaction Management Extensions .....</b>	<b>339</b>
AbortTrans .....	339
CheckStatus .....	339
GetItems .....	339
GetItemsEx.....	340
MacroName .....	341
MoveQueue .....	342
QueueMacro .....	342
QueueName .....	343
SeqNo.....	343
<b>Enterprise Resource Planning Extensions .....</b>	<b>343</b>
BeginTrans .....	343
CommitTrans .....	344
LogOff .....	344
LogOn .....	344
MakeList .....	346
ReadData.....	347
RollbackTrans.....	347
SetHardRelease .....	348
SetSession.....	348
<b>Printer Extensions .....</b>	<b>348</b>
Activate .....	348
Copies.....	349

EndDoc .....	349
FontBold.....	350
FontItalic .....	350
FontName .....	350
FontSize.....	350
FontStrikeThru .....	351
FontUnderline .....	351
GetName.....	351
NewPage .....	351
Orientation .....	352
PageWidth .....	352
Print.....	353
PrintQuality .....	353
PrintRaw .....	354
<b>Screen Mapping Extensions .....</b>	<b>354</b>
BeginTrans.....	354
CommitTrans .....	355
Connected.....	355
CurScreen.....	355
FindText.....	356
GetArea.....	356
GetAttribute.....	357
GetBackColor .....	358
GetCursor .....	358
GetForeColor .....	359
GetText .....	359
GoToScreen .....	360
IsScreen.....	360
LogOff .....	360
LogOn .....	361
PadInput.....	361
PingHost .....	362
ResetConnection .....	362
SendCTRL .....	362
SendCTRLAlt .....	363
SendKey .....	363
SendKeyAlt .....	364
SendText.....	364
SendTextAlt .....	365
SessionID.....	365
SessionPwd .....	366
SessionUser .....	366
SetBase .....	366
SetCursor.....	367
SetDelay .....	367

SetSession.....	367
SetTimeout .....	368
WaitForCursor .....	368
WaitForCursorMove .....	368
WaitForHost.....	369
WaitForScreen.....	369
WaitForText .....	370
WaitForWrite.....	371
<b>Chart Object .....</b>	<b>371</b>
AddDataPoint.....	372
AxesFont.....	372
AxesLabel.....	373
AxesTitleFont.....	373
BackColor .....	373
ForeColor.....	373
Header .....	374
HeaderFont.....	374
Height .....	374
Image.....	375
LegendFont.....	375
SeriesColor .....	375
SeriesLabel.....	375
ThreeD .....	376
Type.....	376
Width.....	376
<b>Chart Examples.....</b>	<b>378</b>
Area .....	378
Bar .....	379
Candle .....	380
FilledRadar .....	382
HiLo .....	384
HiLo OpenClose .....	386
Pie.....	388
Plot.....	389
Polar .....	391
Radar .....	393
StackingBar .....	395
Surface .....	397
<b>SearchList Object.....</b>	<b>399</b>
AddItem .....	399
Cell.....	399
Cell(x,y).BackColor1 .....	399
Cell(x,y).BackColor2 .....	400
Cell(x,y).BackGradient.....	400
Cell(x,y).Bold .....	401

Cell(x,y).ForeColor.....	401
Cell(x,y).Italic .....	401
Cell(x,y).Underline .....	402
Cell(x,y).Value .....	402
Clear .....	402
Column.....	402
Column(x).Align .....	403
Column(x).Autosize .....	403
Column(x).BackColor1.....	403
Column(x).BackColor2.....	404
Column(x).BackGradient .....	404
Column(x).Bold .....	404
Column(x).Caption .....	405
Column(x).DisplayOnly .....	405
Column(x).ForeColor .....	405
Column(x).Format.....	406
Column(x).ImageMode .....	406
Column(x).Italic.....	406
Column(x).ScaleDecimals .....	407
Column(x).TrimSpaces .....	407
Column(x).Underline.....	407
Column(x).Visible.....	408
Column(x).Width .....	408
Columns.....	408
Count .....	408
Index .....	409
List .....	409
MaxRows .....	409
Normalize .....	410
ReturnAllRows .....	410
SetColumn .....	410
ShowEmptyList .....	411
ShowList .....	411
SQL.....	412
Value(x).....	412
<b>Embedded Procedure Object.....</b>	<b>412</b>
<b>Embedded Procedures .....</b>	<b>412</b>
<b>Embedded Procedure Properties and Methods .....</b>	<b>416</b>
ClassObject.....	416
Clear .....	416
ColumnCount.....	416
ColumnName .....	417
DataSource .....	417
DebugLog .....	417
DisableParam .....	417

Execute.....	418
ExecuteMethod.....	418
LogMode.....	418
Name .....	419
Param .....	419
ParamCount.....	419
ParamEx.....	419
ParamName.....	420
Queue .....	421
QueueName .....	421
QueueOffline.....	421
QueueSeqNo .....	422
RowCount.....	422
<b>DataRecord Object.....</b>	<b>422</b>
AddNew .....	422
Clear .....	423
IsEOF .....	423
MoveFirst .....	423
MoveLast .....	423
MoveNext.....	423
MovePrevious.....	423
MoveTo.....	424
Param .....	424
ParamCount.....	424
ParamName.....	425
RowCount.....	425
Schemadl.....	426
<b>Dynamic Array Extensions .....</b>	<b>426</b>
DCount.....	426
Del .....	427
Ext.....	429
FixLeft .....	430
FixRight.....	430
Ins .....	431
LField .....	432
Locate .....	433
LocateAdd.....	434
LocateDel.....	435
Rep .....	435
RField .....	436
<b>Socket Object.....</b>	<b>437</b>
BytesReceived.....	437
LocalHostName .....	437
LocalIP .....	438
Protocol.....	438

RemoteHost .....	439
RemoteHostIP .....	439
RemotePort.....	439
State.....	440
sktClose .....	440
sktConnect.....	441
sktGetData .....	441
sktPeekData .....	442
sktSendData .....	443
OnClose.....	443
OnConnect.....	444
OnDataArrival .....	444
OnError .....	444
OnSendComplete .....	444
OnSendProgress .....	444
<b>Web Object .....</b>	<b>445</b>
ConnectTimeout .....	445
Data .....	445
DataSource.....	446
HeaderValue.....	446
QueryType .....	446
ReceiveTimeout.....	447
Reply.....	448
Request.....	448
SendTimeout .....	448
Execute .....	449
Login .....	449
Logout .....	449
<b>Stored Procedure Extensions .....</b>	<b>449</b>
CallAction (not used with Sybase).....	449
CallProc (must be used with Sybase).....	450
CallSelect (not used with Sybase).....	451
<b>Database Stored Procedure Object .....</b>	<b>452</b>
CommandText .....	452
CommandTimeout .....	452
CommandType .....	452
CreateParameter .....	453
Data .....	453
DataSource.....	454
Dict.....	454
Execute .....	454
Param .....	454
Param( ).Datatype .....	455
Param( ).Direction.....	456
Param( ).NumericScale .....	456

Param( ).Precision .....	457
Param( ).Size .....	457
Param( ).Value .....	457
ParamCount .....	457
Prepared .....	458
Results .....	458
<b>Initialization Files .....</b>	<b>459</b>
RFgen.ini .....	459
ERPDIALOGS.ini .....	461
GPRS.ini .....	462
Client.ini .....	463
<b>Index .....</b>	<b>464</b>

# Introduction

The RFgen Mobile Enterprise Application Platform, Mobile Development Studio, Mobile Enterprise Application Server and Mobile Administration Console are Microsoft Windows-based software programs that give numerous 'Data Base Management System' (DBMS) and 'Enterprise Resource Planning' (ERP) system users the ability to program advanced 'Radio Frequency Data Collection' (RFDC) applications via an easy-to-use development environment.

The Mobile Development Studio is a 'point-and-click' remote / wireless data collection applications generator that allows programming results to be achieved in hours, instead of days or weeks. The system is structured to interface with systems using: (1) ODBC/SQL, (2) Host Screen Mapping protocols, (3) XML and (4) select ERP packages. A major advantage of the Mobile Development Studio is that no investment in hardware is required until users actually place their programs into production use.

The Mobile Services Manager enables applications written with the Mobile Development Studio to work in a multi-user mode with remote (typically wireless) data entry devices. It can also be used to manage data connections, transaction processing and the stopping and starting of data collection sessions. Hard-wired and mobile devices are also supported.

The Mobile Services Dashboard is the software administrators use to monitor and interact with users as well as see metrics of the users and connectors.

## Mobile Development Studio Overview

The Mobile Development Studio is the software used for the design phase of the mobile applications. It is important to note that most corporate databases contain data tables (and procedures) that would be useful in supporting data collection programming efforts. Mobile Development Studio can utilize database table 'fields' (i.e., column names and their associated properties) to determine the makeup of a database, and to allow data collection system developers to automatically interface with it. A preliminary step is to download table fields and properties from a database to a PC-based RFgen development system (see RFgen Basic Implementation Steps).

Thereafter, on the PC, a developer may construct appropriate data entry screens by pointing and clicking on the downloaded items, including dragging and dropping the items onto a simulated display (see The Mobile Development Studio Applications). A similar methodology is available for working with ERP business rules and procedures (BAPIs and business functions) when using the RFgen 'Integration Suites' for SAP, Microsoft Dynamics, Oracle EBS and JD Edwards / Oracle.

The visual components of an RFgen-based development process are established via the Mobile Development Studio Applications, Menus, and Users development modules. Each module appears as a heading in the Design tree in the Mobile Development Studio main screen display.

Many features of an RFgen development system are available as 'properties'. For example, numerous data entry features (including data defaults, data edits/validations, table lookups, etc.) have already been pre-programmed. The programming language used by RFgen is compatible with Visual Basic for Applications (VBA), the language used by Microsoft in its suite of 'Office' products.

When RFgen-based development efforts are complete, applications run closely coupled with remote devices, under Windows control, with access to corporate databases being 'SQL-driven'. Data processing tasks such as opening database(s) and tables, selecting and updating data records, list box generation, data validations/edits, executing custom SQL statements, VBA program execution, and calling stored procedures are automatically accomplished by RFgen. In a Thin client mode, RFgen-based Application, Menu, and User objects are stored and executed on a Windows-based server, not on remote data collection devices. The benefits associated with this mode of operation are: (1) increased processing power, (2) the availability of higher level programming tools, (3) ease of system deployment, and (4) ease of system maintenance; i.e., program changes are deployed instantly, without having to flash or re-flash remote devices. In a Mobile client mode development items are stored on the mobile devices.

Organizationally, data entry Applications are listed on Menus and those menus are made available to data collection device users. All aspects of a development effort may be thoroughly tested in a local (non-network) mode (see the Application Testing option) prior to being placed into service.

## **Mobile Services Manager Overview**

Once you have constructed and tested your system with the Mobile Development Studio, your programming effort is essentially complete. All that remains to activate your RFgen network is to install the Mobile Enterprise Application Server on your Windows server, configure your hardware, and test your complete system.

The Mobile Services Manager is a session enabler and manager that allow data collection devices to interact with databases, or (optionally) with legacy/Host screens and ERP packages, in a multi-user/pooled-connection(s) mode, using the programs developed by the Mobile Development Studio. Active data devices may be viewed and managed from the Services Manager.

Technically, a server-based 'client' task is spawned for each remote device as it logs in. The Mobile Services Manager works in both character and graphical modes simultaneously to support a multitude of remote devices. Character-based devices use standard telnet-based 'VT220' communications. Graphical clients use the RFgen 'graphical client' software that must be loaded on the remote graphical device.

To activate a remote device, ensure that the Mobile Services Manager service is running and configure the telnet program to point to the IP address or name of the RFgen server. For more information, see the Mobile Services Manager section.

## **Mobile Services Dashboard Overview**

The Mobile Services Dashboard is an optionally running program that displays all of the connected users to the RFgen server. Additionally, there is the ability to interact with the client's session or simply observe the work being done. Temporarily stopping one or many users and permanently shutting down one or many users provides complete control over the RFgen network. Managing the queuing capabilities is also done from a dashboard application. The queue display allows for transactions that have been queued, completed successfully or have failed to be edited and reposted for another attempt at placing the data into the backend system.

# **Windows (Graphical) Clients**

Note: If you are not currently using Windows desktop PCs or Windows CE-based devices, the information contained in this section may be skipped.

To use Windows-based PCs or mobile devices, the RFgen CD includes both the Windows Desktop Client installation and the Mobile / Wireless Clients installation. These applications take advantage of the Windows operating system and allow special controls like images, signature capture controls or command buttons to be used in an RFgen session. Standard telnet services cannot support these types of prompts. If there is a reason for running multiple copies of the Windows Desktop Client on the same PC, there is a command line option (-IgnoreGUID) that will allow this.

Note: not all PC-based telnet programs support function keys, in particular the telnet program provided by Microsoft with Windows 2000. The Windows Desktop Client supports function keys, and is a great alternative to the use of character-based telnet programs.

## **Before You Begin**

To best use the Mobile Development Studio, it is necessary to understand the basic concepts of your Data Base Management System (DBMS), or for screen mapping applications, the structure and use of your IBM or UNIX-based (legacy) host system. Knowledge of data structures is of particular importance for database applications since it is necessary to understand the basic concepts of 'tables', 'fields/columns', and 'data types' prior to creating applications. Understanding Structured Query Language (SQL) 'syntax' is also helpful with database applications. Experience with the Microsoft Visual Basic/VBA programming language is helpful in the development of advanced data collection applications, for use with both SQL databases and legacy host-based applications.

## **Loading the Software**

The RFgen CD-ROM is set to 'AutoPlay' when the CD is inserted into your CD-ROM drive. If the CD does not AutoPlay, click Start, Run, and open 'CDSetup.exe' on your CD-ROM drive. Selections include:

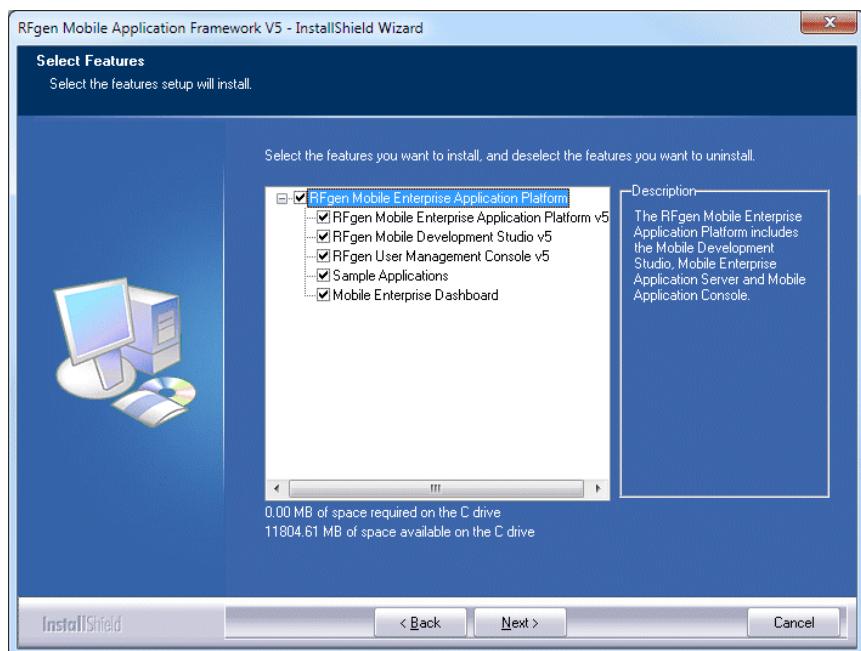
### **Install Mobile Enterprise Application Platform**

## Install Windows Desktop Client

### Install Mobile Clients

By default, all RFgen Software items are loaded to your \Program Files\RFgen directory or possibly Program Files (x86). A Microsoft Access database called 'RFgen.mdb' is also loaded to C:\ProgramData\RFgen5 directory and contains a sample application, if it was chosen for installation. This database houses the application programs / objects written with the Mobile Development Studio.

The Mobile Enterprise Application Platform install includes all RFgen components except for: the Windows Desktop Client application and the mobile client files. By default, each of the components shown here are checked for installation.



The Mobile Enterprise Application Platform (for licensed RFgen users only) includes several components. Mobile Application Server, Mobile Development Studio, User Management Console, Transaction Management Dashboard, Mobile Enterprise Dashboard, User Management Console, sample applications, client files and the Load Balancing service.

The Mobile Application Server allows multiple remote devices to use RFgen. It is the graphical user interface to the Mobile Enterprise Application Server program that resides in the Windows System Tray. This program allows the Mobile Enterprise Application Server to be started manually or automatically, as a Windows ‘task’ or as a Windows ‘service’ when your system is booted. Other options include starting the Mobile Administration Console and suspending or stopping all server activity.

The Mobile Development Studio is the development environment that is optional for production servers. The User Management Console provides minimal configuration access from the server. It allows the creation of menus and the assignment of users to menus only.

The Mobile Services Dashboard allows administrators to view and manage remote devices. The Client file is started for each connection made by a user. This isolates each user and helps protect the system as a whole.

The **Windows Desktop Client** is a ‘graphical alternative’ to the standard Windows ‘telnet.exe’ character-based interface and currently has its own install program. Objects such as command buttons, signature capture boxes and images are available for the application when using a graphically-capable device.

The **Mobile and Wireless Clients** are files used to create appropriate installs on a mobile-based device. Based on the processor type and operating system, RFgen will draw from these files to create a working application for each device. They have the same ‘look and feel’ as the Windows Desktop Client for (full) Windows-based systems (above).

## Basic Implementation Steps

After you have installed and configured the Mobile Development Studio, the software is normally implemented as follows for ODBC/SQL compliant databases:

1. ‘Transaction tables’ specific to your application are designed via your main database system (i.e., data to be captured is defined in your database). Typically, an individual ‘transaction table’ with appropriate field definitions is established for each data entry process.
2. Database transaction table field definitions are ‘downloaded’ to RFgen (via the Mobile Development Studio menu bar ‘Enterprise Connections / Download Enterprise Objects’ selection). These

definitions contain all the necessary information concerning the transaction data to be written to your database. No data is downloaded from the database. Table definitions may be partially edited (inside Mobile Development Studio), if required. Similarly, ERP business rules and functions may be accessed and downloaded, if using the RFgen ERP integration suites.

3. Data entry Applications are made for each transaction table. Pre-programmed data properties such as 'defaults', 'validations/edits', 'table validations', etc. are added, as required.
4. Applications are listed on Menus (e.g., menus allow users to select a multitude of Applications).
5. User IDs and passwords are defined; a menu is assigned to each user.
6. System Objects (Applications, Menus, and Users), as constructed, are tested in conjunction with your database. Customized VBA programs and/or SQL statements are added as required.
7. Your data collection hardware (RF and other devices) are configured to support character-based or graphical communications sessions with your Windows-based Mobile Enterprise Application Server.
8. The Mobile Enterprise Application Server software from the RFgen Software CD is installed on your RFgen Server to activate full (multi-user) network usage and the remote management of data collection devices.

The above steps apply to database applications where ODBC/SQL is used to communicate with a database or ERP applications where predefined business rules and functions are used to update the ERP system.

For 'screen mapping' applications that interact with legacy host screens, see the RFgen Screen Mapping section.

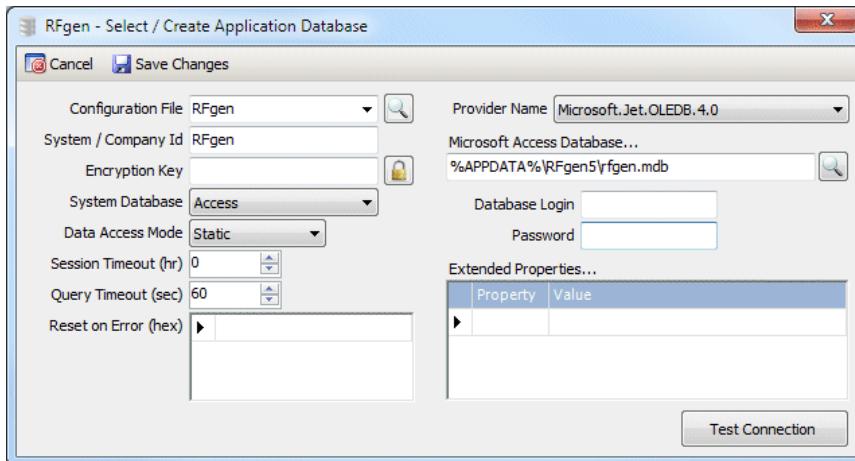
# Configuring RFgen Software

Once the Mobile Development Studio has been loaded, it must be configured. From your Windows desktop, execute the Mobile Development Studio software as follows:

- Click on 'Start'
- Click on 'Programs', then click on 'RFgen v5.0.'
- Click on 'Mobile Development Studio'

## Configuring Mobile Application Database

To configure a Mobile Application Database, click on the Configuration menu bar selection, then Application Database menu selection. The following window will appear.



By default, a Configuration File called 'RFgen', defines the profile of the solution database, as shown in the window above.

This **Configuration File** was created when the Mobile Development Studio software was loaded and it identifies a Microsoft Access file called 'RFgen.mdb' located in the C:\ Users \ <username> \ AppData \ Roaming \ RFgen5 directory as the database that contains the

programming items (Applications, Menus, Users and VBA code) written with the Mobile Development Studio, including the pre-written example/demo items. This is only the default place where the sample applications are deployed. It is not necessary to use this location.

The **System / Company Id** field is used to describe the owner of the configuration file. Since there may be many configuration files referencing different databases for different customers or copies of the same customer's database, this field acts as the description.

An '**Encryption Key**' entry provides users the ability to encrypt their Application Database. This feature, a system option starting with release 3.0, allows the database to be locked so that users may not view or modify Application objects or VBA scripts.

When active, a unique key may be entered in the Application Database selection window to lock, encrypt, unlock, or decrypt the database.

To encrypt the database: enter a key (e.g. 'abcdef') in the Key textbox, and click the Encrypt button with the lock icon.

To lock the database: display the panel again, remove the key and click 'Save'. The database is now locked. Applications will execute but may not be accessed.

To unlock the database: display the panel and again enter the key, click 'Save'. The database will be unlocked.

To decrypt the database: enter the key, click the Encrypt button and click 'Save'. The database will be decrypted and unlocked.

You must not export encrypted applications to a non-encrypted MDB. The server will prompt for the password and decrypt the exported application.

The **System Database** drop down field selects which type of database is to be used to host the solution objects. Changing this value changes the window to show database specific configuration fields. The server supports Access, SQLite, SQL Server and Oracle as database containers. The solution stores the information to connect to these databases in an "rcf" file. You can also select these rfc files when exporting / importing to that database container.

**Data Access Mode** sets the cursor to either Static or Dynamic when retrieving data from the database. Usually, Static is best because it is fast and safe. However, if you have a database like Pervasive that will

actually make a copy of the data from the database system to the RFgen system when using a static cursor, you can change this option to Dynamic, so performance will not suffer. Internally, this sets the cursor option to either adoOpenStatic or adoOpenDynamic.

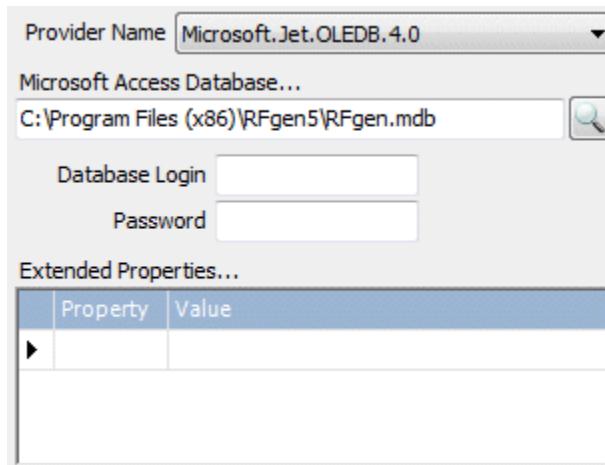
The **Session Timeout** value (in hours) will disconnect and reconnect to the database at the specified interval. This may be required if the database is configured to not allow a connection that never times out.

The **Query Timeout** specifies how long the server should wait before giving up on the ODBC driver to come back with a response.

**Reset on Error** is a list of hex values that if returned by the ODBC driver will cause a reset of the connection. The process for adding a value is to first get the error number from the error log.

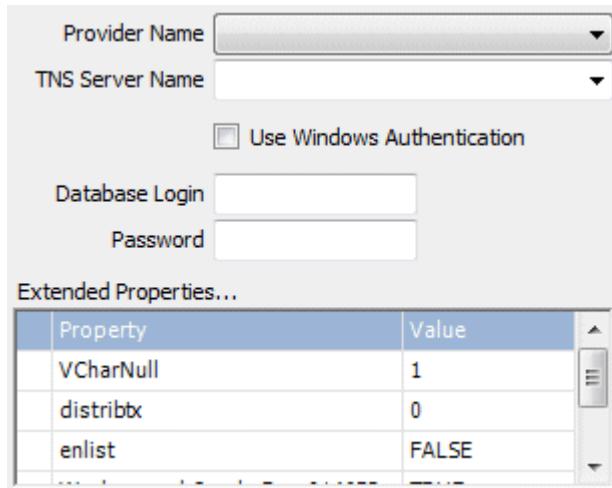
Example: the error log shows -21456327. Use the Windows calculator in Programmer mode, select Dec and Dword options, enter the number and if you need the negative sign use the ± button to change its sign. Then click the Hex option. You should get: FEB89A39. Enter this value into the box with a "0x" prefix like: 0xFEB89A39

The **Provider Name** selection will depend on the type of database you want to use. Note that these providers are not necessarily installed. All provider options must already exist on the server to be used.

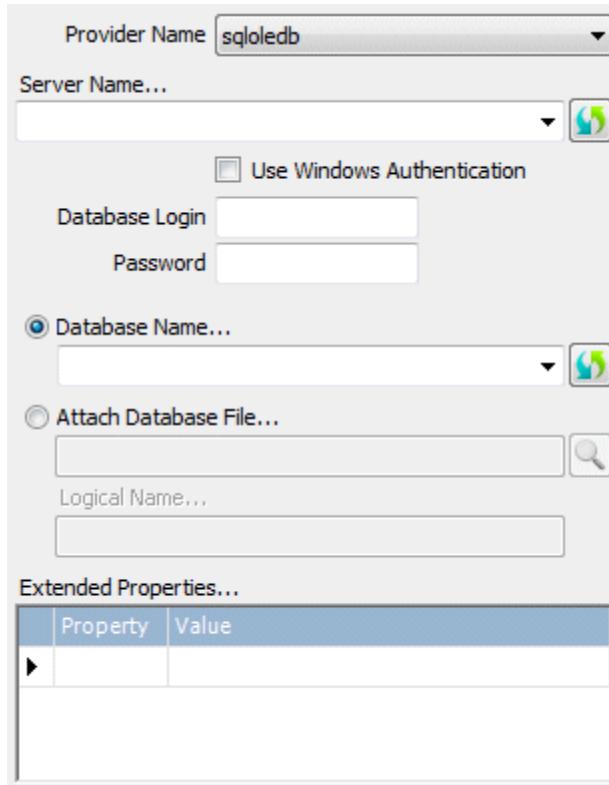


For an Access database select the appropriate Provider Name for the type of system (32 bit or 64 bit).

The path, login, password and extended properties are then used to make the connection. In the case of Access most of these fields are not necessary.



In the case of Oracle ODBC is not used but the TNS Server Name points to the Oracle server. Also specify the Provider Name and review the Extended Properties for accuracy. The Use Windows Authentication option will take advantage of the Active Directory when connecting to the database.



For SQL Server specify the Provider Name, Server Name and Database Name. The Use Windows Authentication option will take advantage of the Active Directory when connecting to the database. If you want to connect directly to the MDF file itself, specify the Attach Database File option and locate the database file directly. The Logical Name is typically the filename without a file extension and should not be necessary. The Extended Properties are usually not required.

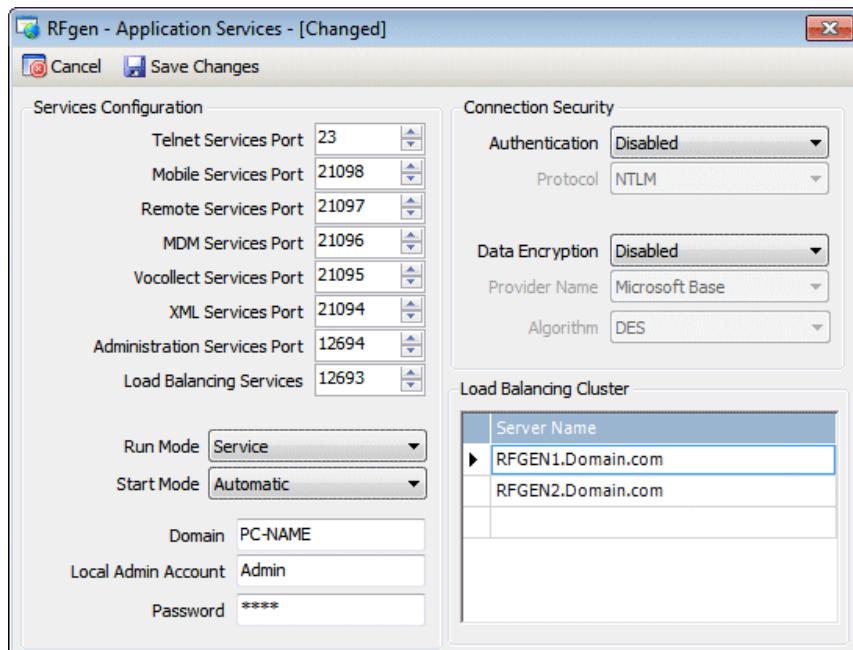


For SQLite database connections just specify the DB file itself. There are no other settings. You can specify a location and name that does not exist and clicking the Test Connection button will create the database for you.

Finally click on the **Test Connection** button to verify connectivity. If the database has not already been setup to support the solution tables they

will be created at this time. Clicking the Save Changes button will also create what is necessary but won't test the connection. Either button will also notice if the database came from an older release and ask if you want it upgraded.

## Configuring Application Services



**Telnet Services** are set to port 23 (the standard default setting for telnet servers such as RFgen). If you have another telnet server (i.e., another telnet-based application) running on your RFgen system which is listening on port 23, then your remote devices must be configured to some port other than the 23 default. That port number will have to be entered here and on the mobile device's telnet client.

The **Graphical Services Port** is the port used to transmit the graphical user interface between the development environment and the client. The default setting is port 21098.

The **Remote Services Port** is the port used to transmit the data portion of the session between the development environment and the graphical client. The default setting is port 21097.

The **MDM Services Port** is used by the server's Mobile Device Management process that deploys updated files, transactions or data to the device. The default setting is port 21096.

The **Vocollect Services Port** is the address that the Vocollect product uses to communicate with the server. Vocollect is a hardware solution that replaces barcode scanners for a speech-processing device that accepts the spoken word as data input. The default setting is port 15008. Note: create a Vocollect profile that uses 15008 for both the LUT and ODR services.

The **XML Services Port** is the address used for interfacing external / 3<sup>rd</sup> party data with server applications. The default setting is port 21094.

The **Administration Services Port** is the address that the server uses to communicate with the server's dashboard applications. The default setting is port 12694.

The **Load Balance Services Port** is used by the server to communicate with the other RFgen servers so that connected clients can share the client load. The default setting is port 12693.

The recommended configuration for running RFgen as a service is to use a local administrator to the PC where RFgen is installed. Select the **Run Mode** Service option and fill in the **Domain** field which contains the name of the local PC and the **Local Admin Account** and **Password** fields which are the credentials for a local administrator account. This typically avoids a problem where a domain admin account is believed to have enough rights but really does not.

The Service option means RFgen will be configured in the Windows services list. Selecting the **Start Mode** of Automatic will set that service to start automatically. Otherwise it will be set to manual start.

By using the Run Mode of Application and the Start Mode of Automatic, RFgen will not be configured as a service but will start RFgen as an application when a user logs in to the desktop.

### **Connection Security**

Security is broken into two categories, device authentication and data encryption settings.

Authentication is used to verify the user credentials beyond the RFgen login process. Authentication protocols currently supported are "NTLM, and SSL". If authentication is enabled then when a RFgen client first

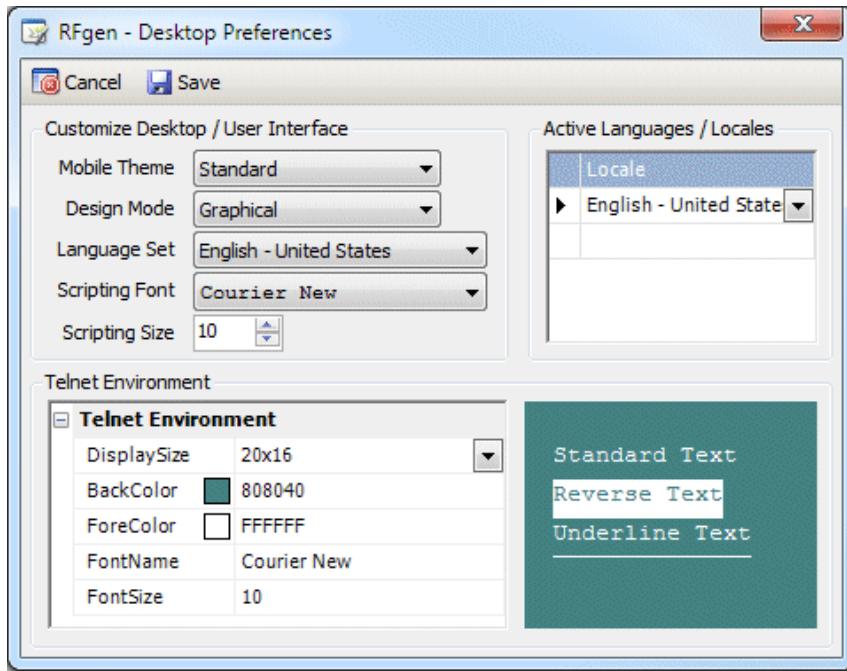
tries to connect, it will pop up a dialog box to capture user information (user id, password, and domain.) An encrypted package of this information will be sent to the configured protocol. A core Windows service on the RFgen server will attempt to authenticate the login request and accept or reject the connection.

Data encryption is used to secure the data being transmitted in the wireless environment. Microsoft provides several cryptographic choices and algorithms that are taken from what the operating system is capable of doing. The client must be configured exactly the same way as the server or it will not connect.

### **Load Balancing Cluster**

Each server is capable of sharing the client load with other servers if their server names are listed in this group. If multiple servers are sharing a single MDB solution file, it is ok to have a server's own name in this list. This feature supports load balancing as well as server failover capability. Even if server number one is authorized for 10 users and server number two is authorized for 20 users, if either server goes down, the other will allow 30 concurrent connections for a period of seven days before reverting back to its original number of users. Adding additional servers authorized for zero users in this configuration would essentially add load balancing and hot spare failover capability. Note that telnet connections and Vocollect connections are not load balanced.

# Configuring Desktop Preferences



The **Mobile Theme** option sets the default theme from the available Mobile Theme resources. This resource contains all graphical design and runtime elements.

A **Design Mode** option sets the default mode for screen development either to the text-based character style or the Windows Mobile style.

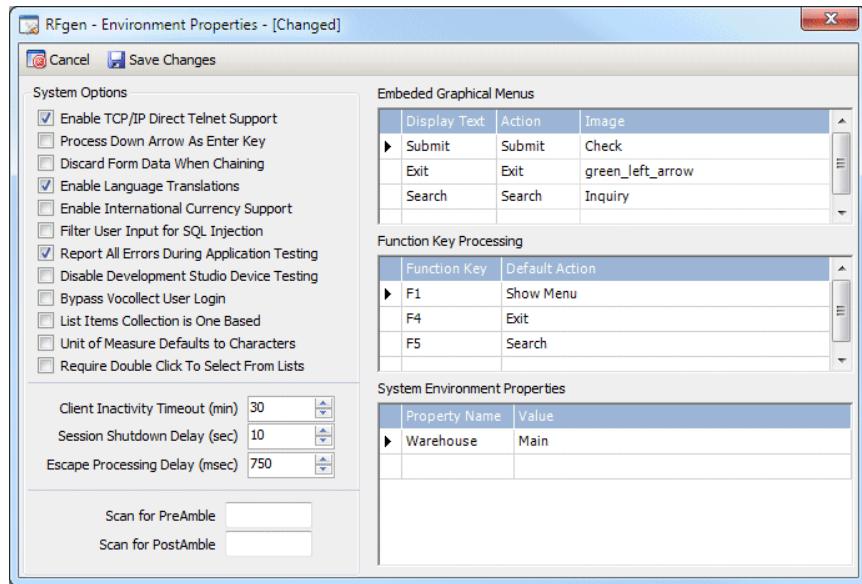
The **Language Set** option changes the expected input language character set.

The **Scripting Font** and **Scripting Size** fields affect the code windows.

The **Active Languages / Locales** can be used to override the operating system's default locale so RFgen can translate strings into foreign languages correctly. This is done using the App.GetString command with the Translations resources.

The **Telnet Environment** settings are strictly for the display of telnet devices on the server since RFgen has no control over the client.

# Configuring Environment Properties



Here, **Enable TCP/IP Direct Telnet Support** is checked, meaning that communications will be established with individual remote devices, rather than by using an intermediate 'communications controller' (accordingly if using a network communications controller, uncheck the 'TCP/IP Direct' option).

The **Process Down Arrow as Enter Key** option is not checked indicating that it will not be used as an alternative for the <Enter> key.

**Discard Form Data When Chaining** sets the current application data to null when another application is called, rather than keeping the original data in memory (the default condition) should the calling application be returned to.

**Enable Language Translations** allows for the names of the prompts on the screen to be used directly without the legacy RFPrompt wrapper function. So txtPlant.Caption = "Hello" is valid syntax.

**Enable International Currency Support** enables support for international currency formats, by using the current system regional and language option settings (e.g. \$1,234.56 becomes \$1.234,56).

**Filter User Input for SQL Injection** – This option will check the SQL statement for specific key words and remove them before sending the SQL request to the ODBC driver. All semi-colons ( ; ) are removed; any single quote ( ' ) marks are doubled; any instances of a double dash ( -- ) are replaced with a single dash; and any words specified in a user-created FilterInput.ini file stored in the RFgen installation directory will be removed. For example, if one line in the ini file had the word “select” then a user input of “select \* from inventory” would become “ \* from inventory” with the select word missing.

Note: this is done on a space separated word basis so a phrase like “selected items” would not be affected as the word “selected” is not “select”.

These are sample words a user might want to filter for:

alter, analyze, associate, audit, backup, call, close, commit, connect, create, dbcc, delete, deny, desc, disassociate, drop, exec, execute, explain, grant, insert, intersect, join, kill, lock, minus, open, purge, recover, rename, restore, revoke, rman, rollback, rpc, select, shutdown, startup, truncate, update, union.

Additional Note: unless you are using a public kiosk where user tampering is of concern, this feature is not recommended.

**Report All Errors During Application Testing** – This option will bring immediate attention to a developer for incurred “soft” errors. It does so by displaying a message box displaying the error (only in Mobile Development Studio.) For example, each screen mapping command “SM.SendText” could timeout and fail, but the script will continue along (unless they are checking the return value for the function.) Turning this on will visually alert them of this type of “soft” error condition.

**Disable Development Studio Device Testing** allows for the project to be started on a downed production server and prevent mobile clients from connecting while an issue is being debugged. Otherwise the Application testing window would keep being interrupted by production clients.

**Bypass Vocollect User Login** allows a connecting Vocollect device to use the operator’s name and menu stored on the Talkman device without RFgen verifying that the user is in the RFgen database or assigning them a menu.

For legacy applications turn on **List Items Collection is One Based** so any code written to manipulate listbox or combobox controls remains

functional. For new applications the default for list controls is a zero-based collection. Solutions upgraded from version 4.1 or earlier will have this turned on automatically.

For legacy applications turn on **Unit of Measure Defaults to Characters** so any code that refers to columns or rows such as Screen.Print remains functional. For new applications everything is pixel-based unless otherwise changed by SYS.UsePixels(True/False). Solutions upgraded from version 4.1 or earlier will have this turned on automatically.

For the user preference regarding lists, **Require Double Click to Select From Lists** can be turned on or off if the users want to single click or double click list items. This will include controls like the list box and search lists as well.

A **Client Inactivity Timeout** of 30 minutes is set for network data collection devices (i.e., no activity at the device for 30 minutes will cause the device to be logged off). This setting may be modified as desired.

A **Session Shutdown Delay** of 120 seconds waits an additional 2 minutes after the mobile device sends the “disconnect” command, before ‘releasing’ the session. Sometimes a mobile device will terminate a session and reboot, but the user’s intention is to reconnect and keep working.

An **Escape Processing Delay** of 750 milliseconds tells RFgen to wait for the entire ‘Escape’ character sequence for the specified number of milliseconds. This feature compensates for some systems that have a delay in sending escape character sequences.

**Scan for Pre-Amble and Post-Amble** filter entries are character strings that are automatically sent from a scanner. They ‘surround’ the scanned data. They are optional and neither is required.

Common pre-ambles include a location number, or perhaps an operator number. Common post-ambles include control characters such as a tab or perhaps a carriage return-line feed. See your scanner documentation for information concerning how to establish these entries, or how to disable them.

Pre-amble and post-amble entries entered here are used by RFgen: (1) to identify scanner input, and/or (2) to automatically strip the pre/post entries from the character sequence received from a scanner. They will also cause a VBA Application ‘OnScan’ event to trigger.

Valid values are \n for new line, \r for return, \t for tab, \# where the # is any single character, and a group of characters like HELLO. If multiple characters are used then they are looked for as string text.

The **Embedded Graphical Menus** settings control the functioning of the menu bar display that appears at the bottom or top of graphical data collection devices. Activation / Deactivation of the graphical menu bar is controlled by the Mobile Themes / Form Defaults / Menu Strip setting. Images can be added from the Image Resources for these buttons if desired.

When using this optional on-screen graphical menu bar:

Submit	means to enter the data appearing for the current prompt
Refresh	refreshes the display screen
Clear	clears the data entered for the current prompt
Exit	exits the current process (same as F4 normally does).
Search	executes the OnSearch event if one exists for the prompt

#### The **Function Key Processing**:

F1	is used to show the popup menu
F4	immediately Exits your current application or menu.
F5	executes the OnSearch event if one exists for the prompt.

These keys may be reset if desired. For example, changing F4 to F10 would make F10 the exit key for RFgen Applications and menus. When first using RFgen with remote devices, try using the established default settings before making function key changes.

Run your mouse arrow over the function keys to see their function displayed. Screen tips will appear so that you won't have to memorize their usage.

Note that an F4 entered anywhere in the application will return you to your user menu. Selecting Logout or F4 on your menu will return you to your previous menu or the Login screen.

Additionally, you can enter the name of a function or procedure that exists in the application's VBA Win32 or RFgen BAS modules. Choosing a Default Action or Command really intends for you to type in your own subroutine name. Pressing the assigned F key or Windows menu bar button will execute that code.

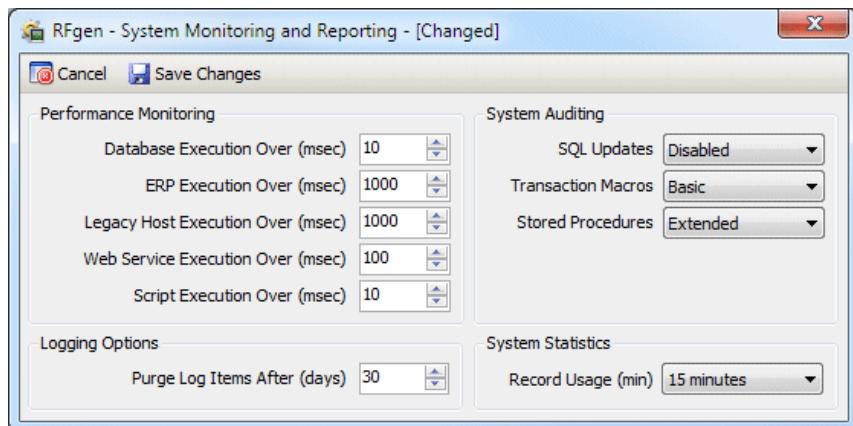
RFgen graphical mode includes 'sculpted' display screens and allows users to use graphical objects such as 'Images' (pictures), 'Buttons' or

Signature Capture prompts as part of their RFgen applications. Graphical mode displays must use Windows-based devices to display properly.

### **System Environment Properties**

Entering a name and value in this box is like creating a global constant with a read-only value. For example, if this installation was designed for multiple warehouses but this particular installation was for a warehouse called 'Main Street' then entering the property name of 'Warehouse' and a value of 'Main' would allow the programmer to identify which warehouse was being used. The command used is System.EnvironmentProperty.

## **Configuring the Performance Monitoring**



### **Performance Monitoring**

If specific connectors are taking too long to process a request, these properties can be configured to capture processing requests that take over a certain amount of time. Set the property to zero to disable. Some connections usually take longer than others. For example the database execution time will usually be significantly faster than a screen mapping connection or especially a Web Service connection. Setting all the properties to the same number would not be appropriate.

### **Logging Options**

Purging logged items will allow the database to not get too big. Turning this feature off is not recommended.

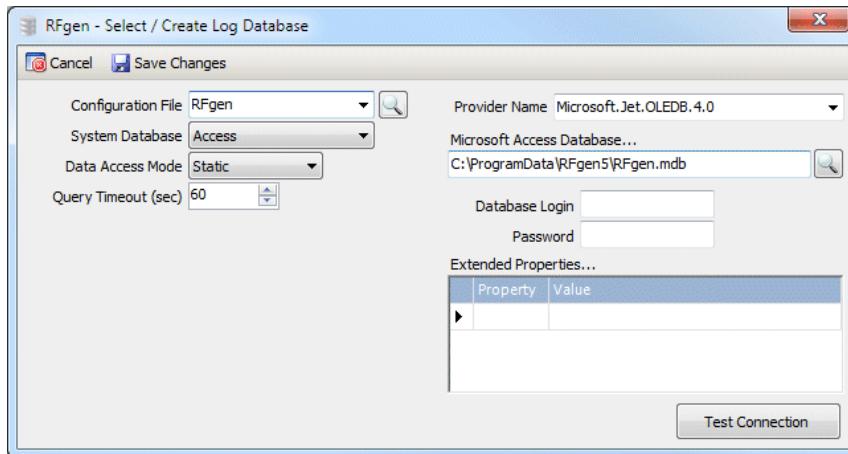
### **System Auditing**

SQL updates, transaction macros and stored procedures are all methods for updating the backend system and therefore can be logged if strict compliance to regulatory law is required. There are three modes, Disabled, Basic and Extended. This simply refers to the level of detail provided in the log.

### System Statistics

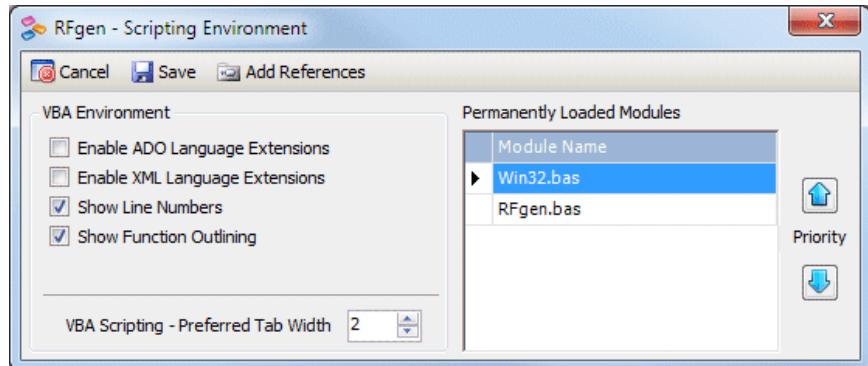
This will enable the Application Statistics under the Reports menu option to generate graphs of data connection information.

## Configuring the Event Logs



In the same way that the Application Database configuration window uses a **Configuration File** to pre-fill all the other settings, the Event Log configuration uses the same format. Specify any database and the event log tables will be created automatically.

# Configuring the Scripting Environment



**Enable ADO Language Extensions** allow you to access database(s) directly in VB rather than just through the pre-built RFgen programming extensions available for database access. If you are planning to write your own database access code, you will need to check the ADO option. Support for the method will automatically be loaded as required.

**Enable XML Language Extensions** provides additional parameters for manually specifying XML communication settings.

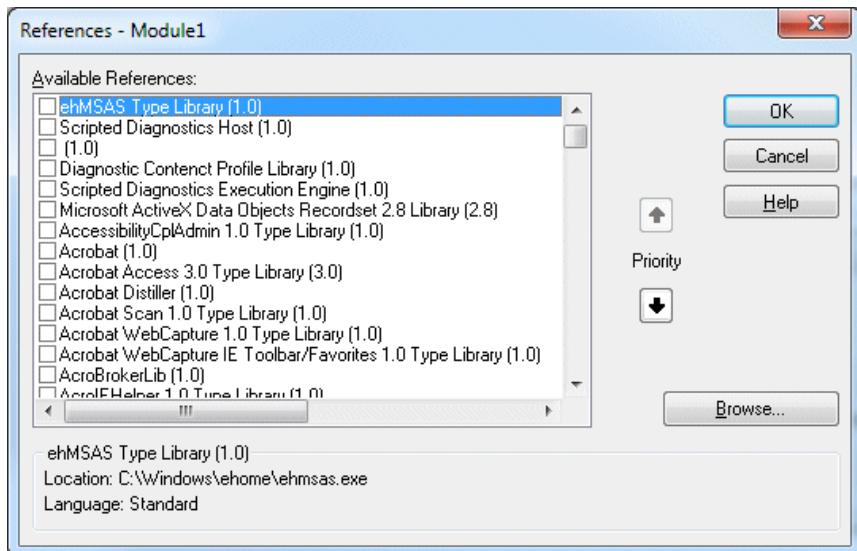
**Show Line Numbers** and **Show Function Outlining** features are meant to enhance the programming environment to provide collapsible functions, where code has changed, line numbers, etc.

The **VBA Scripting – Preferred Tab Width** setting (default is 2) is the number of spaces indented by use of the Tab key when developing VBA programs.

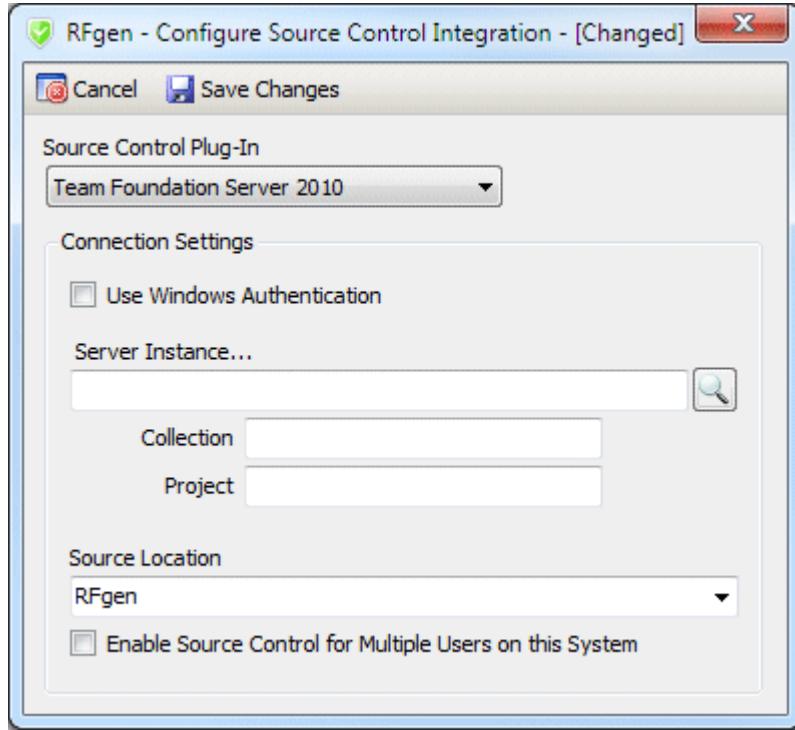
## Permanently Loaded Modules

The Win32 and RFgen BAS files are always loaded for each transaction. If another BAS file is created and the programmer does not want to place the code into either of these pre-loaded modules then it may be added to this list. The Win32.bas is typically used to store global variables. The RFgen.bas is typically used to contain functions and procedures that need to be accessible from any transaction. If a BAS file needs to be referenced for only a few or one transaction, the VBA Scripts / References menu option should be used.

The Add References menu option will globally add Global Assembly Cache (GAC) classes to the RFgen solution. This is the window that appears.



# Configure Source Control Integration

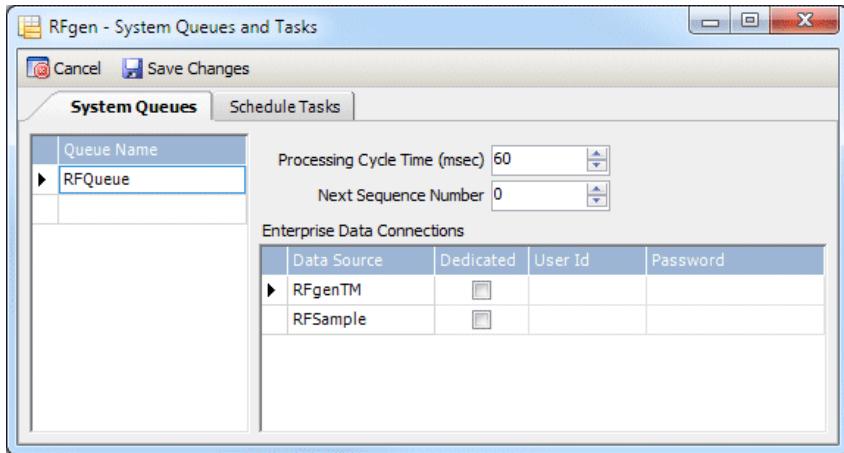


The Source Control Plug-In option contains one or more source control products that can be used to provide the development process a check-in / check-out procedure for the developed objects such as users, menus and applications. Usually the product needs to be installed on the local developer's PC to function.

The server instance and remaining properties are identical settings as those in the source control product itself.

The **Enable Source Control for Multiple Users on this System** checkbox manipulates workspace names and used folders on the PC to support Team Foundation Server with multiple people on the same system.

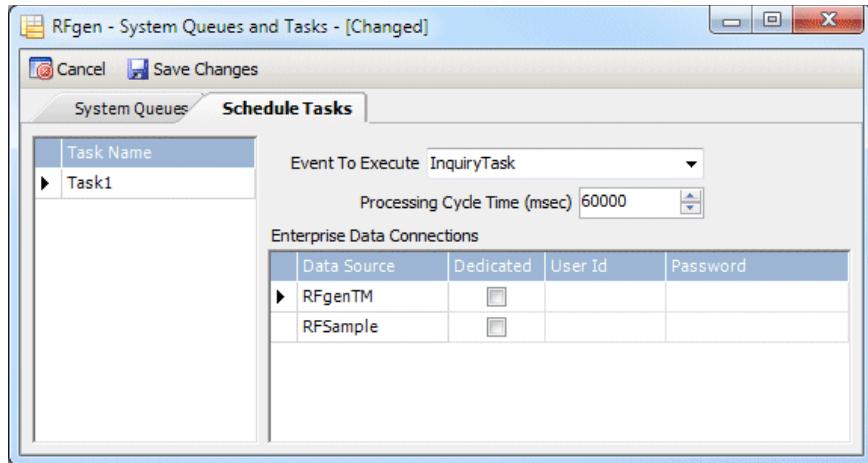
# Configuring System Queues and Tasks



This table of queues allows for several queue processes to take place at the same time. The name RFQueue is the default name for the first thread that will process transactions. The **Processing Cycle** number is the number of seconds that will pass before the system checks this queue for transactions and if one or more are found the entire queue is processed. The **Next Sequence Number** option will allow the user to change the sequence number used when queuing or making entries in the logs.

Each data source can be configured separately for each queue meaning that each queue can have either a dedicated connection to a specific data connector or it can share a limited pool of handles to that data connection. This is the **Dedicated** check box option. In either case a user and password could be specified for the queue to use when it communicates with that connection.

Note: each queue process uses one client license because it is, in essence, an automated user performing tasks against the server and the backend systems. For example, a 10 client licensed system with three separate queue processes will only allow up to seven concurrent devices.



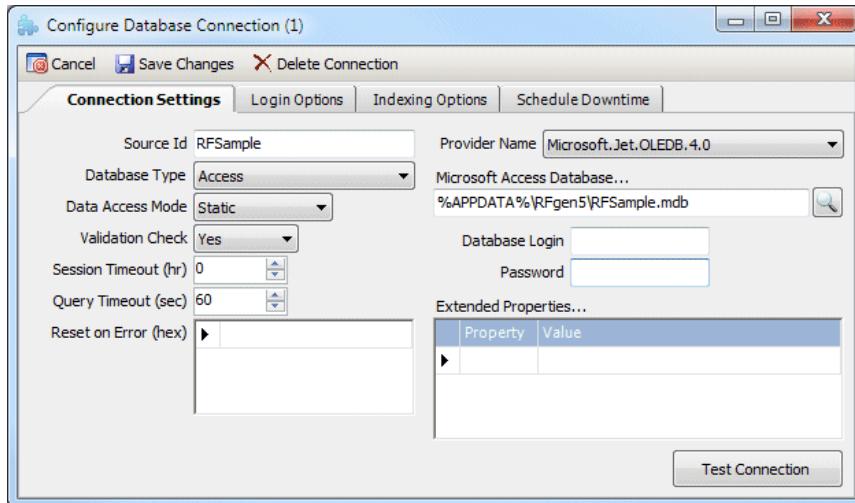
The **Schedule Tasks** tab allows the user to specify a task name and then assign it to a Timed Event macro chosen from the **Event to Execute** drop down. The **Processing Cycle Time** is in milliseconds and determines how often the server will run the Timed Event macro. Just as with the queues each data source can be configured separately as well as taking a client license. See the above section for more details.

## Configuring Database Connections

An unlimited number of data connections can be added to the solution. From the Mobile Development Studio, click on the Enterprise Connections menu bar selection.



Click on Add New Database Connection (or on an existing Connection) to establish (or modify) communications settings for the database.



Upon installation, if you have chosen to load the sample files, the system database will be configured to a data source called RFSample.

At some point, you will want to configure Mobile Development Studio to your own application database. In theory, any SQL compliant database is usable. When the server connects to a database, it will display a connection indicator at the bottom of the Mobile Development Studio window. If a red circle appears in the indicator, a valid connection has not been made. To troubleshoot an invalid database connection, click on the Mobile Development Studio Reports menu bar selection, then Event Logs to see if a message has been generated. Most likely, a problem was encountered with your Data Source entry.

The **Source Id** is the name of data connection. Spaces and extended characters are not recommended for this field.

The **Database Type** drop down field selects which type of database is to be used to host the solution objects. Changing this value changes the window to show database specific configuration fields.

The **Data Access Mode** sets the cursor to either Static or Dynamic when retrieving data from the database. Usually, Static is best because it is fast and safe. However, if you have a database like Pervasive that will actually make a copy of the data from the database system to the RFgen system when using a static cursor, you can change this option to Dynamic, so performance will not suffer. Internally, this sets the cursor option to either adoOpenStatic or adoOpenDynamic.

The **Validation Check** option will enable or disable the system's ping to the database (using the statement "Select count(\*) from RFgen\_CC") determining if there is a good connection. If this statement fails, but the server thinks the connection is good, it will create the table so that it never fails again. It is not recommended that you turn this off, unless it is specifically causing a problem.

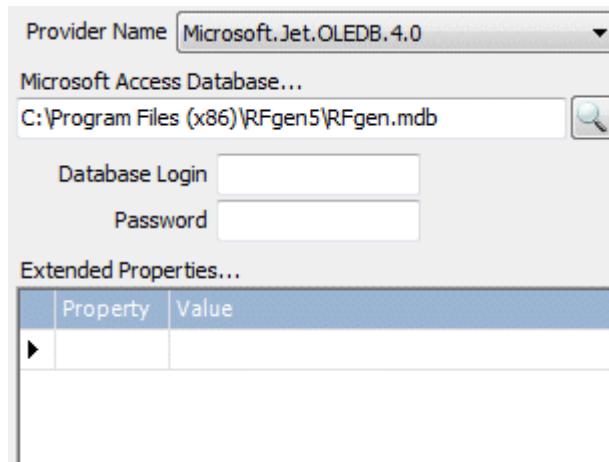
The **Session Timeout** value (in hours) will disconnect and reconnect to the database at the specified interval. This may be required if the database is configured to not allow a connection that never times out.

The **Query Timeout** specifies how long the server should wait before giving up on the database driver to come back with a response.

**Reset on Error** is a list of hex values that if returned by the ODBC driver will cause a reset of the connection. The process for adding a value is to first get the error number from the error log.

Example: the error log shows -21456327. Use the Windows calculator in Programmer mode, select Dec and Dword options, enter the number and if you need the negative sign use the ± button to change its sign. Then click the Hex option. You should get: FEB89A39. Enter this value into the box with a "0x" prefix like: 0xFEB89A39

The **Provider Name** selection will depend on the type of database you want to use. Note that these providers are not necessarily installed. All provider options must already exist on the server to be used.



For an Access database select the appropriate Provider Name for the type of system (32 bit or 64 bit).

The path, login, password and extended properties are then used to make the connection. In the case of Access most of these fields are not necessary.

This screenshot shows the configuration dialog for a provider named 'IBMDA400'. The dialog includes fields for Host Name, Authentication (checkbox for 'Use Windows Authentication'), Database Login, Password, Default Library, CCSID Conversion, Catalog Library List (\*USRLIBL), Initial Catalog (\*SYSBAS), SQL Package Name (QGPL), Block Size (4096), and Query Optimize Goal (2). Below these, there is a section for 'Extended Properties...' containing a table:

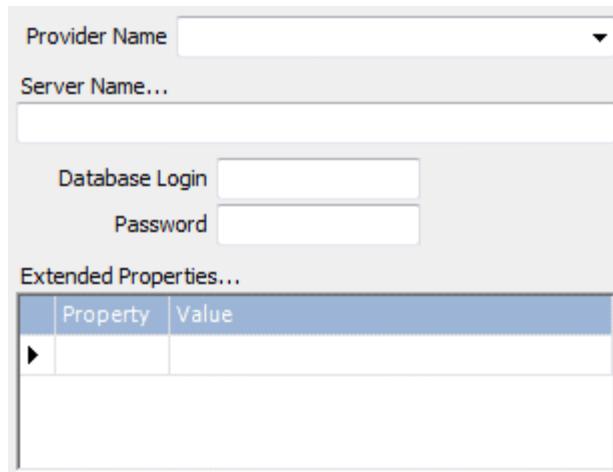
Property	Value
Block Fetch	TRUE
Convert Date Time to Char	0
SSL	DEFAULT

In the case of DB2, you can specify all the same settings that would normally go in the ODBC DSN entry for the iSeries Access driver.

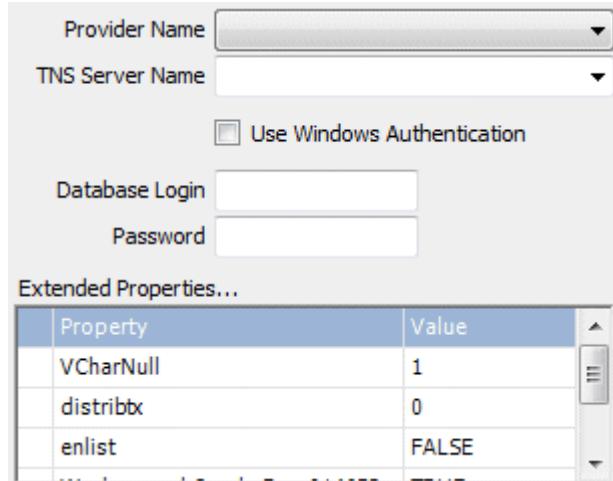
This screenshot shows the configuration dialog for a provider named 'Data Source'. It includes fields for Database Login and Password.

For ODBC DSN entries that come from Control Panel / Administrative Tools, this option assumes that the connection has already been established the server will just reference what is setup in Control Panel. This option should be used if other programs also rely on the same

database connection and it is easier to maintain the settings in one place rather than many.

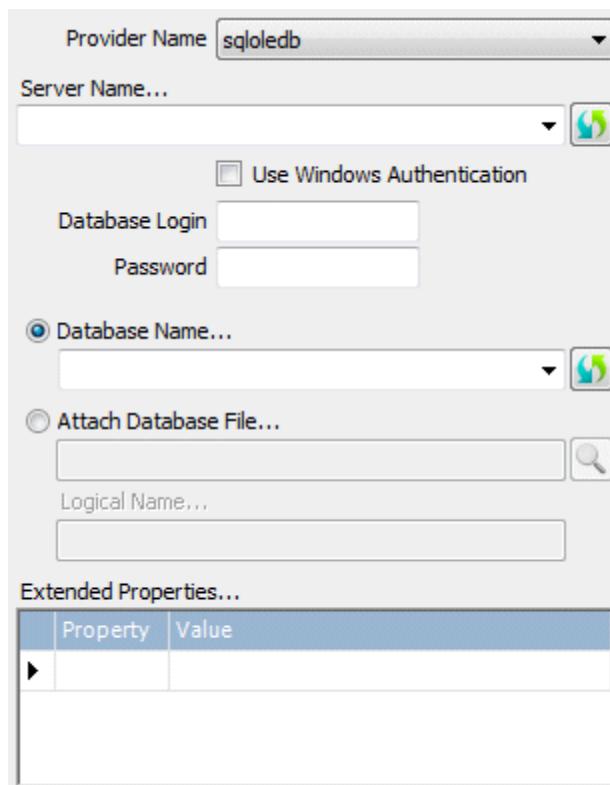


The OleDb option is the most generic method of connecting to a database. The Provider list shows many options most of which need to be manually installed (acquired by the manufacturer) before the server can take advantage of them.

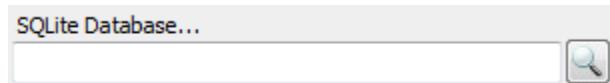


In the case of Oracle ODBC is not used but the TNS Server Name points to the Oracle server. Also specify the Provider Name and review the Extended Properties for accuracy. The Use Windows Authentication

option will take advantage of the Active Directory when connecting to the database.



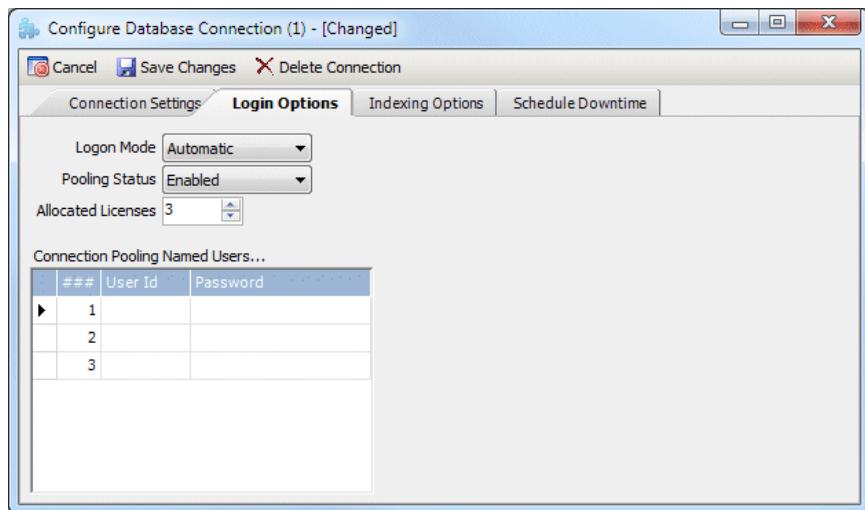
For SQL Server specify the Provider Name, Server Name and Database Name. The Use Windows Authentication option will take advantage of the Active Directory when connecting to the database. If you want to connect directly to the MDF file itself, specify the Attach Database File option and locate the database file directly. The Logical Name is typically the filename without a file extension and should not be necessary. The Extended Properties are usually not required.



For SQLite database connections just specify the DB file itself. There are no other settings. You can specify a location and name that does not exist and clicking the Test Connection button will create the database for you.

Finally click on the Test Connection button to verify connectivity. If the database has not already been setup to support the solution tables they will be created at this time. Clicking the Save Changes button will also create what is necessary but won't test the connection. Either button will also notice if the database came from an older release and ask if you want it upgraded.

The Login Options tab contains the Login Mode and Pooling options.



The **Login Mode** can be set to either Manual or Automatic where the connecting client will either have the data connector created at the start of the connection or wait until the client first needs the data connection before making the database connection.

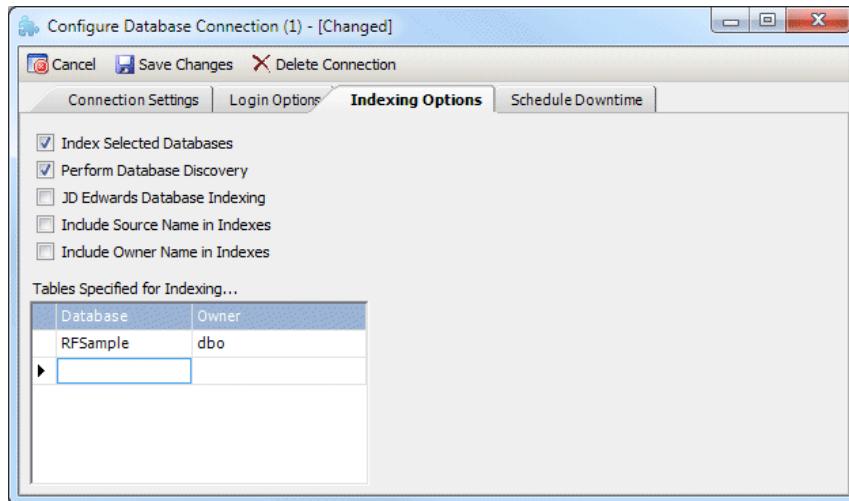
The **Pooling Status** enables the server to pool multiple remote devices to share database licenses. For example, 10 (or more) devices may typically share one database connection when pooling is enabled.

As **Allocated Licenses** are incremented, a Pool(n) session appears for each. Enter a User ID and Password for each in the corresponding boxes (if different from the defaults).

The connection pooling User ID and Password fields contained in this window are for allowing users to log in under non-default settings. As each session is taken from the pool (when simultaneous access is required) the next pool's settings will be used. For example, if your system only allowed two connections with a particular User ID, Pool(3)'s

User ID and Password could be specified and the first two will be taken from the default information.

The Indexing Options tab has advanced features for connecting to the database.



The **Index Selected Databases** checkbox means the server perform the indexing when the user saves the connection. The server will index tables within those additional databases so they can be referenced by name only. For example, F0005 is a control table in the database. Using this indexing it may be accessed simply as F0005 and the server will qualify it with database.owner.F0005 internally before SQL execution.

The Indexing grid allows the user to restrict which tables are indexed for a specific data connection. In this grid, specify the list databases and / or a list of owners of the tables that are necessary for the data collection application.

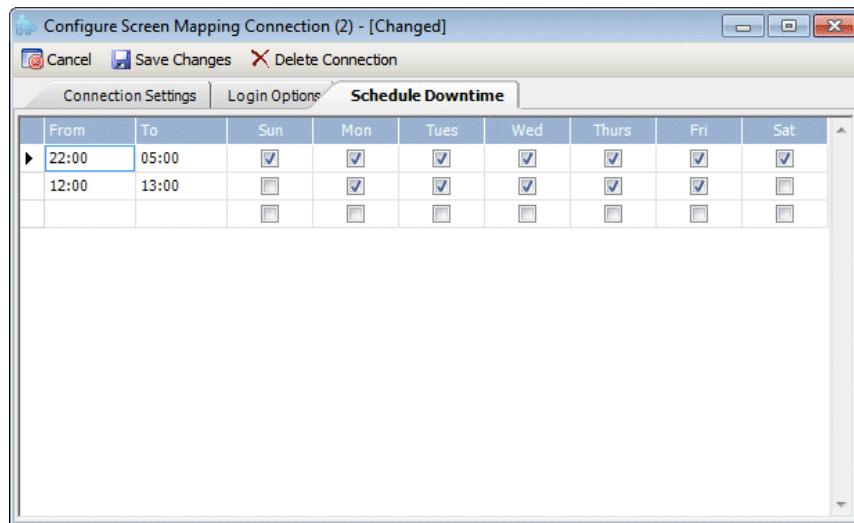
The **Perform Database Discovery** option will attempt to fill in the grid automatically with all the databases and owners ultimately indexing everything in the database.

The **JD Edwards Database Indexing** option is a JDE specific option that will query certain JDE tables to determine how the system is actively configured for the selected environment and then index specific JDE tables.

Including the **Source or Owner Name** information when downloading the tables are options to more uniquely qualify the tables or database structure in case other connected databases have tables that are named the same.

For example, your main database may be SQL Server or Oracle, but you have a need to connect to tables contained in other databases or entirely different ERP or legacy systems. Databases may be different in type, as long as they are SQL compliant.

The **Scheduled Downtime** option allows users to schedule down time for a database connection, during which the server will disable the connection, so that the database may be used without data collection interfering.



In this example the connection will be unavailable between 10PM and 5AM the next day and noon (12PM) and 1PM on the weekdays. The check boxes for each day refer to the starting time only. There is no harm in having overlapping down times. (Time is shown in a 24-hour notation.)

## Configuring ODBC Data Sources

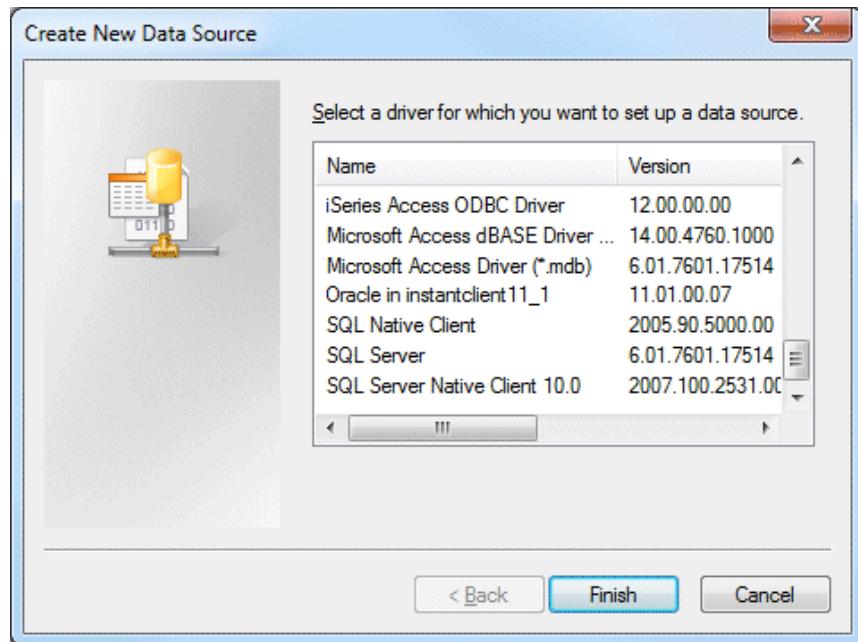
In the ODBC database connection window we noted that a 'Data Source', and an optional 'User', and 'Password' needed to be entered. This is true for each database listed.

Data Sources are normally established prior to configuring the Mobile Development Studio by means of the 'ODBC Data Sources' icon in your Control Panel 'Administrative Tools' window (i.e., click on Start, then Settings, then Control Panel, then double-click on Administrative Tools); then double-click on Data Sources (ODBC). Select the 'System DSN' tab. If on a 64-bit system and you installed a 32 bit version of the software you may need C:\Windows\SYSWOW64\odbcad32.exe.

If nothing appears in this window, click on the 'Drivers' tab. If nothing appears in the Drivers tab window, use the RFgen CD to load the Microsoft Data Access Components (MDAC) drivers. The software ships with MDAC 2.8. **If your existing MDAC is less than 2.8, please install 2.8.** If a special ODBC driver is needed for your application database and it does not appear, load it using the CD provided by the database manufacturer.

The DSN tabs provide you with a list of established User data sources. Clicking on the 'User DSN', 'System DSN' or 'File DSN' tabs may show more data sources. If a data source name does not appear for your application database, you should add a data source at this time.

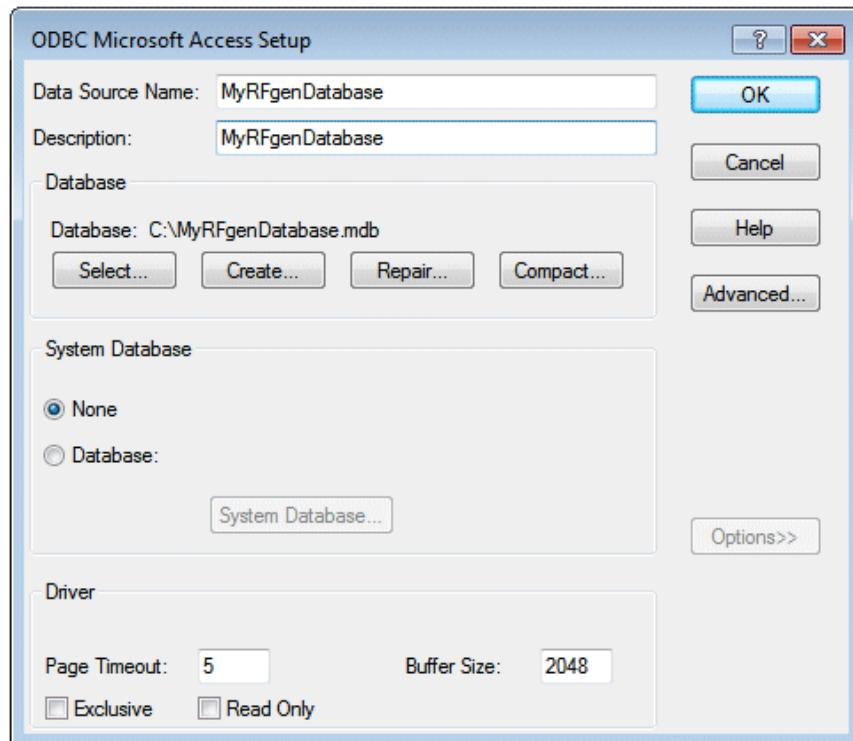
When you click 'Add' the 'driver selection' window will appear.



Here, select the driver used with your application database and click 'Finish.' If an ODBC driver for your database does not appear, you will have to load it from the CD provided by your database manufacturer. Select the ODBC driver to use and click 'Finish'.

The data source window that appears will depend on the ODBC driver being used. Shown below is a Microsoft Access database data source named 'MyRFgenDatabase' located in the 'C:' folder.

Here, we simply filled in the Name and Description boxes and clicked on the 'Select' box to browse for the Access database file 'MyRFgenDatabase.mdb'. You can see the path to the database directly above the 'Select' box.



Click OK to save your data source.

The sample RFgen data connection originally configured (Data Source Name: 'RFSample') contains the tables used to illustrate Mobile Development Studio and is similar to the above (except that the RFSample.mdb Access database is located in the RFgen folder, which

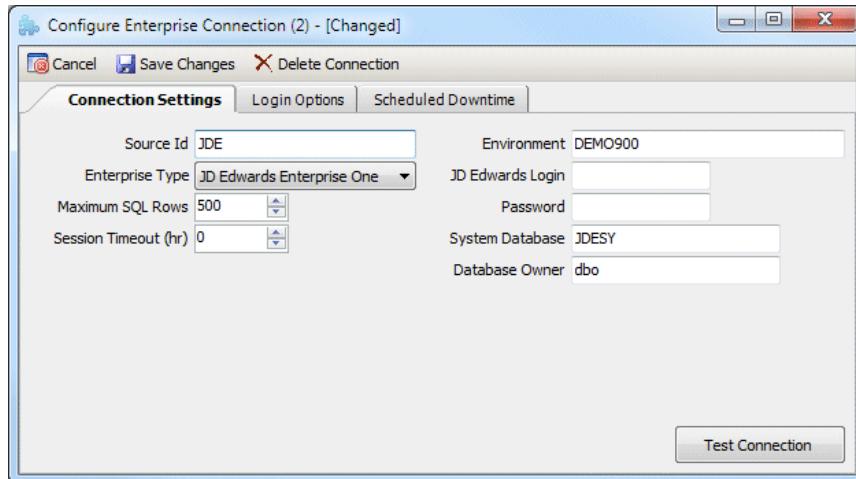
was created when you loaded the Mobile Development Studio Software).

## Configuring a Screen Mapping Connection

For screen mapping related information, please see the RFgen Screen Mapping section.

## Configuring an ERP Connection

To configure an ERP connection, select “Add New ERP System Connection”, the following configuration window will appear.



First, give the connection a **Source ID** name and then choose the **Enterprise Type** to JD Edwards Enterprise One.

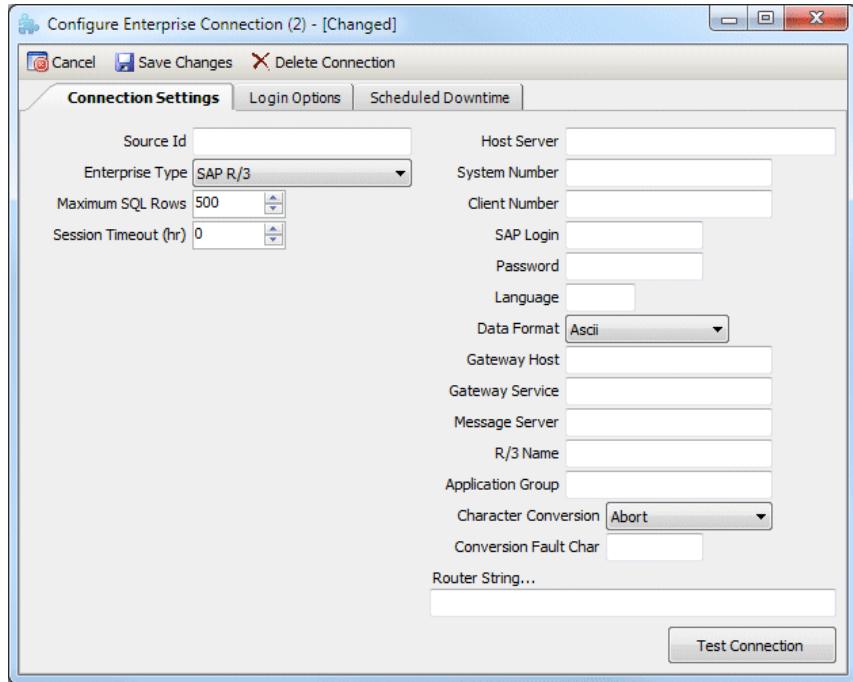
The **Max SQL Rows** prevents a ‘lock-up’ scenario in case a query was bringing back too many records by accident. ERP systems can typically have millions of records and this could prevent a frozen client.

The **Session Timeout** value (in hours) will disconnect and reconnect to the ERP at the specified interval. This may be required if the ERP is configured to not allow a connection that never times out.

The **Logon Timeout** (seconds) will return with a failure in the error log if a request to log on to JD Edwards never comes back.

In the case of JD Edwards, first enter the **Environment**. The **JD Edwards Logon** and **Password** are typically required and are entered next. Since this ODBC connection may have many environment databases, use the **System Database** field to specify the proper database. Use the **Database Owner** option to specify the owner of the tables in this database.

The Logon Options and Scheduled Downtime options work exactly as described in the database connector section.



First, give the connection a **Source ID** name and then change the **Enterprise Type** to SAP R/3.

The **Max SQL Rows** prevents a 'lock-up' scenario in case a query was bringing back too many records by accident. ERP systems can typically have millions of records and this could prevent a frozen client. The command `ERP.ReadData` can perform SQL statements against the SAP connector.

The **Session Timeout** value (in hours) will disconnect and reconnect to the ERP at the specified interval. This may be required if the ERP is configured to not allow a connection that never times out.

The **Logon Timeout** (seconds) will return with a failure in the error log if a request to log on to SAP never comes back.

For SAP, the **Host Server** is the application server. Enter the **System Number**, **Client Number**, **SAP Logon**, **Password** and **Language** with the same data as stored in the SAP GUI Logon Pad application.

The **Unicode** option tells RFgen how to interact with the system, either by using Unicode formatted communication or non-Unicode (ASCII) formatted data.

The **Gateway Host Gateway Service**, **Message Server**, **R/3 Name**, **Application Group** and **Router String** are optional parameters. If your Logon Pad requires these settings, then add them here for RFgen.

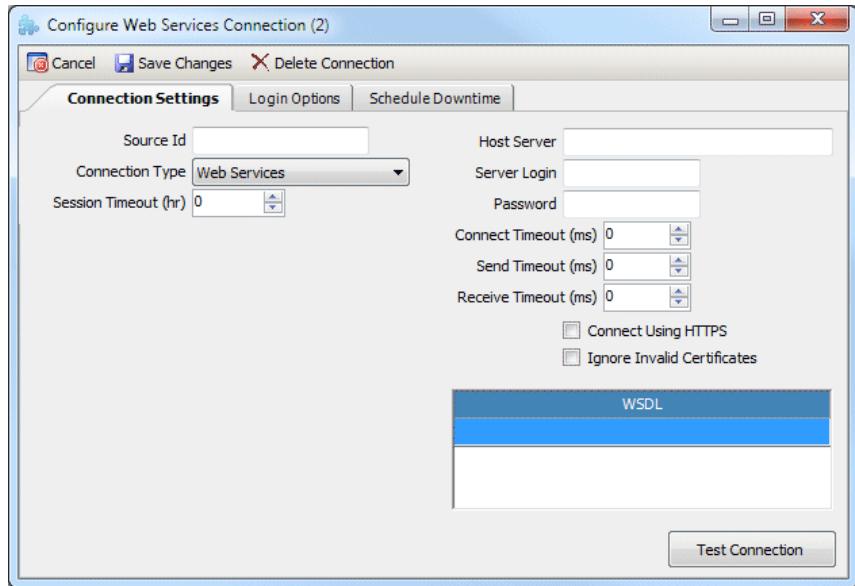
For SAP load balancing configure the following fields: System Number, Client Number, User ID, Password, Message Server, R/3 Name and Application Group. Leave the Host Server blank since the Application Group setting will distribute requests to multiple host servers. In some cases you must leave the System Number blank as well.

The **Character Conversion** and **Conversion Fault Char** properties are designed to handle problems when SAP sends data in a different code page than what RFgen is configured to display. If the text does not have a translation RFgen can be configured to abort the conversion, copy the bad character or replace the character with the character entered in the Conversion Fault Char property.

The Logon Options and Scheduled Downtime options work exactly as described in the database connector section.

## Configuring a Web Connection

To configure a Web connection, select “Add New Web Services Connection.



Enter a **Source ID** which will be the name to reference when Web object's DataSource property.

The **Session Timeout** value (in hours) will disconnect and reconnect to the ERP at the specified interval. This may be required if the ERP is configured to not allow a connection that never times out.

The **Host Server** is the IP address or DNS name of the server being used to process requests.

A **Server Login** and **Password** can be entered if required by the server.

The **Connect Timeout** is a number in milliseconds that will terminate a request for connection if this value is exceeded.

The **Send Timeout** is a number in milliseconds that will terminate a request from the client sent to the server if it has not received the HTTP request from the client.

The **Receive Timeout** is a number in milliseconds that will terminate a response from the server to the client if this value is exceeded.

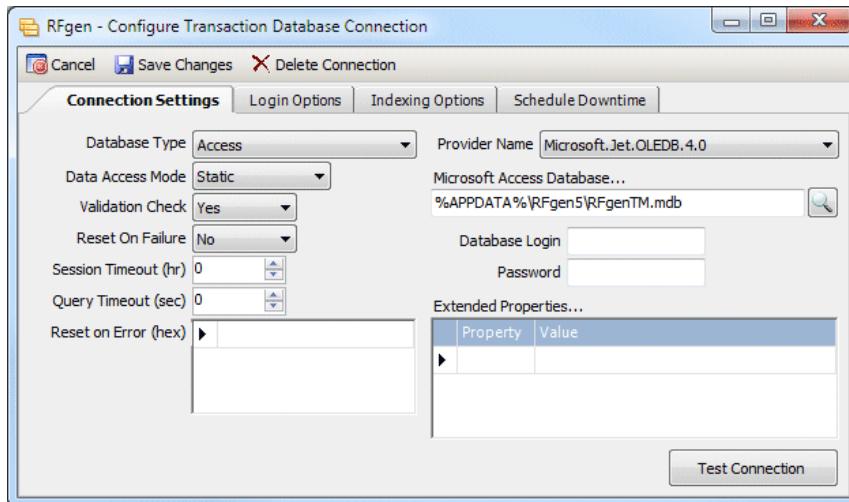
**Connect using HTTPS** is used when the web server communicates using the encrypted protocol.

The **Ignore Certificates** option is a True or False value (only for HTTPS connections) indicating that the data connector will ignore certificate errors from the server. If this is set to False, and there is an error, it will be logged in the RFgen error log and the Web object's Execute method will return a False value.

The **WSDL** grid is for future use.

The Logon Options and Scheduled Downtime options work exactly as described in the database connector section.

## Configuring Transaction Management



Configuring a Transaction Management database connection is the same as any other database connection. All the same fields apply. This connection is simply dedicated for the queuing process. See the section on configuring a database connection for details.

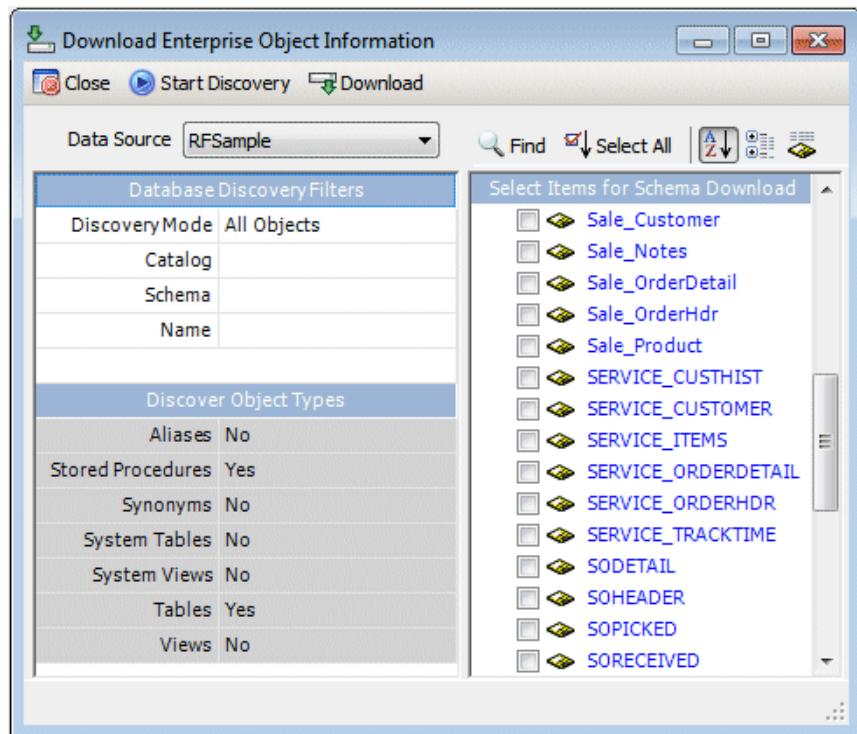
There is one property that is unique. **Reset On Failure** when turned on will reset the data connection if any of the queued transactions should fail. This should not be used unless deemed necessary.

## Download Enterprise Objects

To use table fields directly on an application screen or to take advantage of backend functions or stored procedures, the server needs to know which data connection contains the objects, which tables or

procedures are required and what the object's structure looks like in order to internally generate the proper calls and perform the reads and writes. To do so for ODBC connections:

1. Click on the Enterprise Connections menu item
2. Click on Download Enterprise Objects
3. Select the database from the Data Source drop-down
4. Click the Start Discovery menu item
5. Click the rows for tables to be downloaded (or click Select All), and then click Download.

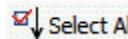


The Discovery Mode option allows for all objects to be discovered or just selected items that may be selected.

Blue items have been previously downloaded. The checked tables will be transferred into RFgen, which uses information from downloaded items when creating data entry / display applications.



If there is a long list of items, the find option can narrow the focus. Using the “Display On-File Items Only” option will limit the list to previously downloaded tables.



This selects all entries in the list.



This button selects the alphabetical list view and enables the Filter feature.



This button selects a tree view of the list of items.



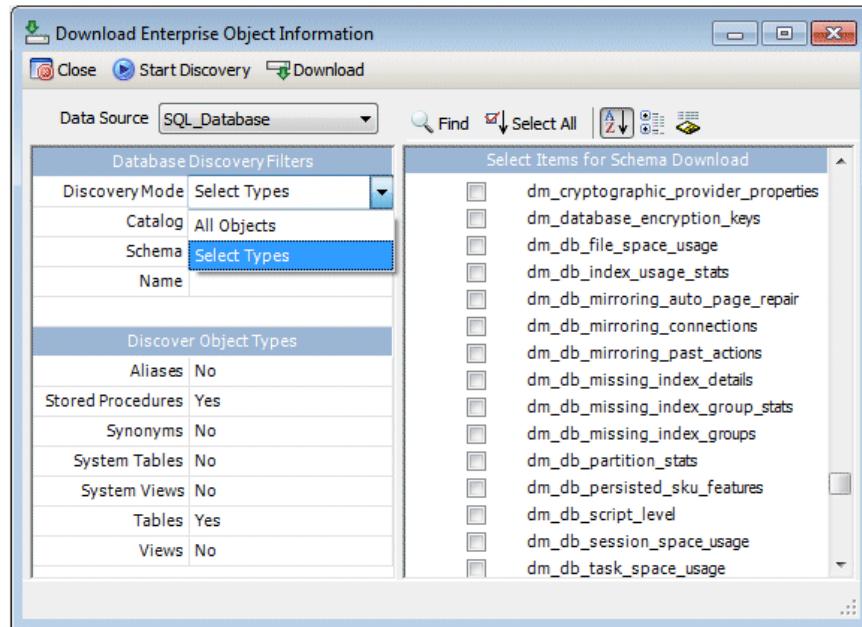
This button displays only the previously downloaded items and enables the Find feature.

Since RFgen is SQL compliant, it is important to note that database table and field names should not use any of the reserved words listed in the section describing the VBA commands.

### **Downloading Stored Procedures**

To work with stored procedures, users must first transfer (download) the desired stored procedure from the database. To do so:

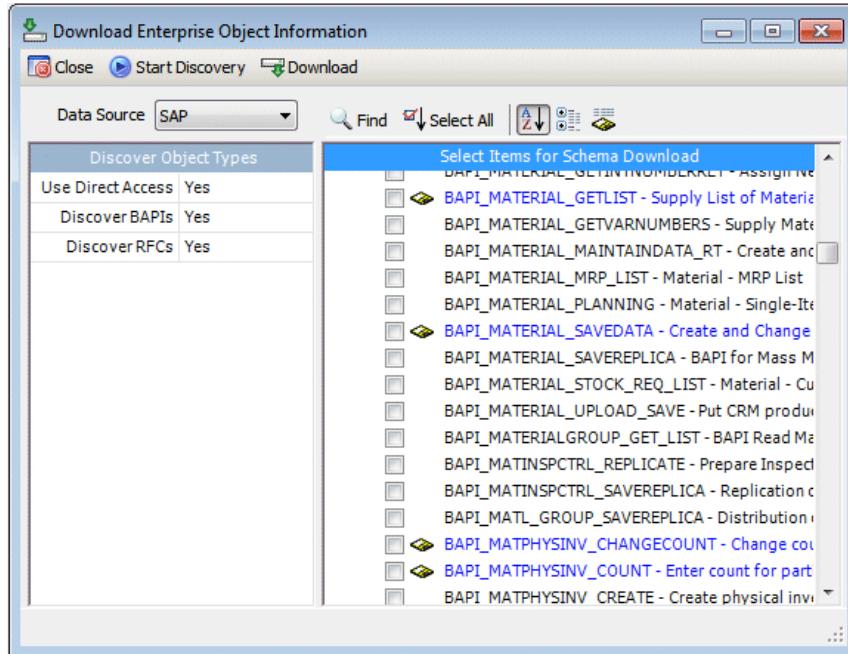
1. Click on the Enterprise Connections menu item.
2. Click on Download Enterprise Objects.
3. Select the database from the Data Source drop-down.
4. Click the Start Discovery menu option
5. Click the rows for stored procedures to be downloaded (or click Select All), and then click Download.



### **Downloading ERP Business Functions**

To work with business functions from an ERP system, users must first transfer (download) the desired business functions from the ERP system into RFgen. To do so:

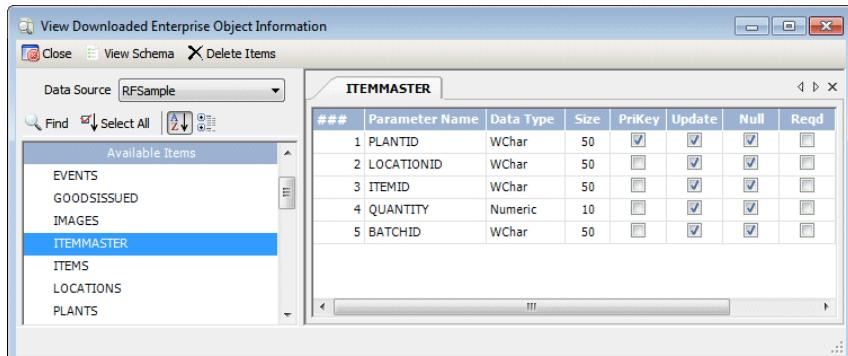
1. Click on the Enterprise Connections menu item.
2. Click on Download Enterprise Objects.
3. Select the ERP connection from the Data Source drop-down and select "Start Discovery" from the menu.
4. Click the rows of functions to be downloaded (or click Select All), and then click Download. Selecting all business functions from an ERP system will save an extremely large set of data in the RFgen application database and could take a very long time. Only download the business functions that are required by the applications.



Blue entries have been previously downloaded. The SAP Discovery Filters allow you to select, if just BAPIs are downloaded or if RFCs are as well.

## View Enterprise Objects

Previously downloaded object schemas may be viewed by clicking on the Enterprise Connections – View Enterprise Objects selection. A view window will appear.



After choosing a data source select a name and click the “View Schema” menu option or simply double-click the table name to view its parameters.

Shown above are the field definition items for the chosen table in our sample/demo Microsoft Access database (RFSample.mdb) ‘ItemMaster’ table. Each transaction table must have at least one primary key ('PartNo' as indicated above). RFgen identifies database keys simply by determining which database items are 'indexed'. If more than one item is indexed, the first item encountered will be marked as the primary key.

In general, use of Numeric, Text/String, Date, and Currency ‘Data Types’ in your database is suggested, as more esoteric data types may cause problems when trying to update your database table(s).

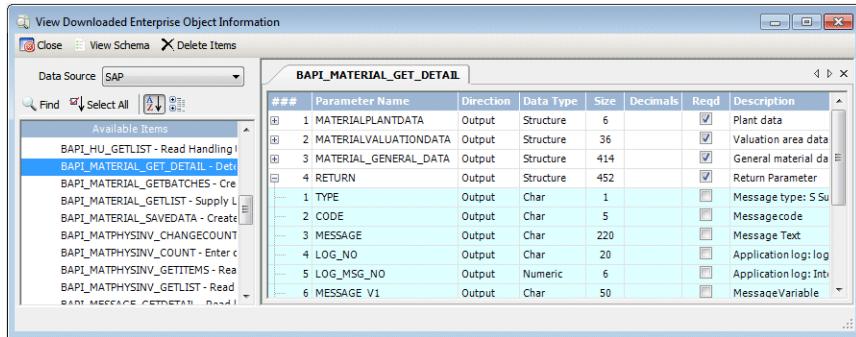
Only fields defined as updateable will be sent to the database when a transaction is completed. This is only true if RFgen is generating the SQL statements internally as opposed to user created SQL statements. Fields not defined as 'null allowed', will send a space or a 0 (zero) if no data is collected for them. Fields marked as 'primary key' are used to access the table data and may be used to retrieve selected data.

When viewing the SAP specific function properties after a download, please note that for “packed” or compressed numeric data elements RFgen displays the byte length and the allowed number of decimals instead of the actual number of characters allowed in the field.

Selecting a table entry and choosing Delete from the menu only removes the stored structure of that table from the RFgen configuration. This delete has no impact on the actual database.

### **Viewing ERP Business Functions**

Business Function definitions may be viewed by clicking on the Enterprise Connections – View Enterprise Objects selection. A view window will appear.



A list of downloaded business functions is displayed. If a description is stored in the database and was retrieved by RFgen, it can be displayed after viewing the parameters. Select a name and click the “View Schema” menu item or simply double-click the business function name to view its parameters.

# Mobile Development Studio Menu Bar

The Mobile Development Studio 'Menu Bar' displays below the 'RFgen: Mobile Development Studio' banner, as shown here.



Here click on:

- Configuration:**
- (1) Application Database configures the database storing all the solution objects
  - (2) Application Services include port configurations, load balancing, server run mode, service credentials, security and controller failover monitoring
  - (3) Desktop Preferences for fonts, default design mode, telnet settings and default locale
  - (4) Environment Properties settings including system options, timeout values, Pre/Post-amble scanner defaults, embedded graphical menu options, function key options and system properties
  - (5) Performance Monitoring sets thresholds for backend communication for debugging purposes
  - (6) Scripting Environment settings like ADO and XML extensions and globally loaded BAS files
  - (7) Source Control Options allow developers to use a third party source control product if its plug-in is supported.
  - (8) System Queues and Tasks configure all queues and timed event macro execution

## Enterprise Connections:

- (1) Specify the types and locations of additional SQL compliant database connections, ERP connections, screen mapping hosts systems and Web service hosts
- (2) The list of configured data connections

- (3) Transaction Management connection setup, queue setup, timed event setup, scheduled downtime and debug tracing options
  - (4) Download Enterprise Objects like tables, business functions or stored procedures
  - (5) View Enterprise objects like tables, business functions or stored procedures
- Testing:**
- (1) Application Testing to debug applications
  - (2) Transaction Testing to debug transaction macros
  - (3) SQL Query Testing to open an SQL window, to view local database tables and data
- Devices:**
- (1) Deploy Mobile Applications can place a small client (MDM) on the device so RFgen can make changes and updates automatically and full profiles
  - (2) Remote Application Explorer will show the users, menus, applications, etc. installed on the device
  - (3) Remote Database Explorer shows the data on the mobile device's RFgen database
  - (4) Remote SQL Explorer will let the user read and write directly to the user database on the device
  - (5) Remote Log Viewer will show the error log file on the attached device
- Utilities:**
- (1) Export Mobile Applications exports RFgen objects (applications, menus, users, VBA code; see the next section)
  - (2) Import Mobile Applications will import RFgen objects
  - (3) Find in Application Scripts creates a list of all applications, modules and macros that contain what was searched for
  - (4) Replace in Application Scripts will perform the find and replace in all applications, modules and macros for the criteria given
  - (5) Undo Save/Delete Action
- Reports:**
- (1) Application Statistics
  - (2) Event Logs
  - (3) Performance Logs show flagged events that exceed processing time thresholds
  - (4) Script Validation will syntax check all application scripts, VBA modules and transaction macros and display which ones have errors

- (5) Task List displays all TODO comments in the code
- (6) Upgrade Log shows any upgrade specific messages
- (7) View Transaction Queues

- Help:**
- (1) Mobile Development Manual displays the RFgen manual
  - (2) VBA Scripting COM Syntax displays VBA COM language options
  - (3) VBA Scripting .NET Syntax displays VBA .Net language options
  - (4) Obtaining Technical Support has links and e-mail addresses
  - (5) About information includes installed options and the version

## Testing

The testing menu option contains the different places developed objects can be exercised. The testing of applications, menus, users, global programming code, data connectors, mobile devices and some other items are all tested using the Application Testing option.

For testing transaction macros, Timed Events and queuing choose the Transaction Testing option.

To test specific SQL statements, interact with table data or test the data connectors themselves choose the SQL Query Testing option.

### Application Testing

The Application Testing menu option is used to test applications in a local location or in a Remote location. Local refers to thin clients connecting to the Studio and running the applications on the server side whereas the Remote location option runs the code on the batch device directly and lets the developer debug the code remotely.

When running locally, there are some **Client Type** options; Mobile Client, Telnet Client, Vocollect Device and Service Requests. Mobile Client refers to the supplied graphical client and the Telnet Client option paints the screens in a DOS style character-based display.

The Vocollect mode displays the data stream from the Vocollect device as it communicates with the server.

Service Requests displays the underlying data stream from such processes as XML or Winsock connections.



The **Device Size** option allows the user to change the emulator's size to test different shaped devices such as forklift displays, tablets or PCs.

The **Theme** selection defaults to the choice in the Configuration / Desktop Preferences but can be changed here to see how alternative themes will look for the displayed form.

The **Language** option is for changing the system locale in case multi-lingual forms are based on the device's locale. This provides a way of changing or emulating a device's locale/

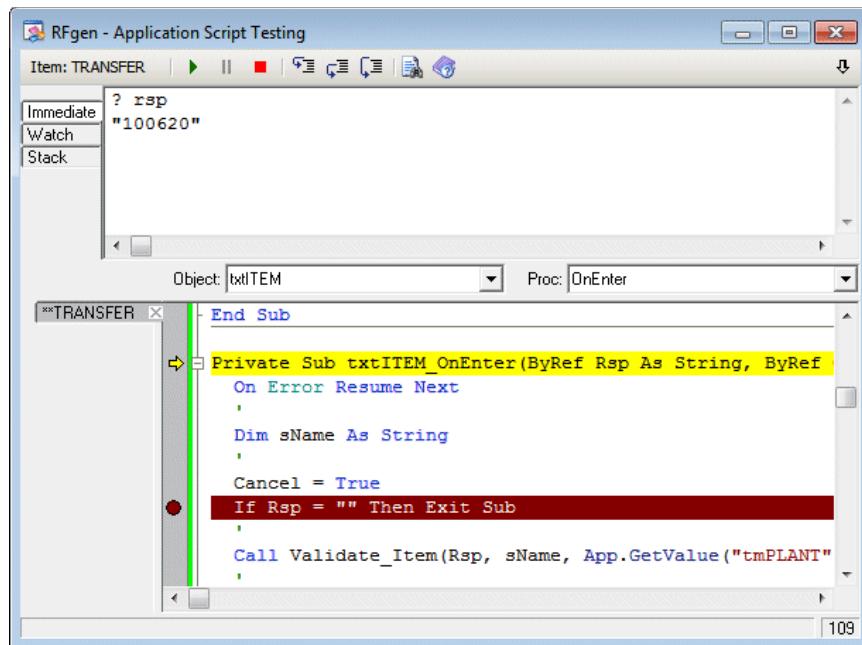
The **RTL Display** option is for displaying all elements on the form in a right-to-left format. Languages such as Arabic and Hebrew are examples.

When the testing window first appears, the display screen will be blank. To start the testing process, click on the 'Start Test' menu selection. This will allow you to Login via one of the user accounts that have been

established, e.g. for the sample applications the user is 'SAM', with no password. The Environment Values display (when the pushpin icon is toggled) the user logged in, the application or menu currently presented and its description and any options that will be passed in to an application from a menu that contains passing parameters. The Debug Script menu option displays a code window used to debug applications.

The Client Type menu option is a drop-down option. Select the proper mode and then click the Start Test menu option to begin testing.

The Debug Script menu option displays a window containing the Immediate, Watch, Stack and code windows. These windows are similar to the programming environment of Microsoft VB and only the differences will be discussed here. The Fkeys assigned to the main buttons are F5 = Play (green arrow), F8 = Step (first step button), Alt F8 = Step Out (second step button) and Shift F8 = Step Over (third step button).



The most effective way to debug code is to click on the blue pause button and then trigger an event. Code execution will be paused on the first line of that event. To debug code in the Form Load event, add the line 'Stop' so that execution is halted on that line.

After you have entered your user ID and password, the menu assigned for the user will display. Use your keyboard/keypad arrow keys to select an application and press <Enter>. In a graphical mode you may use the mouse to select and execute menu items. In test mode, you may enter data exactly as if you were using a remote device.

The Application Values grid on the Watch tab shows what your data item (record) looks like as data is added.

Click on 'Stop Test' to stop the testing process.

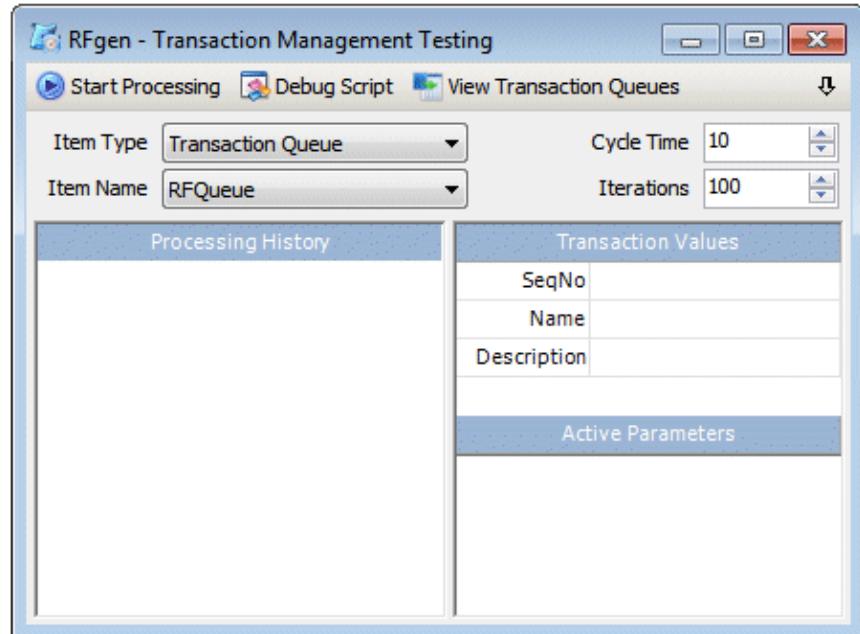
### **Network Application Testing**

For an actual RF network test of your RFgen applications, you'll first need to install and configure your remote devices with either a Telnet client for character-based displays or the Graphical Client (which is available on the RFgen CD).

Mobile Development Studio, by itself, allows one usable network device to log in, for test purposes, without requiring a software license. Multiple device tests require using the Server that can allow over 1000 concurrent users for a period of two hours until the service is stopped and restarted. An RFgen Software license is required to permanently activate your RFgen network (by means of the RFgen 'Mobile Enterprise Service Management' icon that then appears on your Windows system tray).

### **Transaction Testing**

This option will let the user test Timed Event macros and queue processing.

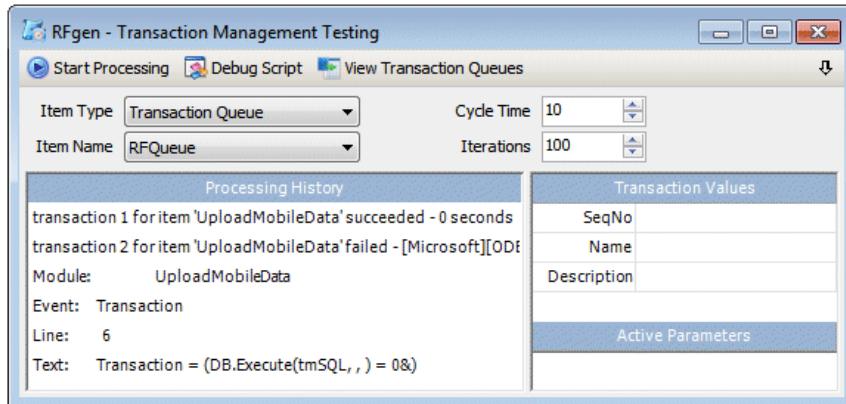


The **Item Type** option switches between queue processing and timed Events. If Transaction Queue is selected, the **Item Name** specifies which queue is to be processed. If Transaction Event is selected, then the Item Name list is populated with the configured events under the Transaction Management / Processing Events option.

The **Cycle Time** is an interval in seconds that is how often the queue will be checked for new transactions. For events this is how often RFgen waits between each execution of the event.

RFgen will continue testing for a total number of times specified in the **Iterations** box.

Select the item to be tested from the drop-down options. In the case of Transaction queues the user should have already queued what they need tested. The window pane on the right will display the parameters of the item being tested.



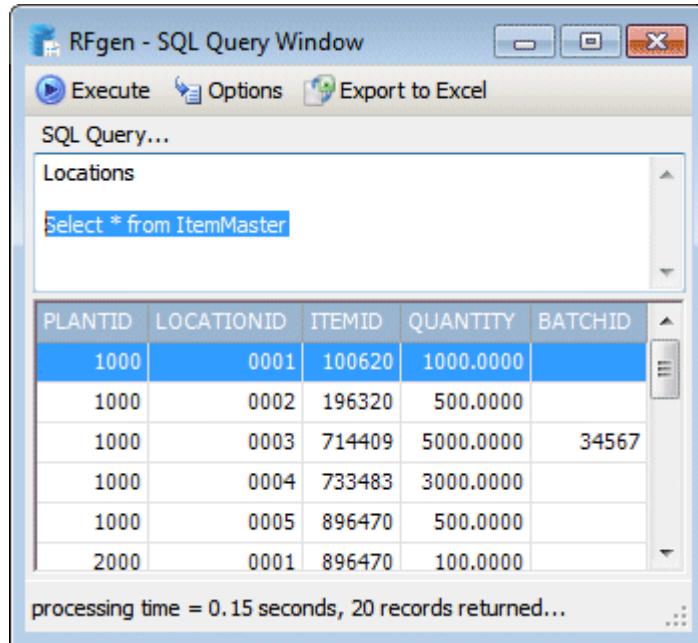
As testing is taking place the window will show a status of the processing. If a transaction fails, the error message is displayed. If it is successful, the amount of time required to process that transaction is displayed with a success message.

The Debug Script menu option allows the user to stop, debug and test the scripting of the Transaction or Timed Event macros as they are executed.

## SQL Query Testing

RFgen provides a utility that lets the user test SQL statements to see the results before executing them in the code. This utility can also be used to undo updates, check results, delete values or even adding and dropping tables. Any SQL command entered here is submitted to the ODBC driver for execution. There are no limitations by RFgen as to what can be submitted.

If the intent is “select \* from TableName”, then only specifying the table name will default to the “select \* from” when executed.



The multi-line text box allows the entry of several SQL statements. In this case, highlight the intended SQL statement and click Execute from the menu.

Multiple SQL statements must be separated by semi-colons ":"

Select \* from items;

Select \* from itemmaster;

These will be considered two different SQL statements. If no text is highlighted, then it will look at the current insertion point to determine which SQL statement to execute based upon semi-colon delimiters. Further, if you click on Options menu / Display Query History – then double-click on an item, it will append it to the SQL window instead of replacing the existing contents.

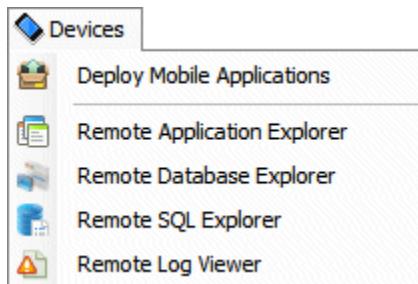
The SQL Query Window gives a current snapshot of the data in your database. As transactions are applied against your data, you will need to re-execute your SQL statement.

The Options menu will allow the user to select from any of the previously executed SQL commands and to also limit the output to a maximum number of records. The default is 1,000 records.

The Export to Excel menu option will prompt for a location to save an Excel file. Microsoft Excel does not need to be installed on the system for this function to work.

## Devices

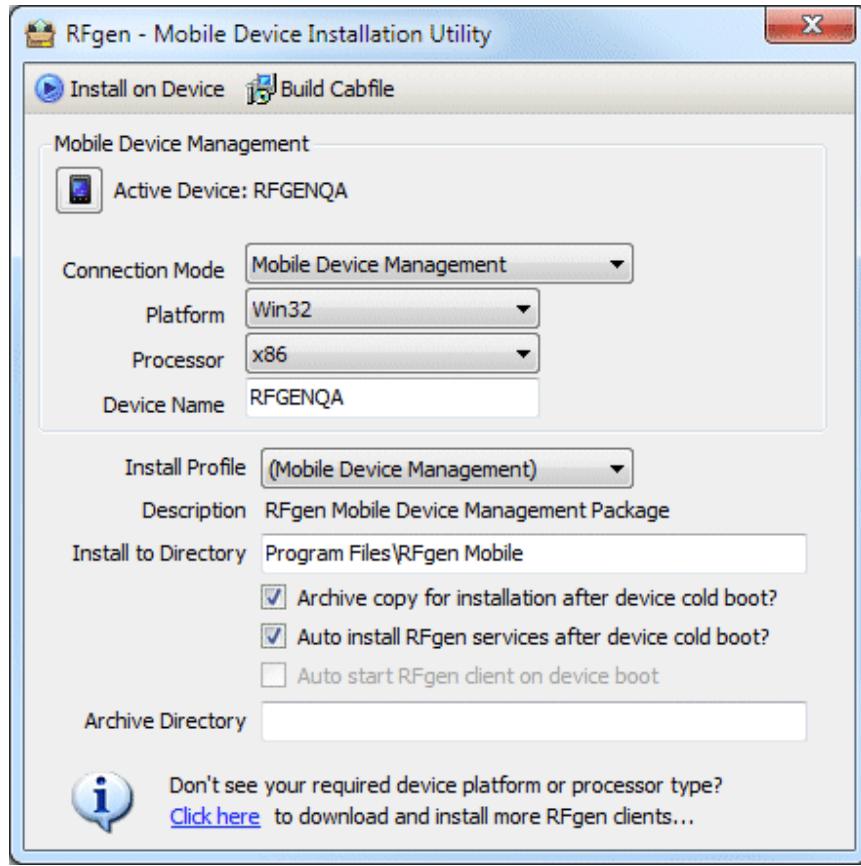
The Devices menu contains the applications to control and deploy files to mobile devices. The mobile device must contain the “listener” application called Mobile Device Management. All communication and interaction with the mobile device go through this application.



### Deploy Mobile Applications

The Mobile Device Management application is used to provide access to the file system of the device. The development platform can then update Applications, Menus, Users, etc. as part of the data collection need and also update the underlying client files in cases where the server is upgraded and new client files need to be distributed. Full profiles can also be sent to the device.

From the menu choose Devices / Deploy Mobile Applications.



For this initial install, the mobile device needs to be connected to the Mobile Development Studio through Microsoft ActiveSync and the Mobile Client files must be installed on the machine running the Mobile Development Studio.

If the device can be detected, the settings will pre-populate automatically.

The **Active Device** is informational only.

The **Connection Mode** tells the system how to connect to the device if at all. No Device means that a CAB file will be created but not placed on the device. The user will need to move that CAB file to the device at a later time or across the network. ActiveSync means that the device is connected to the PC using Microsoft's ActiveSync program. Mobile Device Management means that the CAB will be moved to the device

over the wireless network using the MDM listener already installed on the device.

The mandatory parameters are the **Platform** and the **Processor**. These combine to tell which files must be installed on the device. Usually in the settings on the mobile device, there is an About program that will show these values. If the processor says some variation of "X-scale", it is referring to the ARM processor type. It is recommended to use the latest ARM driver in the list first and only choose others if there are compatibility issues.

The **Device Name** is equivalent to the PC name as seen by the network.

The **Install Profile** option allows the user to select between the MDM deployment or any saved device profiles.

The **Installation Directory** is where all files will be placed.

The **Archive Copy** option places a CAB file on the storage card that can be run again as needed to install the software, usually upon a cold boot.

The **Auto Install** option places a CAB file in a specific folder that is used by the operating system to automatically install any CAB files placed in that folder. This is dependent on the operating system having or supporting this concept.

The **Auto start** option will launch the client whenever the device is warm-booted. A cold-boot may require the software be reinstalled first as set by one of the above options.

The **Archive Directory** usually is a place on the storage card where a backup of the CAB file resides. Check the Archive Copy check box to use this field.

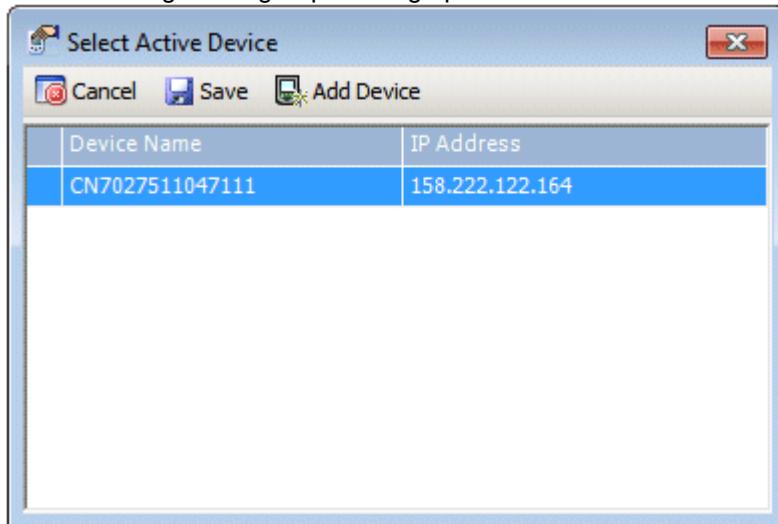
Click the **Install on Device** option to download and install the MDM application or profile. Once this is done, the mobile device can be disconnected from the server. Click the **Build Cabfile** option to create an installation CAB file that will be manually deployed to the mobile device at a later date.

The link at the bottom of the window will take the user to a web page where additional files may be downloaded, if various files for devices were not installed.

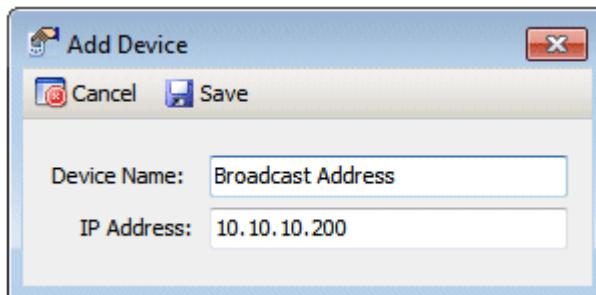
## Remote Application Explorer

This screen is used to send updated or additional components to a mobile installation on the device. Thin client solutions do not have users, menu, applications, etc.

Click on the Remote Application Explorer menu option and then select the active device that requires an update. (See the Server.SyncApps command to update multiple devices.) Click the button in the Mobile Device Management group to bring up this window.

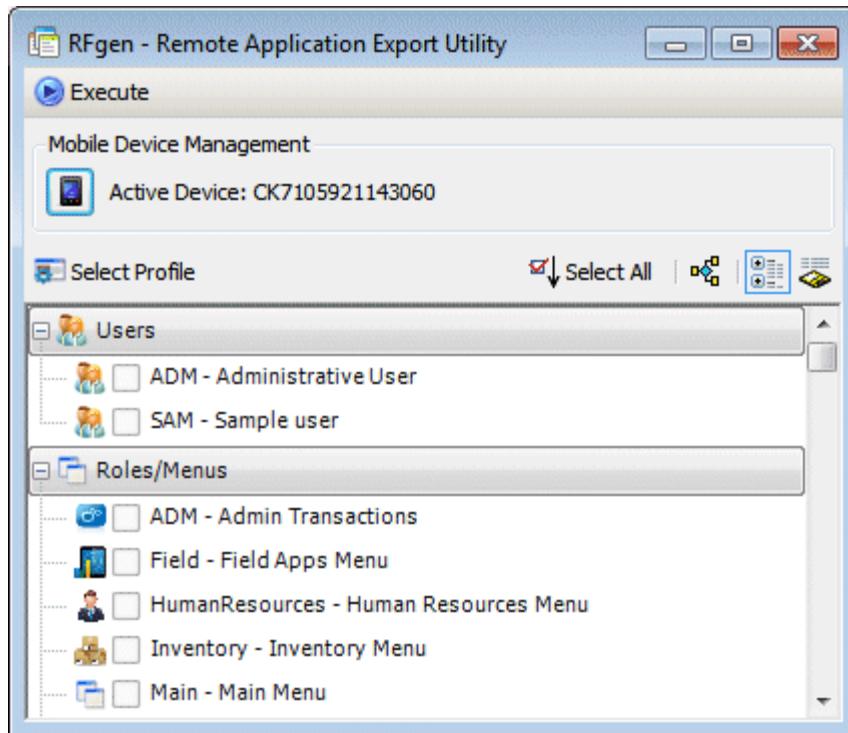


Double-click the proper entry or use the Select Device menu option. This will take the user back to the previous screen.



Selecting the Add Device menu option allows the user to enter the name and IP of a specific device or the name and IP of a broadcast address to a different subnet. These values will not appear in the previous screen

but any discovered devices in the new subset or the specific device will display.



This list is the available objects that may be selected for export to the mobile device. Using the check boxes or the Select All option and toolbar options, select the required items and click Execute from the menu. The mobile device now has an updated copy of the selected objects.



The Select Profile option goes to the device, gets the profile name, looks at the server for all objects within that profile and checks the boxes in the window corresponding to the profile. The other toolbar options allow the user to simply select all objects, select automatically associated objects and display the objects either by name or selection. Most importantly, selecting the automatic association will select all the menus, applications, macros etc. related to the chosen user for example.

## Remote Database Explorer

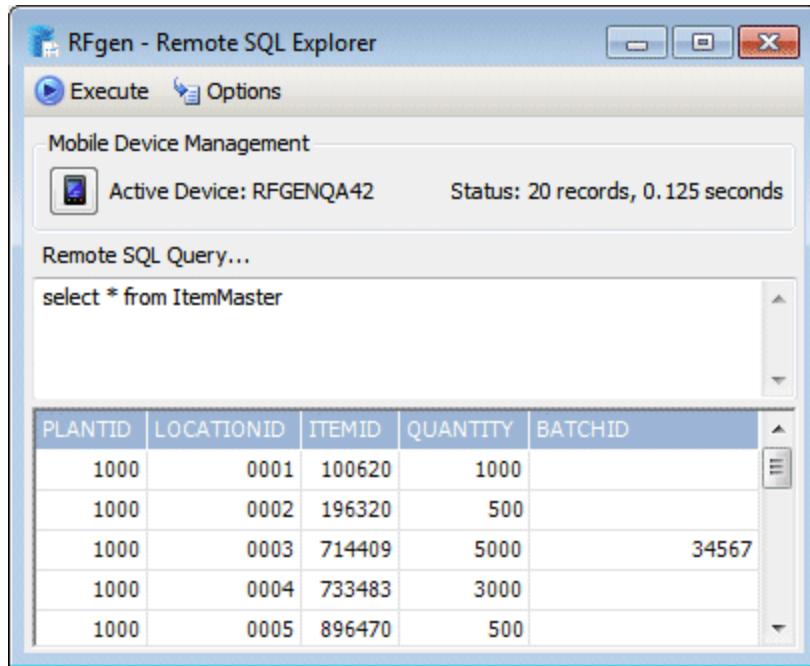
This screen is used to see the schema of the tables stored on the mobile device when the device is setup as a mobile client. To see the data in these tables use the Remote SQL Explorer.

The screenshot shows the 'RFgen - Remote Database Explorer' application window. At the top, there's a toolbar with a play button labeled 'Execute'. Below it is a section titled 'Mobile Device Management' with a smartphone icon and text 'Active Device: RFGENQA42 Status: 125 seconds'. The main area is a table viewer showing the schema of two tables:

Table	Type	Size
POBooks		
BookNo	Integer	0
OrderNo	Integer	0
PartNo	VarWChar	0
OrdUnits	Integer	0
RecUnits	Integer	0
Receipts		
TranId	Integer	0
RefDate	Date	0
RefUser	VarWChar	0
PartNo	VarWChar	0
Units	Integer	0
BookNo	Integer	0

## Remote SQL Explorer

This screen is used to inquire or update data stored in the mobile device's database. Thin client solutions do not have users, menu, applications, etc. and therefore do not contain databases.

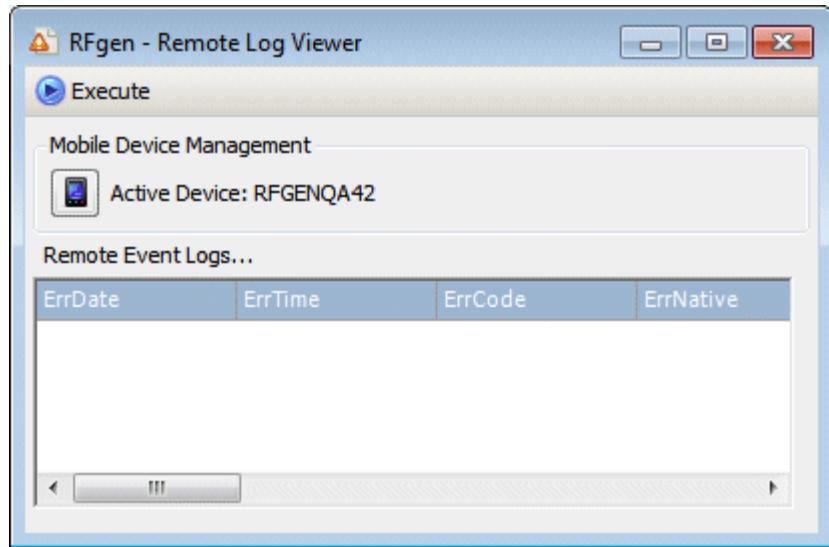


Click on the Remote SQL Explorer menu option and then select the active device. Enter any valid SQL statement, or multiple SQL statements separated by a semi-colon in the Remote SQL Query window and click the Execute menu option. In the case of multiple SQL statements, the statement that contains the blinking I-beam cursor or has been highlighted will be executed.

The Options menu item allows for a restricted number of rows to be returned in case the statements will bring back more data than desired. It will also switch the grid display into a list of previously executed SQL statements for easily repeating previous statements.

## Remote Log Viewer

This screen displays the client's error log without the user needing to copy the file to the host PC and open it manually.



## Utilities

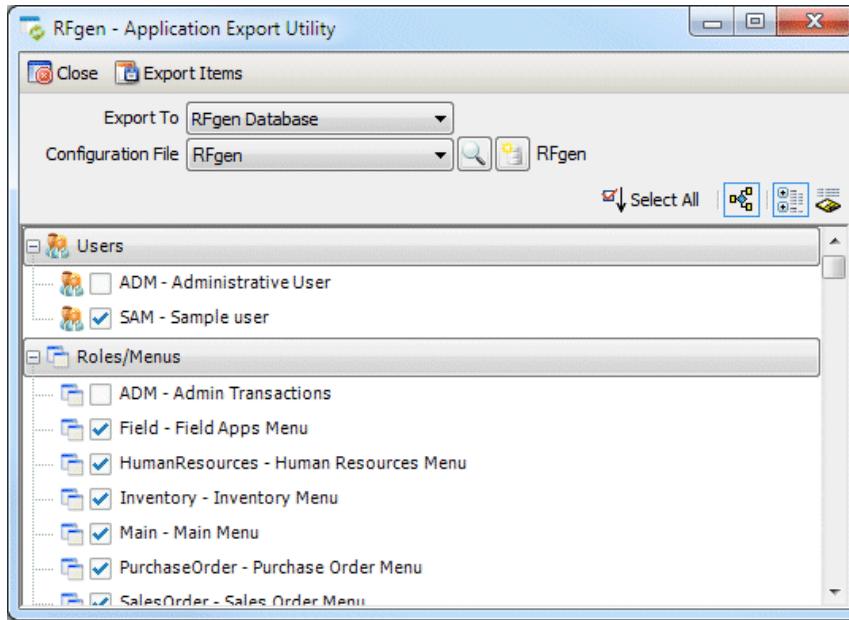
The Utilities menu option provides tools to manage objects. Objects such as users, menus, applications, VBA modules and others can be imported and exported. The ability to undo otherwise permanent save or delete actions in the solution can also be found here.

### Export or Import Mobile Applications

If your applications have already been developed and you merely wish to use them, you'll need to load the Mobile Development Studio objects (Applications, Menus, Users and VBA code and macros) into your newly installed system.

The 'Utilities' Menu Bar selection allows you to transfer specific (or all) objects into your server. Objects are freely transferable from/to other solution databases, or other external files, for the following purposes: (1) production usage, (2) ongoing development, and (3) backup / retrieval.

Note: to transfer an entire set of (same release) applications, and to overwrite the current set, simply copy the solution database from the development system to the production system while the production system is not in use. Be sure to alter any data connectors if necessary.



Here we are exporting selected objects, chosen in the 'Export To' dropdown option. A standard Windows file is an alternate choice.

Select the configuration file of a solution database (or search for one) or enter a path to the directory where you want to export the selected objects.



This menu option will toggle whether or not dependent items are also automatically selected.



If this menu option is highlighted, all possible items will be available for selection.



If this menu option is highlighted, only the already selected items will be visible.

By choosing only the user, the Export utility selected all related items from all other categories, in this case, the applications, menus and the BAS files utilized by the user. You can always remove those items that are not wanted.

The available groups for Import and Export are:

<b>Users:</b>	This includes the user, assigned applications and associated properties.
<b>Roles/Menus:</b>	The menu and assigned applications/sub-menus
<b>Applications:</b>	The application layout, VBA script, and properties
<b>Scripting Modules:</b>	VBA script modules
<b>Transactions:</b>	VBA script for database or screen mapping transactions
<b>Host Screens:</b>	The screen picture and VBA navigation script
<b>Device Profiles:</b>	The profile created to deploy a complete package to a mobile device
<b>Images:</b>	Images contained and used within the solution
<b>Mobile Themes:</b>	Theme configuration settings for the mobile devices
<b>Translations</b>	Language translations for multi-lingual solutions
<b>Vocollect Tasks:</b>	The Vocollect tasks used in the application
<b>Object Schema:</b>	Downloaded tables, stored procedures and business function definitions
<b>System File:</b>	Data connection configurations, global properties and other system level attributes can be moved between application databases.

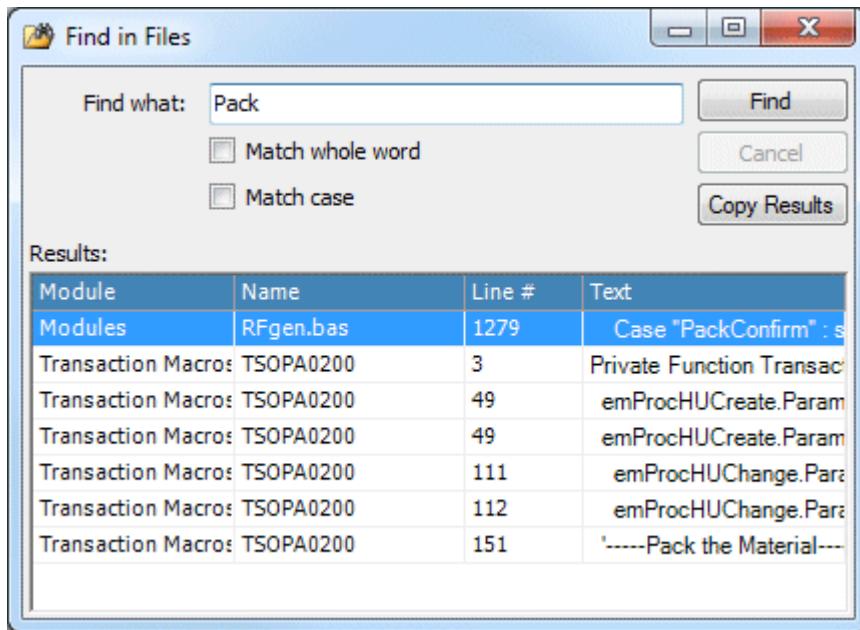
Importing objects works similarly, except that you will be overwriting items in your local solution from a remote file.

When importing or exporting items, the release number used to create the items must be the same as the release number for the items being overwritten. You will be stopped if they are not the same.

## Find in Application Scripts

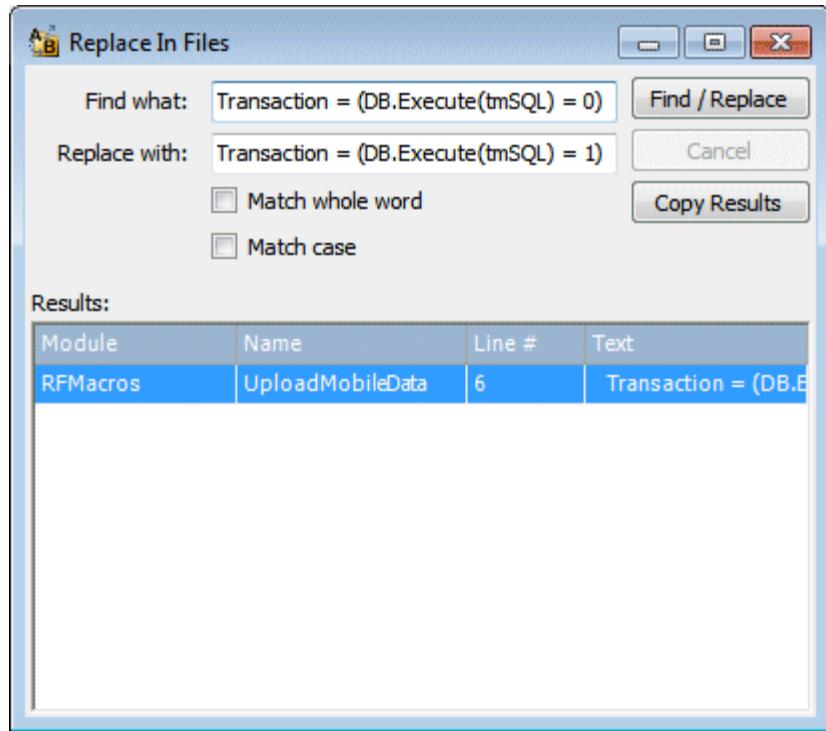
The Find in Application Scripts will generate a list of applications, modules and macros that contain in their scripts the text to be found. Double-clicking on a resulting application, module or macro will open the code window for that item and position the cursor on the correct line.

The **Copy Results** button creates a comma-delimited list of the columns displayed in the grid and places it on the Windows Clipboard.



## Replace in Application Scripts

The Replace in Application Scripts finds and replaces specified text in all applications, modules and macros at one time.



The **Copy Results** button creates a comma-delimited list of the columns displayed in the grid and places it on the Windows Clipboard.

### Undo Save/Delete Action

During the time the Mobile Development Studio is open, all saves and deletes of applications, menus, users, or other objects are listed here and can be undone. When the Mobile Development Studio is closed, this Undo list is lost permanently.

The screenshot shows the 'Restore Saved or Deleted Items' dialog box. It has a 'Close' button and a 'Restore' button. Below the buttons is a table with four columns: 'Time', 'Action', 'Table', and 'Id'. Three rows are listed in the table:

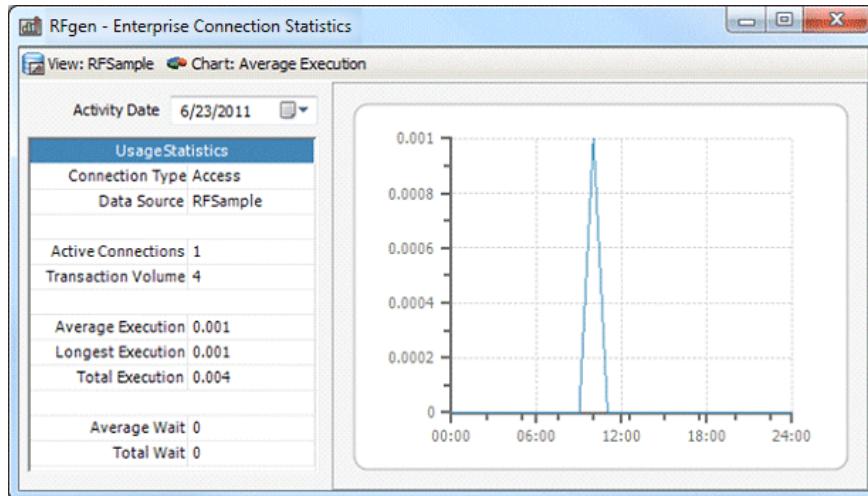
Time	Action	Table	Id
6/28/2013 2:59:31 PM	UPDATE	Applications	FieldSales
6/28/2013 2:59:15 PM	DELETE	Images	Delete
6/28/2013 1:42:13 PM	UPDATE	Device Profiles	Batch

# Reports

There are a number of logs and views that can be seen from the Reports menu option. Application Statistics, error and upgrade logs and more are available for fine tuning or debugging the system.

## Application Statistics

Clicking on the ‘Reports’ menu selection, the ‘Application Statistics’ will display statistics regarding database and ERP transactions.



The View menu option selects between each of the configured data connections and the Transaction Management database.

The Chart menu option will show the performance of a given statistic over the course of the day. The options are:



**Average Execution** is the typical time it takes to execute one call to the specified data connector. The graph shows this average across the whole day. The lower the number, the better. Typical values should be well under one second.

**Longest Execution** is the longest time RFgen had to wait for one call to the specified data connector. The lower the number, the better. Typical values should be well under one second.

**Total Execution** is an accumulated amount of time that RFgen has spent waiting for all executed calls to the specified data connector.

**Average Wait** refers to how long on average a user must wait for RFgen to provide them a connection to the specified data connector using the Connection Pooling process. Typical values should be less than one second. If the user must wait longer, then the connection pool should be increased.

**Total Wait** is an accumulated amount of time that users have spent waiting for RFgen to assign a data connection handle from the pool.

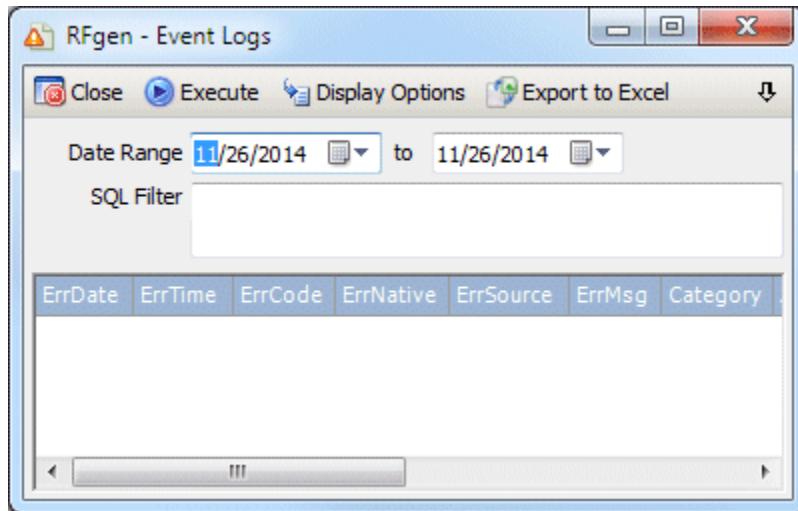
**Total requests** is the total number of times RFgen access the specified data connector for any reason.

**Total Connections** is the number of currently open connections to this data connector. Without Connection Pooling, each logged in user will have their own connection. With Connection Pooling enabled, the maximum should be the limit placed on the pool in the configuration and the minimum should be one.

To enable the Statistics, choose the Configuration / Performance Monitoring menu option, select the Record Usage option and change the value from Disabled to some increment for refreshing the data.

## Event Logs

Clicking on the ‘Reports’ menu selection and selecting ‘Event Logs’ displays a window showing system error messages.



The SQL Filter is the “where” portion of the select command in the error log table. You may add something like **AppId = ‘RFLogin’** and that would restrict the errors to that application.

Note that standard SQL 'SELECT', 'INSERT', and 'UPDATE' statements are used to process device data in conjunction with your database. If, for example, you use SQL reserved words (see SQL Reserved Words section), or include spaces in your transaction table or column field names, your data will not be processed. The specifics of your error(s) will be written in this General Error Log window. Items in the window may be viewed, printed, refreshed as necessary and cleared as desired.

The Export to Excel option creates an XLS file at the selected path. Excel does not have to be installed on the system.

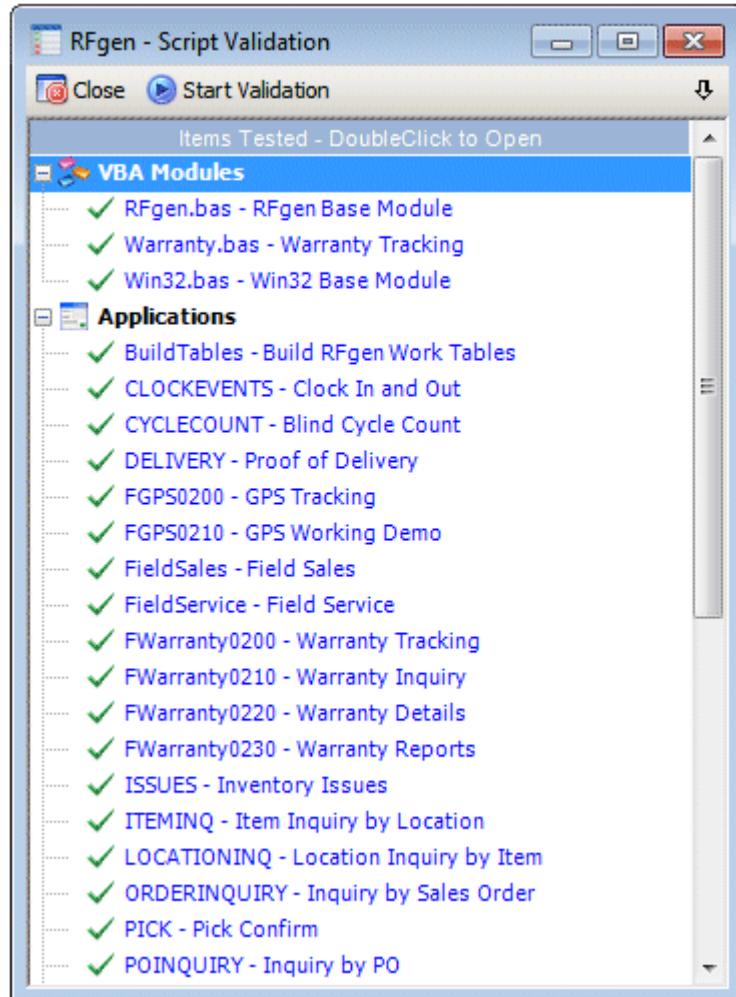
## Performance Log

The Performance Log is a list of all flagged execution events such as database, ERP, legacy host, web service and scripting executions that exceeded the millisecond threshold values. The thresholds are setup in

the Configuration / Performance Monitoring menu option. The time of the event, data source, code module, function, line number and parameters used are all displayed.

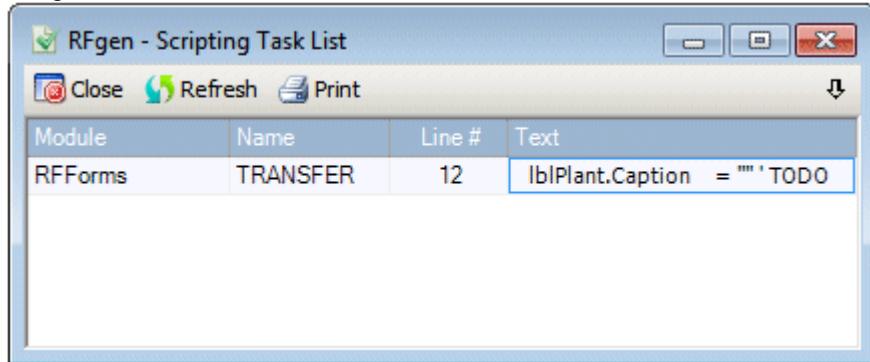
## Script Validation

This utility will perform the VBA syntax check for all coded objects. Any application or macro, etc. that has a syntactical error will display the yellow triangle-warning icon. Double-clicking on any line will load and display that code page for convenience.



## Task List

The Task List is a list of all TODO code markers suggesting the programmer left something unfinished. If the code contains a comment mark and the “todo” word then that line of code will appear in this list. Double-click any entry in the list and that script window and code line will get the focus.

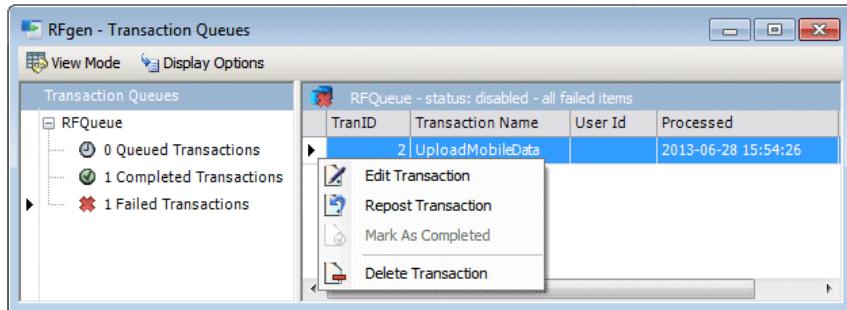


## Upgrade Log

The Upgrade Log is a list of all upgrade markers suggesting the upgrade process left something unfinished or is simply a warning requesting investigation. Double-click any entry in the list and that script window and code line will get the focus.

## View Transaction Queues

Clicking on the ‘Testing’ menu selection, then ‘View Transaction Queues’ displays a window that contains transactions.

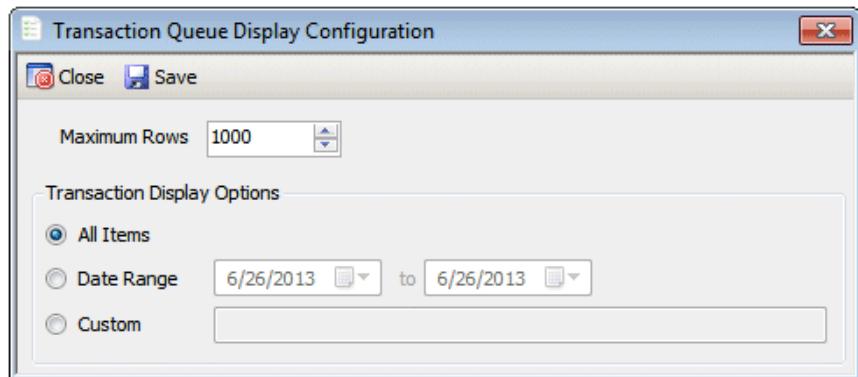


Three types of logs are available: ‘Queued’ transactions are data collection entries waiting to be posted to your host application (typically because your host has been offline or not available); ‘Completed’ transactions and ‘Failed’ transactions may also be displayed. Transactions may be edited, reposted, marked as completed or deleted by means of the right-click menu options as shown above.

Note: Queued transactions will not automatically post when using the Mobile Development Studio. This allows for the testing of posting transactions. When in production using the Server, all queued transactions are posted automatically within 60 seconds (or less depending on configuration) of the host becoming available.

The Refresh menu option simply updates the display if you believe it is necessary.

The Display Options are used to narrow down the records being displayed in this window.



Over time the list of completed transactions can become very large.

**Maximum Rows** will limit the display to the first configured number of entries. To see the most recent entries, use the data range option and set the Maximum Rows to a high value.

**Detail Mode** will show the data passed in to the macro.

Transaction Display Options – **All Items** shows an unrestricted list of entries and **Date Range** will limit the entries to a date-based on their created date.

The **Custom** option is an ability to specify your own Where clause for the lookup. The actual names of the fields in the Queue database must be known as well as the type of field. An example would be:  
where SeqNo = 1

(See *TM.GetItemsEx* for examples of table fields and types.)

# Help

Clicking on 'Help' will display these options.

**Mobile Development Manual** – displays the manual for the Studio product

**VBA Scripting COM Syntax** – displays the COM specifics of the VBA language

**VBA Scripting .Net Syntax** – displays the .Net specifics of the VBA language

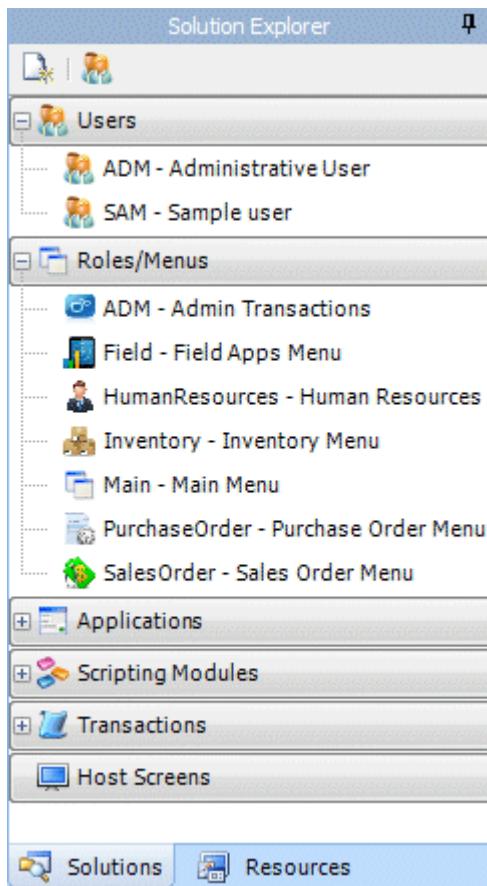
**Obtaining Technical Support** – provides links and e-mail addresses for sales and technical support

**About RFgen's Development Studio** – provides the version and release date of the product

# Mobile Development Studio Tools

The Mobile Development Studio development tabs are displayed beneath the menu bar on the left. The Solutions tab has the most commonly used aspects of development, all items necessary to create a solution. The Resources tab contains deployment and global configuration data.

## Solutions Tab Overview



Here:

The **Users** tree is used to provide user names, passwords, and a primary menu for each user.

The **Roles/Menus** tree is used to organize Applications for selection purposes.

The **Applications** tree is used to create the visual aspects (i.e., device displays) and basic validation for the data collection transaction.

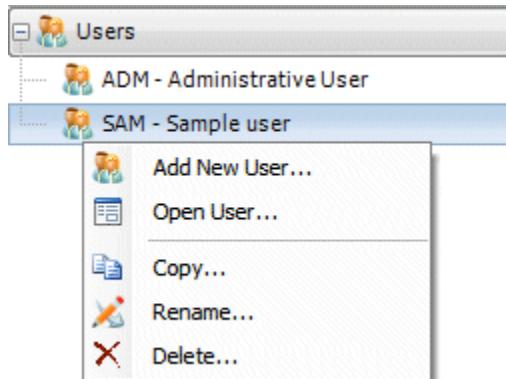
The **Scripting Modules** tree is used to manage global scripting that can be used by any application, timed event, transaction macro, etc.

The **Transactions** tree (used with Screen Mapping and non-screen mapping applications) is used to manage host (data entry) transaction macros. See the RFgen Screen Mapping documentation or the Transactions tree section below for more information.

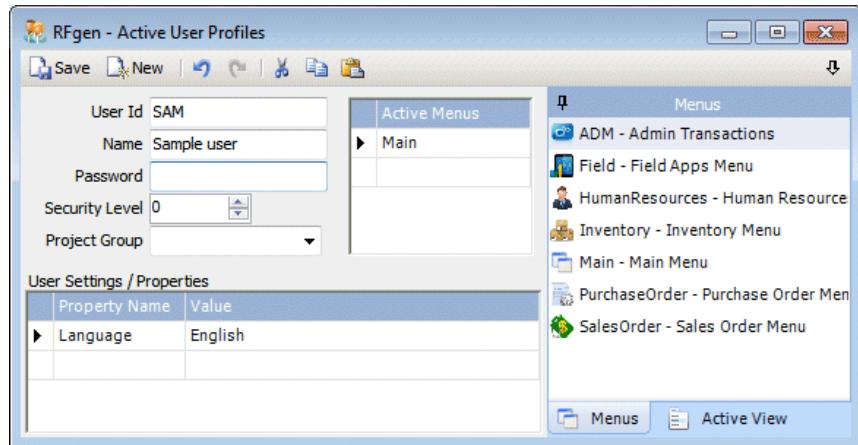
The **Host Screens** tree (used with Screen Mapping only) is used to manage host screen (navigation) macros. See the RFgen Screen Mapping documentation for more information.

## Users List

Right-click on an existing user (or in the blank space) to add a new user. Double-click or right-click on a specific user to make additional changes.



Double-clicking on an existing user opens the User Editing window.



Here a user code of 'SAM' is illustrated. A password is optional for a user account. SAM's startup menu is 'Main Menu'. Security Level is a numeric value between 0 – 100 that will be compared to the menu's required security level before allowing that user access to the following menus or forms. Other menus are embedded in the Main Menu for SAM's use (created using the Menus tree).

To assign a menu to the user double-click one or more menus to add to the Active Menus list.

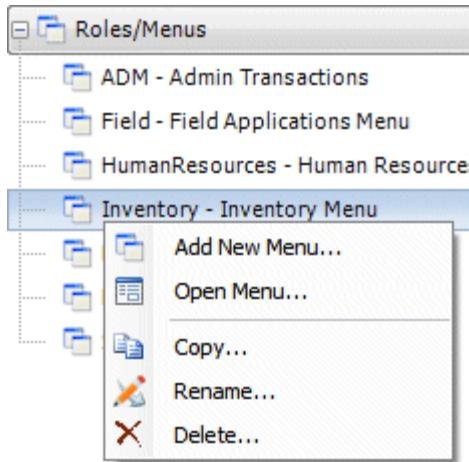
The **Project Group** property allows the users to be grouped together into categories such as "Admin" or "SecondShift" or any other named grouping. This has no impact on production. It is just an option for making the management of the development database easier.

The Optional User Settings / Properties section allows the administrator to include any property and value for the selected user for the purposes of retrieving that data at run time. This has no effect on the user logging in. This data can be retrieved using the VBA extension command App.UserProperty.

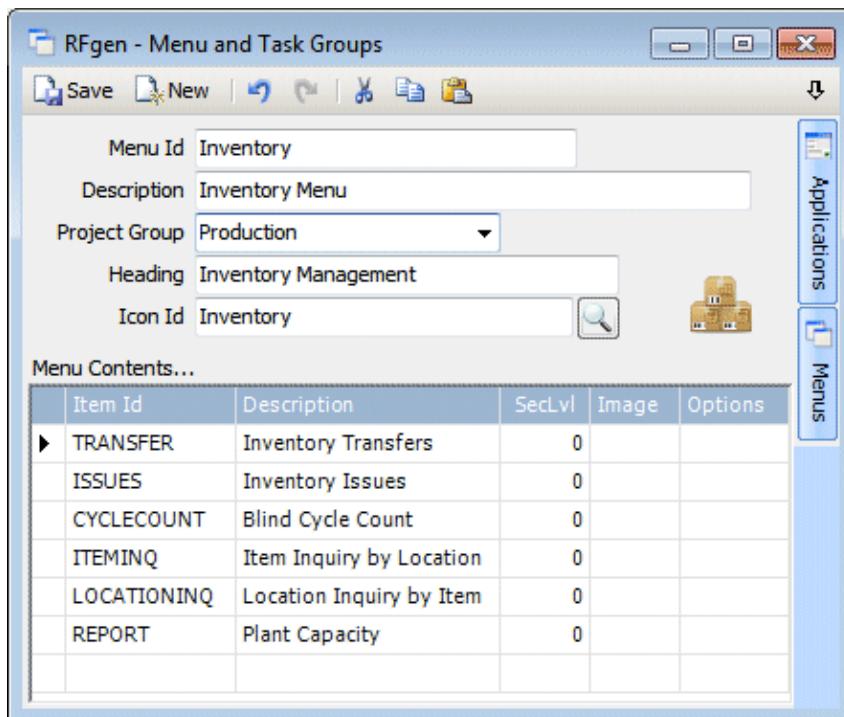
When done, click the Save menu item.

## Menu List

Right-click on an existing menu (or in the blank space) to add a new menu. Double-click or right-click on a specific menu to make additional changes.



Double-clicking on an existing menu opens the Menu Editing window.



Menus are groups of existing applications and other menus that will be made available to Users who are assigned available menus.

The **Menu Id** is the name of the menu that is used when assigning them to users. The **Description** is required and only shows in the Menu list.

The **Project Group** property allows the menus to be grouped together into categories such as “Test” or “Production” or any other named grouping. This has no impact on production. It is just an option for making the management of the development database easier.

The **Heading** value will be placed at the top of the menu and is optional. The **Icon Id** references an Image resource to represent this menu in case the menu is presented to the user with graphical elements. Graphical menu display options include an icon and text description, Windows desktop style and a mobile device graphical button view.

To select an item to appear on your menu, right-click on the first column of the row you want to insert into or delete. The user may also just press the Insert or Delete keys on the keyboard. When the correct row is selected double-click the items in the Applications or Menus lists. When done with your selections, click on the Save menu item.

In the case where one form is used for multiple purposes, variables can be defined in the Menu grid itself. Then those variables can be referenced when the application is loaded to determine how to use the application.

In a case where you want the user to select from two variations of the same application, add the application to the menu twice, give them different descriptions and then use two different command **Options**. Using the format –VARNAME=VALUE will create variables with values that can be referenced at runtime. The **Image** column allows for multiple instances of an application to have different menu icons.

In the Form\_Load event, set a global variable equal to App.GetValue("VARNAME") and based on the result, show or hide prompts or change the logic of the application. To add more than one variable to the command Options add a space and repeat as shown below.

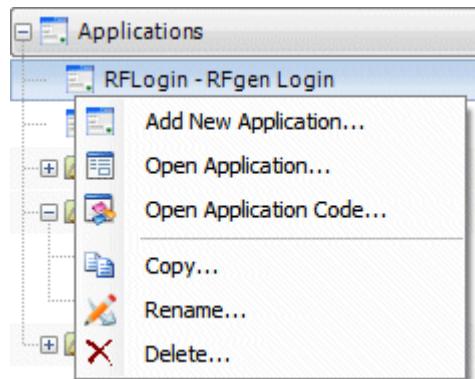
Making the application name a dash, supports comment lines within a menu.

Menu Contents...				
Item Id	Description	SecLvl	Image	Options
TRANSFER	Inventory Transfers	0	Transfer	-PLANT=10 -LOCN=DOCK
ISSUES	Inventory Issues	0		
CYCLOCOUNT	Blind Cycle Count	0		
-		0		
-	[ Inquiries ]	0		
ITEMINQ	Item Inquiry by Location	0		
LOCATIONINQ	Location Inquiry by Item	0		
REPORT	Plant Capacity	0		

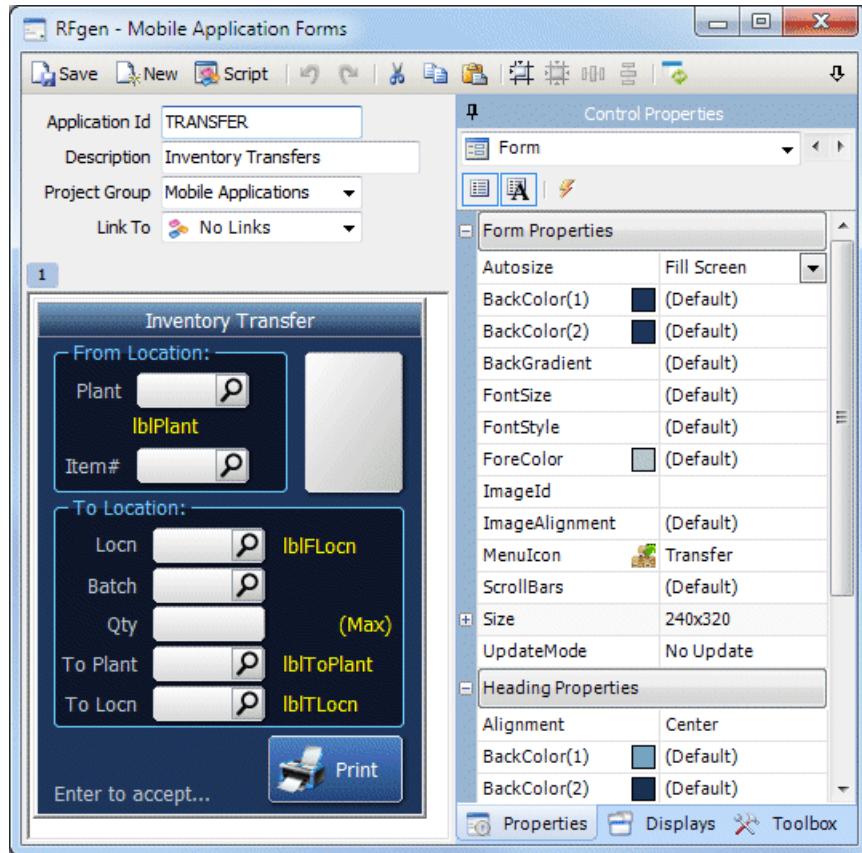
In this case, what is displayed on the menu is a blank line and then a comment line indicating that the following list of items are their own group. The user has no ability to select label entries on the menu. The highlight bar will skip over these entries. This does not do anything if in the Button or Desktop menu modes.

## Applications List

Right-click on an existing application (or in the blank space) to add a new application. Double-click or right-click on a specific application to make additional changes.



Double-clicking on an existing application opens the Application Editing window.

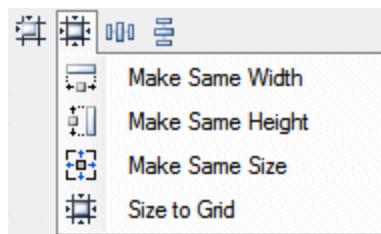
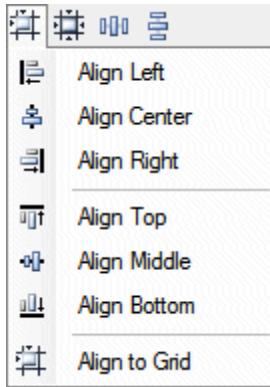


### The Application Editing Menu Bar

The following selections appear across the top of the Application window.



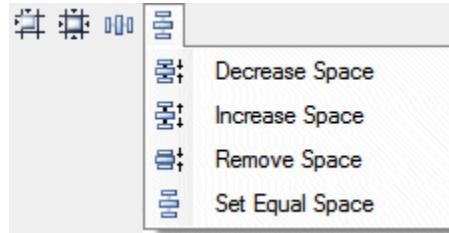
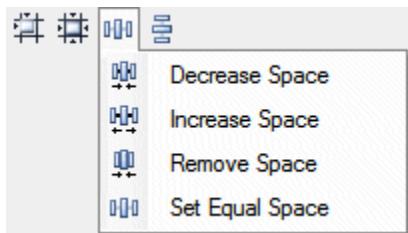
The **Save** option saves the current script to the solution database. The **New** option clears the screen so a new application may be created. The **Script** option displays the VBA script for the application. Also provided are **Undo**, **Redo**, **Cut**, **Copy** and **Paste**.



The **Control Alignment** toolbar button offers multiple alignment options when more than one control is selected. Using the selected prompt with the smallest prompt number, all alignments are aligned with that prompt.

Dragging a prompt around on the screen while holding the shift key moves it per pixel rather than by character or row. The Align to Grid option will put a prompt back in step with other prompts when moving it without holding the shift key.

The **Control Sizing** toolbar button offers sizing options when more than one control is selected. Using the selected prompt with the smallest prompt number, all sizing adjustments are aligned with that prompt. The Size to Grid option stretches a prompt to the right most edge of the form size.



The **Horizontal Spacing** and **Vertical Spacing** toolbar buttons change the spacing between the selected prompts using the first selected prompt with the smallest prompt number as the one all other prompts adjust to.

The **Right to Left Display** button flips the designed form to render in a right-to-left format for languages that typically are written right to left such as Arabic and Hebrew.

The last **Dock** icon, (the down arrow) displays on every un-docked window, allowing the user to re-dock back to the Mobile Development Studio main window. Right-clicking the tab and selecting “Undock This Item” will undock the window.

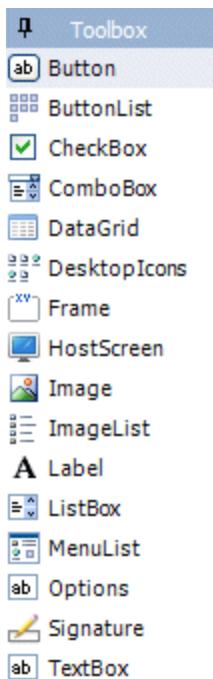
The **Application ID** and **Description** fields are required. RFgen and the menus use the Application ID to record and display the application and the Description field is the default menu description for the user.

The **Project Group** property allows the forms to be grouped together into categories such as “Inquiry Transactions” or any other named grouping. This has no impact on production. It is just an option for making the management of the development database easier.

The **Link To** drop-down property establishes if there will be a connection between the application and any pre-established back-end connections or macros. The options are:

- |                   |  |
|-------------------|--|
| No Links:         | the application does not link to any external connections or macros  |
| Database Table:   | the application will use linked textboxes and potentially automatically update a database  |
| Macros:           | the application will use a screen mapping Application Screen macro or a Data Transaction macro   |
| System Functions: | <u>Login Request</u> : if the application has a prompt called ‘User’ and a prompt called ‘Password’ then the user will be validated against RFgen’s list of users and the assigned menu will be called.<br><br><u>Menu Request</u> : if the application has a prompt called ‘Menu’ then any selection will internally be called with an App.CallForm. You must populate the menu prompt either through the List property or scripting. |
| Vocollect Tasks:  | This provides a list of Vocollect resources that the application may use. The application is acting as the data collection, validation and back-end update script only.  |

Additional objects besides linked fields are available in the Controls Toolbox. They are: Button (graphical only), CheckBox, ComboBox, Frame, HostScreen, Image (graphical only), Label, ListBox, MenuList, Options, Signature Box (graphical only) and TextBox. The following window is the toolbox.



#### The Button Object (Graphical Item)

The Button object is used to allow easy access to a function on the application such as Save or Exit. This object can only be seen in the graphical client and supports images on the button itself.

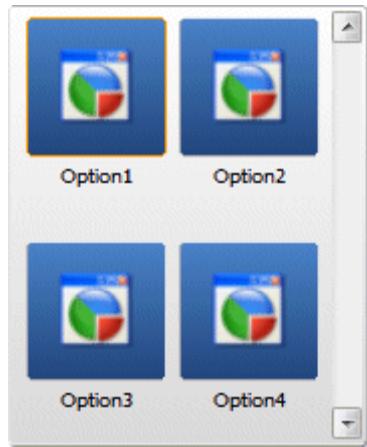


To use a Button, place it on the application and give it the appropriate caption. In text mode, the GotFocus, OnEnter and LostFocus events are executed when the image is next in the tab order. In the graphical mode, the Click event will also execute.

Note: this prompt will never hold the focus. When focus comes to this prompt it immediately processes the events and focus then moves to the next prompt.

### The ButtonList Object

The ButtonList object is a control used to display a list of options in a tiled button format. The size of the buttons and the size of the images are independent so they can be sized in any way. The text is also scalable. Horizontal and / or vertical scrollbars can be added but the control supports swiping up and down or left to right. Depending on the size of the control on the form and the size of the overall buttons, columns of buttons will be added or removed based on the available space.



### The CheckBox Object

The CheckBox object is an application prompt that allows the user to select a True or False option based on the object's label.



### The ComboBox Object

The ComboBox object is an application prompt that allows multiple items to be displayed in an area of the application, one of which may be selected. Items may be sorted and/or selected as required by the needs of the application.



To use a ComboBox, VBA script is used to populate and manage items displayed in the box or the ComboBox prompt can contain the values itself by entering them in the List property.

### The DataGrid Object

The DataGrid object is a grid control allowing multiple columns, multiple rows, scroll bars in both directions and supports swiping in any direction to move the data into view instead of using the scrollbars. The columns can auto-size or be fixed to a set width.

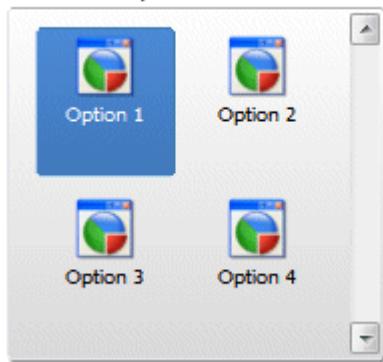
### Employee Record

Name	Phone	E-Mail

### The DesktopIcons Object

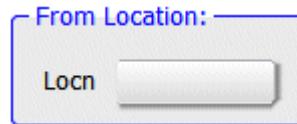
The DesktopIcons object is a control used to display a list of options in a tiled icon format. The size of the rectangle containing the icon and caption and the size of the images are independent so they can be sized in any way. The text is also scalable. Horizontal and / or vertical scrollbars can be added but the control supports swiping up and down or left to right. Depending on the size of the control on the form and the size of the overall icon, columns of icons will be added or removed based on the available space.

Select an Option



### The Frame Object

The Frame object is used to put a box around other prompts to give a grouping effect. The Frame prompt sequence must come before the prompts that will be inside the frame. Otherwise, the frame will cover the prompts on the inside.



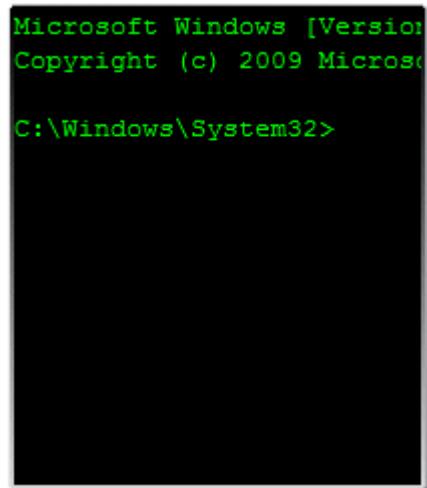
The frame can be stretched to be either a single horizontal line or a vertical line and its caption is optional.

Note: this prompt will never hold the focus. When focus comes to this prompt it immediately processes the GotFocus, OnEnter and LostFocus events and focus then moves to the next prompt.

### The HostScreen Object

#### (Graphical Item)

The HostScreen object provides a pass-through telnet emulation window on a client display. You can map a form to an alternate console application or Telnet service. Placing this control on a form would allow the user to use forms for some data collection and this form to execute SAP Console transactions, Telnet to a legacy host system or simply start any executable with console output. It is configured as a Screen Mapping connection with an emulation type of 'Console'. All the keystrokes pass into the prompt's KeyPress event allowing the user to monitor the keyboard and trigger actions like exiting the application based on custom keystrokes. Only one of these controls is allowed per application.



### The Image Object (Graphical Item)

The Image object is used to display a picture on the application. This object supports a large variety of image formats in a Thin Client environment. They are BMP, DIB, GIF, JPG, WMF, EMF and ICO. This object can only be seen in the graphical version of the client. When using the Windows CE / Mobile environment, this control only supports the BMP format.



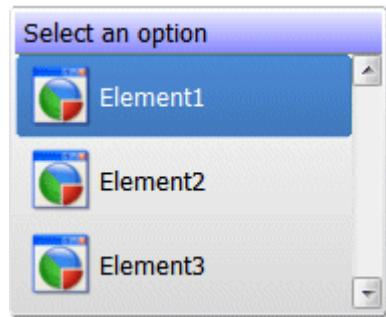
To use the Image object browse for the desired image using the Image property and select it from the Image Resources. The GotFocus, OnEnter and LostFocus events are executed when the image is next in the tab order or when the user clicks on the image. In this case, the Click event will also execute. The GotFocus, OnEnter and LostFocus events will execute in character mode.

At runtime check the Defaults property to get the name of the image resource currently loaded into the prompt. If it is blank the image may have come from the file system using the RFPrompt( ).ImagePath property.

Note: this prompt will never hold the focus. When focus comes to this prompt it immediately processes the events and focus then moves to the next prompt. If it is simply clicked on, the focus will stay where it was before the click.

#### The ImageList Object

This object is a control used to display a list of options in an icon list format. The image size dictates the size of the row's height. The text is also scalable. Horizontal and / or vertical scrollbars can be added but the control supports swiping up and down or left to right. If a title row is not required it can be turned off.



#### The Label Object

This object is used to add additional text to the screen not necessarily associated with one prompt. All of the default objects come with their own label. For example, if the customer desired the text "F4=Exit" at the bottom of the screen, a label would be used. If a prompt's label required 2 lines, you could use the prompt's initial label for the first line and add a Label object to fill in the second line. Generally changing the color of the labels sets them apart so the user knows the difference between input fields and display-only fields.



Note: this prompt will never hold the focus. When focus comes to this prompt, it immediately processes the GotFocus, OnEnter and LostFocus events and focus then moves to the next prompt.

Additionally a Label control can be given a back color and stretched into a panel type object to provide areas that stand out.

#### The ListBox Object

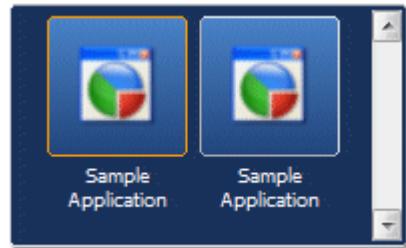
The Listbox object is an application prompt that allows items to be displayed in an area of the application. Items may be sorted and/or selected as required by the needs of the application. A listbox can have multiple columns.



To use a ListBox, VBA script is used to populate and manage items displayed in the box, or the ListBox prompt can contain the values itself by entering them in the 'List' property. A ListBox is different from a SearchList object. Clearing the application prompts and displaying items in a list in full screen mode uses a SearchList.

#### The MenuList Object

The MenuList object when used as a menu control has options for the display such as an Image List, Button selection and a Desktop icon layout. When used like a Listbox control it can only show one column.



#### The Options Object

The Options object (when used) is typically the last prompt in a data entry or display application and is used to write data to the linked database table. This prompt has no effect if the prompts are not linked fields. The object is a prompt that provides a pause in the data entry process so that the device user may select which of four options should be taken next.



Four choices are available with this object:

'V' = verify (cycle through) the current data

'F' = file the data (write to the appropriate table)

'N' = next, i.e., don't write this data

'Q' = quit, i.e., don't write, then quit the process.

When the Options object appears, all processing is suspended until one of the choices above is entered. 'F' is the default, so that if just the device <Enter> key is pressed, the data collected in the application is filed (i.e., written) to the associated database table(s).

This control can be replaced with a standard textbox control that accepts any values of your choosing and then processes the user response in the OnEnter event, possibly even performing SQL statements to write data to the database.

### The Signature Object (Graphical Item)

The Signature object is used to capture any hand-written text that will then be saved as a 2-tone bitmap image. This image can be placed in the Text property of the signature prompt to be re-displayed. There is no built in pattern matching to compare recorded signatures. Since the bitmap is saved in black and white, changing the background color to red, for example, will make the box appear solid black since red is a dark color and converted to black.

Sign here



Signature

A rectangular button-like control with rounded corners. The word "Signature" is centered in a large, bold, dark font. Above the button, the text "Sign here" is displayed in a smaller blue font.

The Text property of this prompt will contain a text representation of the signature which can easily be stored in a character type field in a database. The Bitmap property contains a byte stream representing the signature which can be written to a BMP file if desired or stored in a binary type field in a database.

### The TextBox Object

TextBox

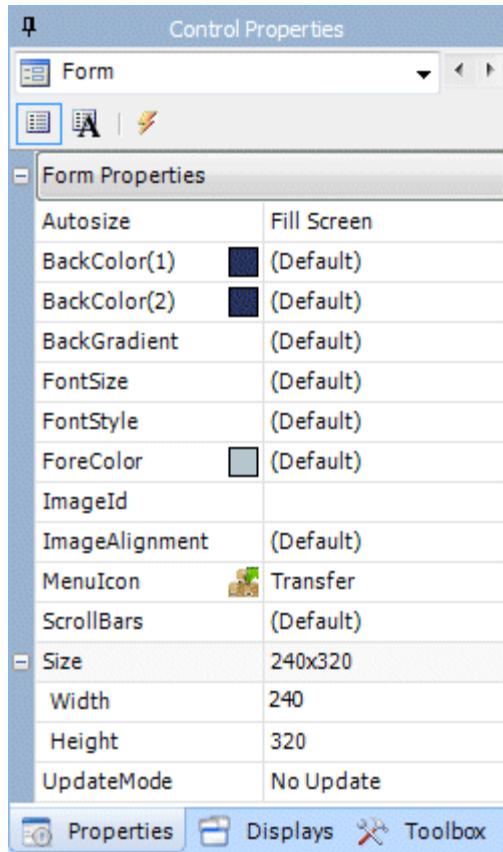


The TextBox object is a text box that can be used for many purposes. Dragging or double-clicking on the TextBox control places a textbox on the screen. Once added, you may set the properties for the object including the data to appear in it. For example, the data display area may be used to print (translate) data from your database using the '@T' default Property (see the Data Entry Default Properties section). Translate data is typically data from your database; i.e., 'CA' is entered at a prompt, 'California' is retrieved from your database and displayed in the unlinked textbox field. Defaults other than 'translate' may also be used.

## **Properties Tab**

The Properties tab contains the properties specific to the form or prompt depending on what is selected. The properties for the control are selected first followed by the control's label properties and finally the control's events.

### Form Group (Control)



**Autosize** will stretch the form's background based on the options of None, Contents, Horizontal, Vertical and Fill Screen.

The application's **BackColor(1)** and **BackColor(2)** properties are used to create either solid backgrounds or gradients depending on the option chosen in the Background Fill property.

The **BackGradient** property uses either just the BackColor(1) for solid backgrounds or both BackColor properties to create gradients in one of several directions.

The **FontSize** and **FontStyle** properties change the size and style of the prompts on the screen.

The **ForeColor** property changes the colors of the labels of the prompts on the screen.

The **Image Id** property sets an Image Resource as the background of the screen. The **Image Alignment** is used to hide or display that image in a number of different ways.

The **Menu Icon** property specifies an image from the Image Resources stored with the solution that will display on the menu if the menu is set to Graphical List or Desktop Icons mode. If the Image column is used on the menu itself, it will override the icon choice here.

The **ScrollBars** property determines if and how scroll bars are used to display this form at runtime.

For graphical applications the **Size** property is the height and width in pixels for the form. For character-based applications the size represents fixed-width characters in rows and columns.

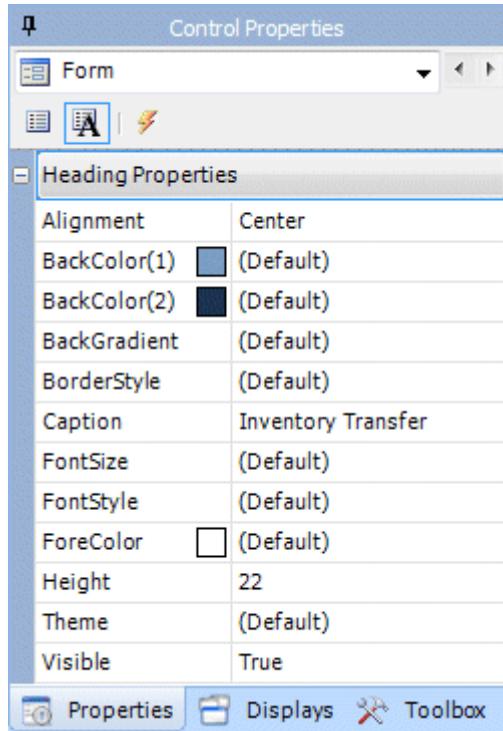
**Update Mode** has 3 options.

No Update – which will not try and update linked tables with collected data.

Update Links – which will update the linked table and for macros the application will try and call the linked macro if it has not been called already. If you set one or more macro fields as “key fields”, then once all those have received values, RFgen calls the macro. If no key fields are defined, the macro executes at the end of the application’s data entry. This field has no meaning for Vocollect links.

Exit Form will take the user back to the calling menu without updating linked tables.

#### Form Group (Label)



**Alignment** refers to the position of the form caption within the form heading area.

The form's heading area **BackColor(1)** and **BackColor(2)** properties are used to create either solid backgrounds or gradients depending on the option chosen in the form's label Background Fill property. The user can select from a color pallet or enter a 6 character hex value (for 16 million colors) to set the back color.

The **BackGradient** property uses either just the BackColor(1) for solid backgrounds or both BackColor properties to create gradients in one of several directions.

The **Border Style** changes the shape of the border on the Form's caption title bar.

The **Caption** property will place text at the top of the form and persist for each additional page should there be more than one.

The **FontSize** and **FontStyle** properties change the size and style of the caption in the form's heading.

The **ForeColor** property changes the color of the caption in the form's heading.

The **Height** of the form heading area is specified in pixels. The default is 22.

The **Theme** property changes the border of the title bar area to one of several hard coded styles.

The **Visible** property turns on or off the heading portion of the form.

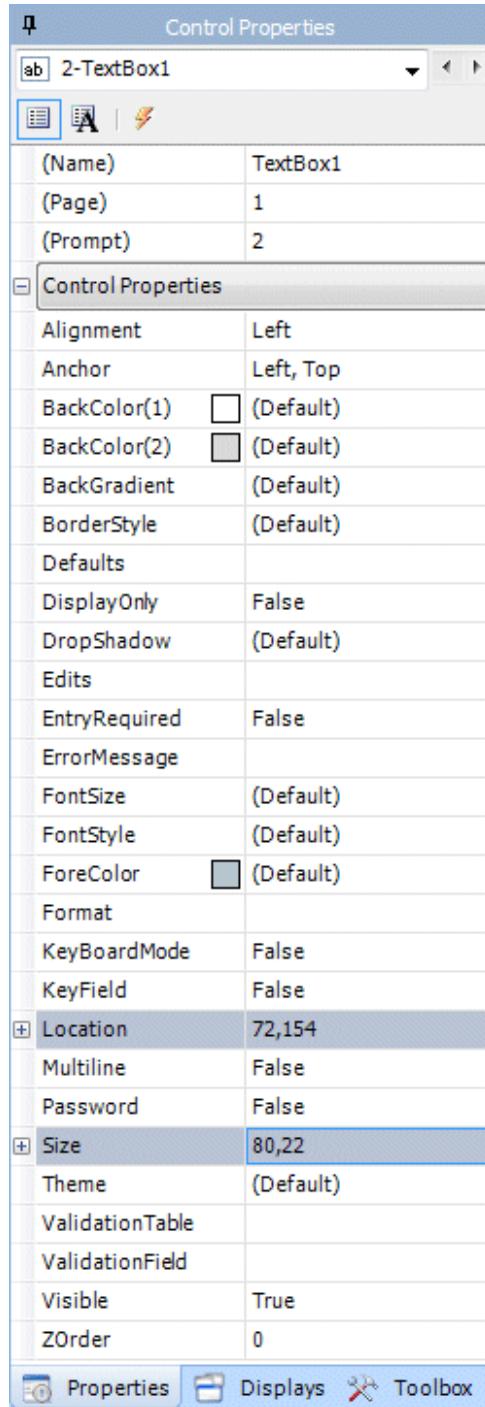
#### Form Group (VBA Events)

Visual Basic for Applications (VBA) compatible programming scripts may be used to achieve results not otherwise available within the Mobile Development Studio. VBA scripts allow technical personnel to use application, field and other events to provide additional functionality (e.g., field defaulting, error checking, network printing, etc.) to standard RFgen applications. Special pre-built VBA 'language extensions' dealing with numerous aspects of the data collection process have been added to simplify programming efforts.

To enter the VBA programming environment from the Application Properties Window, click either the Script menu option or on a 'VBA Events' property under the "show control events" toolbar button  in the Control Properties. The specifics about the VBA events will be discussed later.

#### Field Control Properties

Not all properties appear for all control types. Image controls, check box controls and others will have unique properties.



The **(Name)** is the internal name used to identify the prompt in the script. Standard programming practices suggest using the Hungarian notation where textboxes are named 'txtPart' and list boxes are named 'lstParts' as examples. This way, when referring to them in the script, there is an inherent understanding of what types of data will be used for the prompt.

The **(Page)** property shows which display page the prompt is on and it can be changed to move prompts to other pages. If a prompt is moved from page 1 to page 2, all following prompts in the prompt sequence will also be changed to page 2. If you change the page number from 2 back to 1, any prompts that come before the changed prompt will also have their property changed to page 1. The best approach is to make sure all the prompts are in the proper order and then break them up into multiple pages. There is no limit to the number of pages you can use.

The **(Prompt)** property shows the position of the selected prompt in the sequence of all prompts in the application. This property can be changed to re-order the selected prompt to a different position. If the new position is between 2 other prompts that are both on a different page, the Page Number property will also change otherwise it will not.

The **Alignment** property allows controls to shift their contents to the left, center or right side of the control.

The **AllowFocus** property will turn on or off whether the focus will stay on the selected prompt or not leave the previous prompt. Events will still execute such as a checkbox's Click event even though the AllowFocus property may be set to False.

The **Anchor** property moves the prompt's field portion relative to one or more of the sides of the display. The object could be placed at the top, left, right or bottom of the display or stretched between any of the four sides.

The **Autosize** property will expand the object to the lowest and right-most portion of the screen. The upper left corner is static. Other controls have the options of None, Contents, Horizontal, Vertical and Fill Screen.

The control **BackColor(1)** and **BackColor(2)** properties (graphical mode only) are used to create either solid backgrounds or gradients depending on the option chosen in the control's Gradient Fill property. The user can select from a color pallet or enter a 6 character hex value (for 16 million colors) to set the back color.

The **BackGradient** (graphical mode only) property uses either just the BackColor(1) for solid backgrounds or both BackColor properties to create gradients in one of several directions.

The **BorderStyle** (graphical mode only) property controls the border of the prompt. Options are Standard, Active Border, No Border, Visible with Focus and Transparent.

**ButtonSize** (graphical mode only for ButtonList, DesktopIcons and MenuList) will size the buttons displayed.

**ButtonStyle** (graphical mode only for buttons) sets if the button will contain graphics or just text.

The **Caption** property (graphical mode only for a Button prompt) contains the text portion of the label of the prompt and only exists under the Field group for a command button.

The **Checked** property sets the status of a checkbox prompt.

The **ColumnCount** property sets the number of columns the control will display.

The **ColumnInfo** group contains a number of properties specific to formatting a column within a control. Most of these properties are the same as the other identically named properties. The few that are unique are: **Image** – the same as ImageMode which provides a way to stretch, tile or move the image within the image control, such as Top Center or Bottom Right. **TrimSpaces** will remove leading and trailing spaces from the data so column alignment will be smaller. The **Width** property can either be set to a specific size or -1 to indicate that the column should stretch to the right taking up any available space.

The **Defaults** property sets any number of built-in values or custom values as the initial value of the prompt. See the *Default Property Details at the end of this section for a complete list of options*.

The **DisplayOnly** property, set to True, makes a prompt into a display only field, while setting it to False, allows users to access and change data at the prompt.

**DropShadow** places a dark border around the bottom and right sides of the control for a 3D effect.

The **Edits** property sets any number of built-in values as the requirement for the entered data. See the *Edit Property Details at the end of this section* for a complete list of options.

The **EntryRequired** property, set to True, forces users to enter data into the prompt, while setting it to False, allows users to skip the field. If the prompt never gets the focus, this property will not get used.

The **ErrorMessage** property is text displayed as an App.MsgBox when the data entered fails to meet the criteria in the Edits property.

The **ExtendedColumn** specifies which column will be stretched to the right edge of the control. The default is the last column designated by -1 but specifying 1, 2, or 3 as examples would use the remainder of the width by stretching a middle column.

The **FontSize** property (graphical mode only) sets the prompt's data field display to a particular font size. The default is 10.

The **FontStyle** property (graphical mode only) sets the prompt's data field display to a particular style. The default is Normal. Other options are combinations of Bold, Italic and Underline.

The **ForeColor** property (graphical mode only) allows the user to select from a color pallet or enter a 6 character hex value (for 16 million colors) to set the fore color of the label caption of the prompt.

The **Format** property is an extension of the VBA Format command and pre-formats the entered data to the mask entered here. See the VBA Format command for examples. The double quotes are not necessary as they are in the VBA Format command.

The **FrameStyle** property lets the user create rectangles, vertical or horizontal lines for the frame control only.

The **GridHeading** property turns on or off an extra row at the top of the control where the column heading would appear as opposed to using the control's caption for column headings.

The **ImageId** property (graphical mode only) is for image controls only and retrieves a graphic file from the Image Resources and displays it in an image prompt. When using thin client mode, the supported image types are BMP, DIB, GIF, JPG, WMF, EMF and ICO. When running in mobile mode on a CE device, only BMP is supported. Buttons also support images.

**ImageMargin** provides left, right, top and bottom padding for the image inside the control.

**ImageMode** provides a way to stretch, tile or move the image within the image control, such as Top Center or Bottom Right.

**ImagePath** is the file path to an image located on the hard drive instead of the image resources.

**ImageSize** is the width and height of the graphic itself regardless of the size of the control. Images can be displayed a number of ways and this property sets the image size for graphical lists, button or desktop menu lists.

The **KeyboardMode** option can be set to bring up a soft keyboard for input when the text box gets the focus.

The **KeyField** property is for linked textboxes only and designates which prompts will be used as key fields when attempting to perform an internal SQL Update statement for the linked application. This property is automatically filled in when the user downloads a table or view structure and links the application to that structure.

The **LineColor** property selects the color of the lines between rows or columns in a control that supports multiple rows or columns.

The **ListData** property is for list boxes, combo boxes and list views only and contains a collection of values to be assigned to the prompt when the application loads.

The **ListHeading** property allows the code environment to overwrite the caption of the prompt with formatted data from a database lookup using the Prompt.List.SetColumn method.

The **ListHeight** property is for combo boxes only and sets the number of rows the control will use when displaying a list of possible values.

The **ListSorted** property is for list boxes, combo boxes and list views only and keeps the contents of the list sorted.

The **ListStyle** property changes the presentation of the data displayed between a Standard text list, an Image List that uses images next to the text description, Buttons or Desktop style like a Windows desktop. This is the control used on the internal RFMenu form.

The **Location** property sets the position of the control in pixels for graphical applications and in rows and columns for fixed-length character applications.

The **Margins** property is used to pad the spacing between the rows or images of the displayed data.

The **Multiline** property tells the prompt to word-wrap its contents. The height of the prompt will need to be tall enough to support this visually.

The **NormalizeText** property will trim the spaces from both sides of the displayed data or captions of the buttons or desktop icons.

The **Overflow** property specifies which way the remaining items will be displayed. If there are more items than will fit on the device's screen this option can be set to horizontal or vertical which means the user can swipe bottom-to-top or right-to-left to access the remaining data.

The **Password** property, for the data field portion of the prompt, sets the display of the text equal to asterisks (\*) instead of clear text.

The **ScaleDownText** property increases or decreases the icon captions to better fit under the buttons or desktop icons.

**Scrollbars** can be disabled or enabled for horizontal scrolling, vertical scrolling or both. This property is for select controls only.

The **ShowLines** property will hide or show the lines between rows and columns. The options are (Default) which uses the theme properties, None for hiding all the lines, Horizontal for showing only the lines between rows, Vertical for showing only the lines between columns, and Both for showing lines between rows and columns.

**SelColor** refers to the color of the selection bar shown in controls like the combo box or list box. The highlighted value is what will be chosen when the user presses the enter key.

The **Size** of the rectangle that contains the control is specified in pixels but could be influenced by the Autosize property.

The **Source** property (the HostScreen control only) selects an executable to be emulated within the Host Screen control.

**StretchImage** is used to either shape an image to the size of the control or allow the image to be its natural size whether it fits in the control or not.

The **Theme** property changes the border of the title bar area to one of several hard coded styles.

The **UseMenuTheme** property will override the local properties and apply the default theme properties for the menu control.

The **ValidationTable** property presents a list of downloaded tables that can be used to verify that the data entered already exists in this table and the Validation Field. The two properties must be used together.

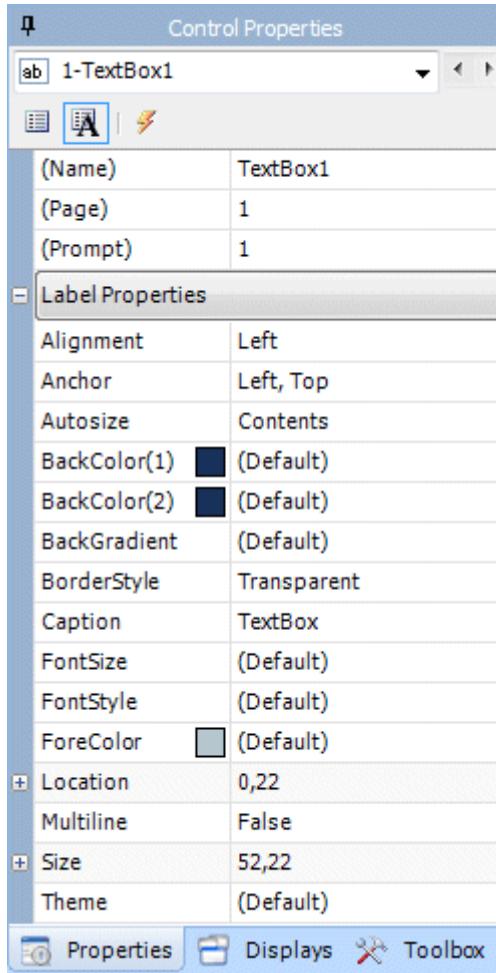
The **ValidationField** property presents a list of table fields specified by the Validation Table property. This is the reference field to determine if the data entered in the prompt already exists. If it does not, the Error Message property will be used to warn the user.

The **Visible** property, set to True, makes a prompt visible, while setting it to False makes it invisible. Even though the prompt may be invisible, the GotFocus, OnEnter and Lost Focus events will still be executed for this prompt if the focus automatically shifts from a prompt before this prompt to one after this prompt.

The **ZOrder** property allows one prompt to be on top or behind another prompt in the same place without changing the prompt sequence. This is used when one prompt is hidden and another made visible in the code based on the data collected. The default is zero. The higher the number, the more on-top the prompt is.

#### Field Label Properties

Not all properties appear for all control types. Image controls, check box controls and others will have unique properties. In the case of the buttons and frames, they don't have label properties.



The **(Name)** is the internal name used to identify the prompt in the script. This is the same property as the prompt's control property.

The **(Page)** property shows which display page the prompt is on and it can be changed to move prompts to other pages. This is the same property as the prompt's control property and cannot be used to separate the label from the prompt.

The **(Prompt)** property shows the position of the selected prompt in the sequence of all prompts in the application. This is the same property as the prompt's control property.

The **Alignment** property allows controls to shift their contents to the left, center or right side of the control.

The **Anchor** property moves the prompt's label portion relative to one or more of the sides of the display. The object could be placed at the top, left, right or bottom of the display or stretched between any of the four sides.

The **Autosize** property for a MenuList or Host Screen control will expand the object to the lowest and right-most portion of the screen. The upper left corner is static. Other controls have the options of None, Contents, Horizontal, Vertical and Fill Screen.

The control **BackColor(1)** and **BackColor(2)** properties (graphical mode only) are used to create either solid backgrounds or gradients depending on the option chosen in the control's Gradient Fill property. The user can select from a color pallet or enter a 6 character hex value (for 16 million colors) to set the back color.

The **BackGradient** (graphical mode only) property uses either just the BackColor(1) for solid backgrounds or both BackColor properties to create gradients in one of several directions.

The **BorderStyle** (graphical mode only) property controls the border of the prompt. Options are Standard, Active Border, No Border, Visible with Focus and Transparent.

The **Caption** property contains the text portion of the label of the prompt and only exists under the Field group for a command button.

**DropShadow** places a dark border around the bottom and right sides of the label control for a 3D effect.

The **FontSize** property (graphical mode only) sets the prompt's caption area to a particular font size. The default is 10.

The **FontStyle** property (graphical mode only) sets the prompt's caption area to a particular style. The default is Normal. Other options are combinations of Bold, Italic and Underline.

The **ForeColor** property (graphical mode only) allows the user to select from a color pallet or enter a 6 character hex value (for 16 million colors) to set the fore color of the label caption of the prompt.

The **Location** property sets the position of the control in pixels for graphical applications and in rows and columns for fixed-length character applications.

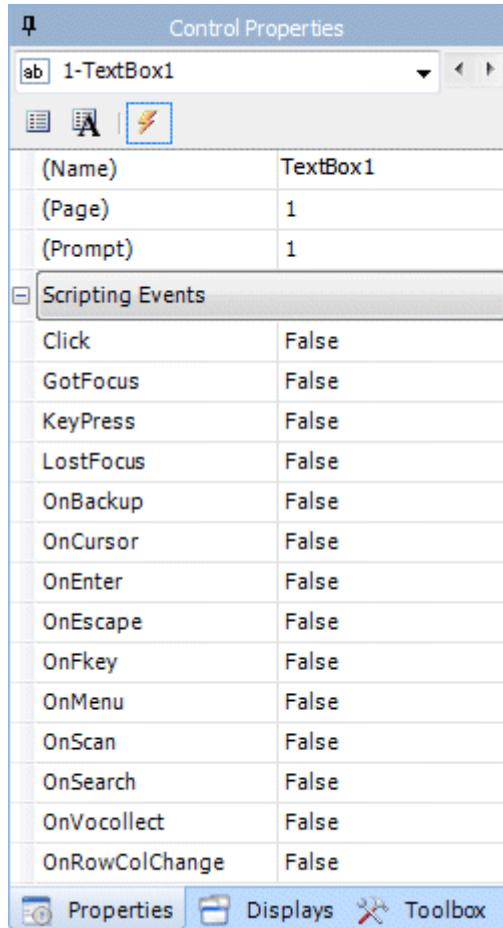
The **Multiline** property tells the prompt caption to word-wrap its contents. The height of the label will need to be tall enough to support this visually.

The **Size** of the rectangle that contains the control is specified in pixels but could be influenced by the Autosize property.

The **Theme** property changes the border of the title bar area to one of several hard coded styles.

The **ZOrder** property (for the label control) allows one label to be on top or behind another label in the same place without changing the prompt sequence. This is used when one label is hidden and another made visible in the code based on the data collected. The default is zero. The higher the number, the more on-top the label is.

#### Fields VBA Events Group



To enter the VBA programming environment from the Application Properties Window, click either the Script menu option or on a ‘VBA Events’ property under the “show control events” toolbar button in the Control Properties. The specifics about the VBA events will be discussed later.

## Default Property Details

A default property allows a data value to be generated automatically. A default property is available for each application entry prompt and defaults are entered in the Properties window. Placing a ';'O' (semicolon and the letter O, not zero) after the default parameter is optional, and allows the default to be overwritten. A blank value means that there is no default. Note: if using the VBA scripts, the Got\_Focus event for a prompt contains a variable called 'RSP' which may be set to the value of

the data default and a variable called 'AllowChange' (for overriding the value) which represents the value of the ';'O' expression; i.e., AllowChange = True, is the same as placing a ';'O' after the specified default property.

### **Text Default**

This will default the string value as entered. A string value can be any number, combination of letters, or special characters. Format is:

String value(;O)

1000;O means default equals '1000', allow 'O'verwrite. Values contained in ( ) are optional.

### **System Date Default**

This will default the current Windows system date in the format specified by 'exp1'. If no format is specified then the date will display using the format specified by the data item used for the prompt. Format is:

@DATE(;O)

@DATE;O means default equals current date displayed using MM/DD/YY format

### **System Time Default**

This will default the current Windows system time using the format specified by the data item used for the prompt. Format is:

@TIME(;O)

@TIME(;O) means default equals current time

### **Translate Default - Displaying Data From Other Tables**

This will extract data from another table; i.e., 'translate' the data. Format is:

@T,table,exp1,exp2,exp3,(exp4)(;O)

e.g.: @T,STATES,ID,3,NAM

Where:

- Table is the name of a database 'translate table'.
- exp1 is the column/field name for the translate table where indexed data is found.
- exp2 is the current prompt number or column/field name where the key for the indexed table data (i.e., data to match on) can be found; value must be a number lower than the current category number.
- exp3 is the column/field name for the translate table which contains the data to be retrieved.
- exp4 an optional expression used only if exp2 is an 'unlinked' textbox field (i.e., not a database field for which RFgen can determine a data type); leave blank if exp2 data is numeric; let exp4 be a single quote " " if exp2 data is a string.

@T,STATES,ID,3,NAM retrieves a state name from a table called STATES. Here 'ID' is an indexed column/field name for the STATES table, and 'NAM' is the column/field name that contains the state name. Data from the current application, in prompt '3', provides the value of ID to use for the indexed search of the STATES table

### **Concatenation Default**

This will concatenate any items specified, and default the resulting string. Valid items are entry categories or items enclosed by quotes. Format is:

@C,exp1,exp2,...,expN(;O)

@C,4,5,6,'\*\*\*' means default equals prompt '4' value, joined with the prompt '5' value, joined with the prompt '6' value, joined with the text string '\*\*\*'. Since no ';O' is present, the data will be entered and the application will continue at the next prompt

### **Character String Extraction Default**

This will allow the default to be a portion of a previously entered input. Format is:

@SE,exp1,exp2,exp3(;O)

Where:

exp1 = entry category for the extraction.

exp2 = starting position of character string.

exp3 = length of string.

@SE,3,4,5;O means default equals data entered at prompt '3', starting at the '4'th character, and continuing for '5' characters

### **Calculation Defaults**

This will allow the default to be the result of an on-line calculation. Calculations may be performed using previously entered entry categories. Calculations are performed from left to right, and only arithmetic data is allowed. Format is:

`@=calculation(;O)(;precision 'n') n=0 to 4`

Where:

`+` = addition

`-` = subtraction

`*` = multiplication

`/` = division

`@=6+7*"1.5";;2` means default equals the result of adding data from prompt '6' and '7', then multiplying the result by '1.5', and displayed using '2' decimal points.

All calculations are rounded to the precision selected after each calculation. If no precision is specified, then '4' decimal precision is used

### **Image Name Retrieval**

In the case image controls, the Defaults property is filled in with the name of the image resource that has been loaded into the prompt.

### **Last/Prior Entry Default**

This will default the last/prior data value, entered in the previous use of this prompt. If the default value is empty then the AllowChange parameter will be ignored. Thus a default of "@Last" will stop and allow input on the first pass and skip the field on all subsequent passes. The intended behavior of "@Last" is to maintain the last entered value until the application is exited.

Format is:

@LAST(;O)

### **Lists (Listing Choices) Default**

This default is used to list multiple specific choices, from which 1 item may be selected. Items selected appear in a 'List' on the device. Format is:

@list,heading,item,item,item,...

Where 'heading' is the column heading name to print.

@list,Colors,RED,WHITE,BLUE allows the user to select 1 of the 3 colors

### **Skip the Current Entry Default**

This default is used to conditionally skip the current prompt. Format is:

@SKIP,exp1,exp2,exp3

Conditional Operators (exp1,exp2,exp3) are:

exp1 = the prompt number

exp2 = a conditional operator from 1 of the following:

- = for an equal to comparison
- # for a not equal to comparison
- > for greater than
- < for less than
- M for a matches comparison
- nM for a not matches test
- I: exp4 exists in exp2 (same as NC)
- C: exp2 exists in exp4

NC: exp4 exists in exp2

`exp3` = the prompt number or string value (in quotes) to match on.

`@SKIP,2,='N'` means to skip the current prompt if the data entered at prompt '2' equals 'N'.

*Note in this example that `exp1="2"`, `exp2="="`, and `exp3="N"`*

## **Set Default**

This default is used to validate data before inserting the specified default value. Format is:

`@Set,exp1,exp2,exp3,exp4`

Conditional Operators (`exp1,exp2,exp3,exp4`) are:

`exp1` = the value that will become the default

`exp2` = the parameter to be compared to `exp4`. This can be a literal, a prompt number, or the RSP variable.

`exp3` = a conditional operator from 1 of the following:

- = for an equal to comparison
- # for a not equal to comparison
- > for greater than
- < for less than
- M for a matches comparison
- nM for a not matches test
- I: `exp4` exists in `exp2` (same as NC)
- C: `exp2` exists in `exp4`

NC: `exp4` exists in `exp2`

`exp4` = the second parameter to be compared to `exp2`. This parameter can be a literal or an edit such as NUM and ALPHA.

`@Set,'Hello',2,='100260'`

This sets the default to 'Hello' if prompt 2 is equal to the string value 100620.

`@Set,'Hello','XYZ',#,ALPHA`

This sets the default to 'Hello' if the string literal XYZ is not an ALPHA string. In this case, the default would never be used.

```
@Set,'Hello','3',C,'12345'
```

This sets the default to 'Hello' if the second parameter (3) is contained within the forth parameter (12345).

### **SQL Statement (List) Default**

This default is used to develop multiple selection items from your database so that 1 may be selected. Format is:

```
@sql, statement
```

(where statement is a valid SQL select statement)

The heading that appears is the database column name.

@sql, select PONum from OpenPO would select the purchase orders numbers column, from open purchase orders table 'OpenPO'.

A List, when used, clears the existing screen and lists selected data items (vertically) on the display screen. The RFgen 'List' display feature is a 'full screen display'. The user then selects 1 of the items in the list. There are 3 examples of list creation which follow.

1. For predefined/known list box items use an '@list' default:

```
@list,heading,item.1,item.2,item.3,...,item.n
```

Here: '@list,' indicates to RFgen that the statement is a 'default' parameter 'heading' is a name for the list that prints at the top of the box when it appears on a device 'item.1, item.2,...' are a listing of selection items to appear in the list.

e.g. - @list,Colors:,RED,WHITE,BLUE means to display a list with a heading of 'Colors:', and 3 selection items RED, WHITE, and BLUE to appear in the list.

2. For selection items which come from a database table, use an '@sql' default:

An SQL statement as an application 'default property' (for the prompt that will use the list) may be used to select the necessary data from your database, as illustrated here:

```
@sql,select fieldname1 from tablename where fieldname2 ='%n'
```

Here: '@sql' indicates to RFgen that the statement is a 'default' parameter

'fieldname1' is the name of the table field/column to be displayed

'tablename' is the name of a database table which contains the fields/columns

'where fieldname2=' allows selection of certain data only

'%n' indicates data entered at prompt 'n' will be used for selection purposes.

@sql,select OrderNO from PObooks where PartNO ='%1' means to create a list box of order numbers from table 'PObooks' where the part number was entered at prompt number 1.

3. Alternatively, for database items, use a VBA script, e.g.:

```
Public Sub FieldName_GotFocus(Rsp As String,  
AllowChange As Boolean)  
    Dim SQL As String  
    SQL = "Select fieldname1 from tablename where  
          fieldname2 = '%n'"  
    Rsp = DB.MakeList(SQL)  
    AllowChange = True  
End Sub
```

Here, the RFgen VBA extension 'DB.MakeList' is used in a GotFocus event to make a list from the results of the specified SQL statement. The results are passed to a prompt default variable named 'Rsp'. A variable called AllowChange is set to True, meaning override allowed.

## User Default

This default will put in the login name of the user. Format is:  
@User

## Edit Property Details

Validations/edit checks are available to test if data meets a certain criteria (e.g., the data entered is numeric). A validation property is available for each prompt and validations are entered in the Properties window. Multiple validations/edits are allowed. Select the edit property, and select the 'Ellipse' button. A multiple line input box will display. Enter one edit per line. *Note that as soon as an entry passes an edit, any subsequent edits are ignored.*

If an entry has no validation criteria, leave its edit property blank.

### **Text (Character String) Validation**

This will test for an exact match between the data entered and the text edits required. A text edit may be entered either with or without quotes. Format is:

'string' (in single quotes)

'CA' means the user must enter CA to pass this edit test

### **Pattern Match Validation**

This will test the data entered to see if it matches the pattern specified. Format is:

xA or xN

For example, 2A means that the data entered must have the format 'AA' (i.e., match 2 alphabetic characters, for example: 'OR'). 4N means the data entered must be in the form 'NNNN' (i.e., 4 numbers; for example: '1234').

4N means the user must enter a valid 4-digit number.

### **Pattern Not Allowed Validation**

This will test the data entered to assure that it does not match the pattern specified. Format is:

#PATTERN

#4N means the user may enter anything but 4 numbers (i.e., not 4 numeric characters)

#Hello means the user may not enter Hello in the prompt.

### **Alpha Only Validation**

This will test the data entered to assure that it only contains alpha characters (A-Z and a-z). Note: no spaces are allowed. Format is:

ALPHA

### **Numeric Only Validation**

This will test the data entered to assure that it only contains numeric characters (0-9), and the special characters '.' (period), '+' (plus), and '-' (dash). Note: no spaces are allowed. Format is:

NUM

### **Integer Only Validation**

This will test the data entered to assure that it only contains an integer number (no decimal). Note: no spaces are allowed. Format is:

INT

### **Greater Than Validation**

This will test the data entered to see if it is greater than a specified value. Format is:

>number or >=number

>99 means that the user must enter a number greater than 99.

### **Less Than Validation**

This will test the data entered to see if it is less than a specified value. Format is:

<number or <=number

<99 means that the user must enter a number less than 99

### **Not Equal To Validation**

This will test the data entered to assure that it does not equal a specified value. This is a string comparison only. Format is:

#'stringvalue'

#'99' means that the user must enter a string that does not equal '99'. Since prompts accept data as strings by default this validation would work with numbers while treating them like strings.

### **Data Range Validation**

This will test the data entered to see if it is within a specified range. This is a numeric comparison only. Format is: RG/exp1,exp2

Where:

exp1 = starting number for range.

exp2 = ending number for range.

RG/1,100 means that the user must enter a number between 1 and 100, inclusive

### **Date Validation**

This will test the data entered to see if it is a proper date. Format is:

DATE

Entering MMDDYY, MMDDYYYY, MM-DD-YY, MM/DD/YY among others are all valid entries.

### **Index Validation**

This will test the data entered to see if it can be found within the edit string. Format is:

I:string

I:YN means data must be either 'Y' or 'N', or 'YN'.

### **Contains String Validation**

This will test the data entered to see if it contains the specified edit string. Format is:

C:string

C:abc means data entered must contain the characters 'abc' somewhere within it

### **Not Condition Validation**

This will test the data entered to see if it does not match a given condition. Format is:

NC,exp1,exp2,exp3

Conditional Operators (exp1,exp2,exp3) are:

exp1 = the parameter to be compared to exp3. This can be a literal (in quotes, even if numeric, eg '3'), a prompt number, or the RSP variable.

exp2 = a conditional operator from 1 of the following:

- = for an equal to comparison
- # for a not equal to comparison
- > for greater than
- < for less than
- M for a matches comparison
- nM for a not matches test
- I: exp3 exists in exp1 (same as NC)
- C: exp1 exists in exp3

NC: exp3 exists in exp1

exp3 = the second parameter to be compared to exp1. This parameter can be a literal or an edit such as NUM and ALPHA.

NC,2,=,'100260'

This checks prompt 2 to see if it is equal to the string value 100620. If it is NOT, the edit is satisfied.

NC,'XYZ',#,ALPHA

This compares if the string literal XYZ is not an ALPHA string. Because XYZ IS an alpha, this edit is satisfied.

NC,'3',C,'12345'

This edit is NOT satisfied because the first parameter (3) is contained within the third parameter (12345).

### **Branch/Goto Validation**

Technically, this is not a validation. Allows the display to Goto a 'downstream' prompt, based upon the data 'RSP' entered. Format is:

`@GOTO,prompt#,conditions)`

`@GOTO,5,RSP,=,"99"` means to go to prompt 5 if the response at the current prompt equals "99". See the `@SKIP` default parameters for examples of 'conditions'

### **Strip (Data Entry) Validations**

Validates that 'str' is there, then strips data 'str' from the entry. Format is:

`@STRIP,P,str` to strip data prefix 'str' if it occurs

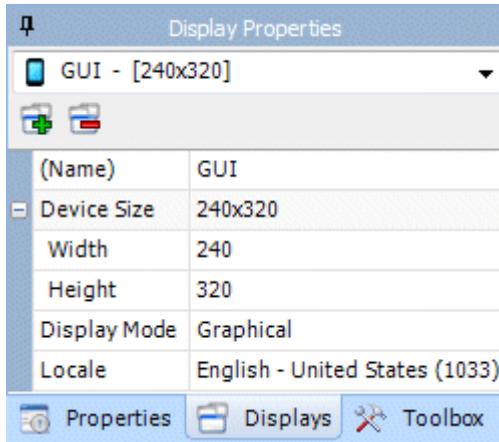
`@STRIP,S,str` to strip data suffix 'str' if it occurs

`@STRIP,C,str` to strip the first occurrence of the data 'str' if it is contained in the data.

`@Strip,P,NBR:` means to strip the character string 'NBR:' if it prefixes the data.

## **Displays**

The Displays tab allows the user to create variations of the display for the same transaction.



The drop-down box lists all the displays already created. To add an additional display or remove the current display, click on the plus and minus icons.

The **(Name)** is the display that can be referenced in the script to call up the custom display based on any given criteria like user profile properties or IP address. `App.SetDisplay` is the script command.

The **Device Size** width and height are properties of this display that can be used by the server to automatically choose this display based on the type of device connecting. When operating in Character Mode, remote devices use telnet to communicate with the Server and the size is dictated by columns and rows; in Graphical Mode, remote devices use the RFgen 'windows/graphical client' to communicate with the Server and the size is referenced in pixels.

The **Display Mode** property is only for the design time creation of the application. If at run time the user is using a graphical device and calls an application designed in character mode, it will still render as a graphical application. RFgen supports both 'character/text' and 'graphical' data entry applications. Consider the following display differences for the Transfer application:

```

Inventory Transfer
Plant:3000
Sacramento
Item#:100620
Office Chair
Locn:0001
Dock Area

Max Qty:6000
Qty:10
To Plant:3000
Sacramento
To Locn:0002
Shipping
Enter to accept...

```

Character Mode

**Inventory Transfer**

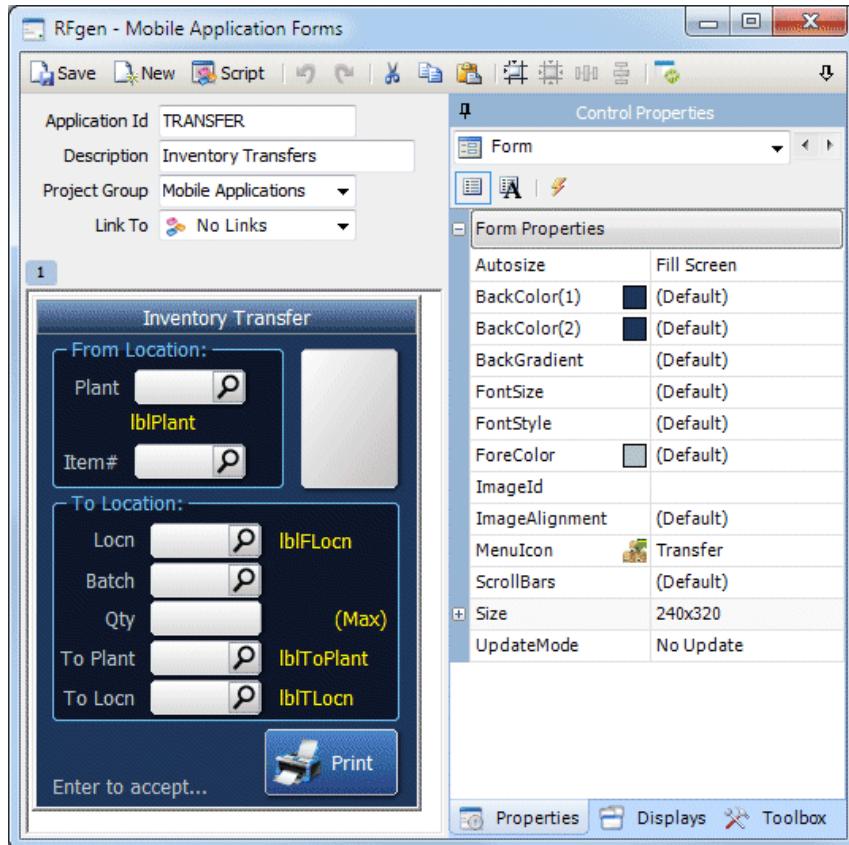
From Location:	Plant 3000 Sacramento	
	Item# 100620	
To Location:	Locn 0001 Dock Area	
	Qty 10 5990 (Max)	
	To Plant 3000 Sacramento	
	To Locn 0002 Shipping	
	 Print	
	Enter to accept...	

Graphical Mode

The **Locale** property is also used to determine if a specific display should be used over another. The server receives the locale of the device connecting to the server and can present the appropriate display for that device type.

### The Application Construction Area

This is where the application is constructed. Multiple pages can be constructed if an application is to have more than one page to complete a transaction. This is only the case if there are more prompts than will fit within the confines of the mobile device's screen.



To build a data collection application, drag-and-drop prompts from the Toolbox into the display area (or double-click on the objects and they will be automatically transferred to the display), in the order in which prompts will appear. Note that a data field and its associated Caption (prompt) are addressable separately.

The upper left-hand corner of the prompt display area is 0,0; the next line down is 0,1 (Column 0, Row 1), and so forth. In graphical mode the same applies for pixels.

There are several options for moving and resizing a control. To move a control, position your mouse on the desired control and select it with the left mouse button. You can use the arrow keys or the mouse to move the control to a new location. Holding down the Shift key will move the control one pixel at a time. To resize a control first select it and grab the edge of the control and drag in a direction. Again holding down Shift will resize one pixel at a time.

Holding down <ctrl> while clicking in multiple prompts will select all of them and make moving several prompts easier. Dragging a square around all the prompts to be selected is another method for quickly selecting multiple prompts.

Notice the double arrows located on the Properties tab.  They may be used to select each prompt in the order that they were placed. Click on the left or right arrow to select a different prompt. The combo box provides the same feature in a list format.

To insert a prompt, add the prompt as usual and then change the order on the Properties tab using the Prompt Number property. Another option is to select an existing prompt on the screen that you want the new prompt to follow. Then double-click the new prompt from the Toolbox and it will be inserted in the prompt sequence. To delete a prompt, right-click on the prompt and click on Cut from the popup menu. The prompt will disappear.

To reorder prompts, change the 'Prompt Number' property on the Properties tab for each prompt.

## Scripts – Menu Bar

When the script window is opened, the menu bar at the top of the window provides some standard and some custom features.



The **Item** label displays which application is being edited. The **Save** option saves the current script to the solution database. The **Syntax** option performs a syntax check of the script. Also provided are **Undo**, **Redo**, **Cut**, **Copy** and **Paste**. The **Comment** and **Un-Comment** options allow quick removal or addition of code blocks.

The **Find** icon allows the user to find text or replace text within the current application or all applications and macros at the same time.

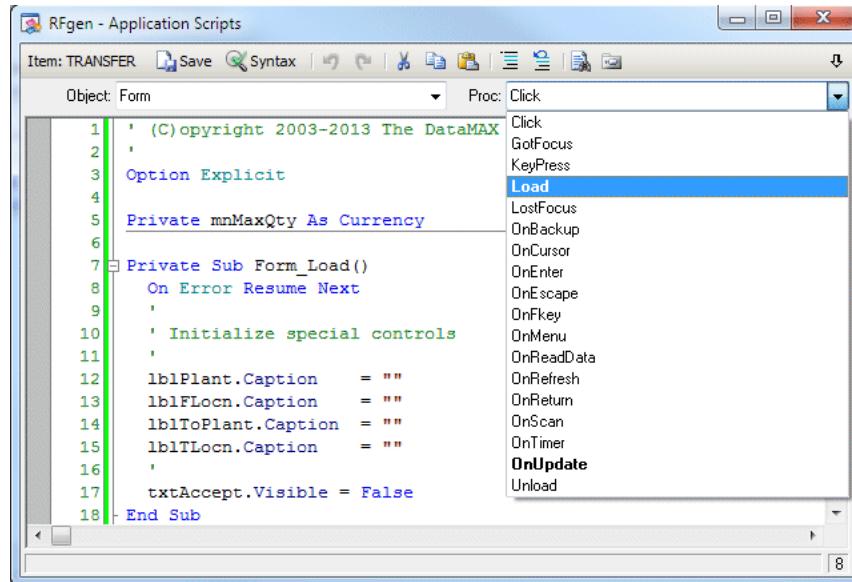
The **Add Module** specific dependencies button allows for non-globally loaded VBA Modules to be associated to the application. This window displays existing VBA modules. (See the VBA Modules tree) Check those modules you wish to include with the current application. Modules designated Win32.bas and RFgen.bas are automatically included for each application.

The **Dock** icon, (the down arrow) displays on every un-docked window, allowing the user to re-dock back to the Mobile Development Studio main window. Double-clicking the top-most tab will undock the window.

Scripting windows also provide some keyboard shortcuts that may make moving around in the environment easier. Ctrl + A will highlight all the text in the window. Ctrl + G will request a line number and then move the focus to that location. Ctrl + Spacebar will pop up a search window for language extensions, embedded procedure parameters and other similar uses.

## Scripts – Global, Application and Prompt Events

When the user clicks on the Script menu option a scripting window similar to the example shown below will appear.



The screenshot shows the RFgen application window titled "RFgen - Application Scripts". The menu bar includes "File", "Edit", "View", "Script", "Save", "Syntax", and "Help". The toolbar contains icons for Save, Syntax, Undo, Redo, Find, Replace, Copy, Paste, and others. The main area has a "Object: Form" dropdown and a "Proc:" dropdown showing a list of events. The code editor contains VBA code:

```
1 ' (C)opyright 2003-2013 The DataMAX
2 '
3 Option Explicit
4 '
5 Private mnMaxQty As Currency
6 '
7 Private Sub Form_Load()
8     On Error Resume Next
9     '
10    ' Initialize special controls
11    '
12    lblPlant.Caption = ""
13    lblFLozn.Caption = ""
14    lblToPlant.Caption = ""
15    lblTLocn.Caption = ""
16    '
17    txtAccept.Visible = False
18 End Sub
```

The "Proc:" dropdown is open, showing a list of events:

- Click
- GetFocus
- KeyPress
- Load**
- LostFocus
- OnBackup
- OnCursor
- OnEnter
- OnEscape
- OnFkey
- OnMenu
- OnReadData
- OnRefresh
- OnReturn
- OnScan
- OnTimer
- OnUpdate**
- Unload

When using VBA, note that each prompt on the application, and the application itself, may have associated scripting for the numerous associated events. All possible VBA events are:

**Click** occurs when the user clicks on a prompt or button with a mouse or pen

**GotFocus** occurs whenever the user enters a prompt. First, the application level executes and then, the prompt level executes

Keypress	occurs when the user presses and releases a key on the keyboard
Initialize	occurs when an RFgen client is initially loaded. It will occur only once, and is typically used to initialize variables, open additional database connections or create links to ActiveX objects. (RFgen.bas only)
InRange	occurs when a Mobile Client notices that there is an IP address available on the network adapter. This event does not execute when using the Mobile Client in a Roaming state because the OnConnect will execute anyway. (RFgen.bas only)
Load	occurs once when the application is initially loaded
LostFocus	occurs whenever the user leaves a prompt. The prompt level executes first and then the application level executes
OnBackup	occurs when the up arrow was pressed to move to the previous prompt
OnConnect	occurs when the Mobile Client in the Roaming state automatically discovers it has connectivity to a network and makes a connection to the RFgen server. The event only executes after 10 seconds of continuous connectivity to the RFgen server. To maintain data integrity, it may be a good idea to include the Server.SyncApps and Server.SendQueue in this event. (RFgen.bas only)
OnCursor	occurs whenever one of the arrow / cursor keys are pressed
OnDisconnect	occurs when the Mobile Client in a Roaming State is disconnected from the RFgen server because the client moved out range. This event does not execute when the Server.Disconnect command is issued. (RFgen.bas only)

OnEnter	occurs whenever the user exits a prompt by the Enter key, or the AllowChange variable in the GotFocus event is set to False
OnEscape	occurs whenever the Escape key is pressed
OnFkey	occurs whenever a function key is pressed
OnLocale	occurs after OnConnect (only when a client application makes a connection) and passes in the locale number based on the client device's location (RFgen.bas only)
OnMenu	occurs when the user clicks on an item in the embedded graphical menu. This event exists at the global, application and prompt levels.
OnReadData	occurs whenever data is successfully retrieved from a database by an RFgen application that is linked to a table (application level only)
OnRefresh	occurs when the screen is completely repainted (as when the user presses the F2 key). It is typically used to redisplay information to the user that was displayed via Screen.Print statements (application level only)
OnReturn	occurs when the user returns from a called application that had its return flag set to True. It is typically used to process data from the called application (application level only)
OnRowColChange	occurs when the focus moves between a column or row in a control that supports them.
OnScan	occurs when RFgen identifies a pre-amble or post-amble string in the data stream coming from a client device. This event is typically used to validate that data was scanned, or to parse the data (PDF, RFID, etc.) into a more usable format.
OnSearch	occurs when a user clicks on the search icon created at the end of a textbox. There is an icon to click only if there is code in this event. (field level only)

OnSpeech	Using the language extension TTS.Listen, RFgen will detect that speech has occurred and execute this event. (Also exists in RFgen.bas which executes first, then the form level and finally the prompt level)
OnTimer	occurs at a user-defined interval when it is enabled. This event is used to perform some action based upon defined time interval (application level only)
OnUpdate	occurs after the last prompt has been entered and just prior to RFgen updating the database (application level only)
OnVocollect	a specialty event that is used with Vocollect tasks embedded on an application that communicates to and from Vocollect hardware and RFgen (field level only)
OutOfRange	occurs when a Mobile Client notices that the IP address is no longer available on the network adapter. This event does not execute when using the Mobile Client in a Roaming state because the OnDisconnect will execute anyway. (RFgen.bas only)
Terminate	occurs when an RFgen client is terminating. It will occur only once, and is typically used to close manually opened database connections or release links to ActiveX objects. (RFgen.bas only)
UnLoad	occurs once when the application is exited (application level only)

Application events take precedence over prompt events; e.g., events linked to the application will occur prior to the event firing for a prompt.

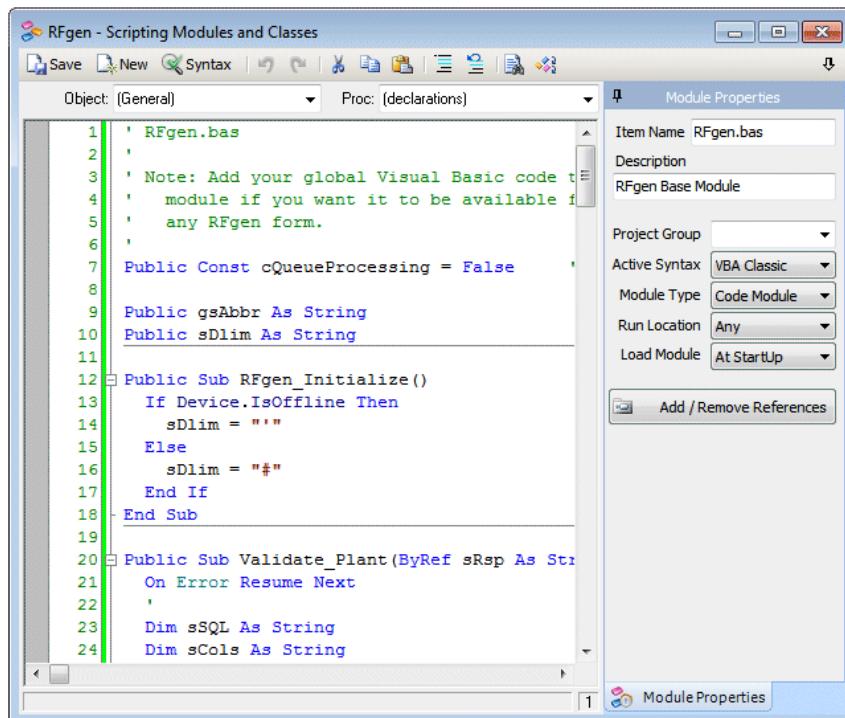
Note: VBA code is very sensitive to variable typing. Most errors result from using a String variable when an Integer or Long data type is required.

For complete information on VBA scripting, see Visual Basic Scripts, and the RFgen documentation accessible via the Start \ Programs \ RFgen v5 \ Documentation menu.

## VBA Modules List

The VBA Modules contain global definitions of variables, functions, and procedures that can be referenced by any application or macro script. The default installation provides two “Code” BAS modules: Win32 and RFgen.

The Win32.bas module provides a place for global variables and procedures that pertain to system level requirements. The RFgen.bas module provides a place for global variables and procedures that pertain to operation of the transactions. Both are always available for all applications and macros and are functionally identical. The RFgen.bas module does contain an RFgen object with its own events as described above.



## VBA Module Editing Menu Bar

The **Save** option saves the current script to the solution database. The **New** option clears all fields and provides a blank window for creating a new bas module. The **Syntax** option performs a syntax check of the script. Also provided are **Undo**, **Redo**, **Cut**, **Copy** and **Paste**. The

**Comment** and **Un-Comment** options allow quick removal or addition of code blocks. The **Find** icon allows the user to find text or replace text within the current application or all applications and macros at the same time. The **Module** icon opens the Visual Basic Environment Configuration screen and lets the user prioritize the load sequence of the BAS files. This is important if some BAS files rely on other BAS files for declared variables or functions.

The **Dock** icon, (the down arrow) displays on every un-docked window allowing the user to re-dock back to the Mobile Development Studio main window. Double-clicking the top-most tab will undock the window.

### VBA Module Editing Window Fields

The **Item Name** is the RFgen identifier for the bas module. Typing in an existing name and exiting the field will check if that name is already used and load that module, if it is found. The **Description** is listed in the Design tree for clarification of the modules purpose.

The **Project Group** property allows the modules to be grouped together into categories such as “Mobile” or “ThinClient” or any other named grouping. This has no impact on production. It is just an option for making the management of the development database easier.

**Active Syntax** specifies what kind of module is being created. A VBA Classic BAS file is treated as a standard module that can be referenced. A .Net module allows for Global Assembly Cache (GAC) references to classes created outside of RFgen. This option is not compatible with a Mobile installation because the device will not have access to these classes. Microsoft .Net Framework 4.0 is supported.

The **Module Type** option will create the BAS file to be either a Code, Class or Object module. A Code module is the standard module that can be referenced as before. The Class module with public and private functions can be created and then referenced in the application’s code. For example a Class module called Math has one public function called Add and one private function called Subtract then in the application script:

```
Dim MyClass As New Math  
MyClass.Add()
```

The Subtract function is not available because it is private. Because it is a class you may create multiple instances of the class as needed. The Object module type is similar but RFgen will create the first instance

automatically. Using the same example the Dim statement would not be required. Instead all that is needed is:

```
Math.Add()
```

In this case the initial object instance is used. If more are required then you would use the same example as the Class.

The **Run Location** refers to the mode of the client. The module will either be available on both the server and the device (in the case of a Mobile implementation), just the server (in case there is code that will not be compatible in the CE / Android / iOS environment) or just the device (in case the code is specific to the mobile environment. For example if there was a need to have the same global function work in two different ways depending on if the device was in a mobile mode or thin client mode, two different BAS files could be created with identical function names, one set to Server and the other set to Device. Then if the device was online the Server version would be used and if the device went offline the Device version would be used and a global syntax check of the solution would not see a conflict.

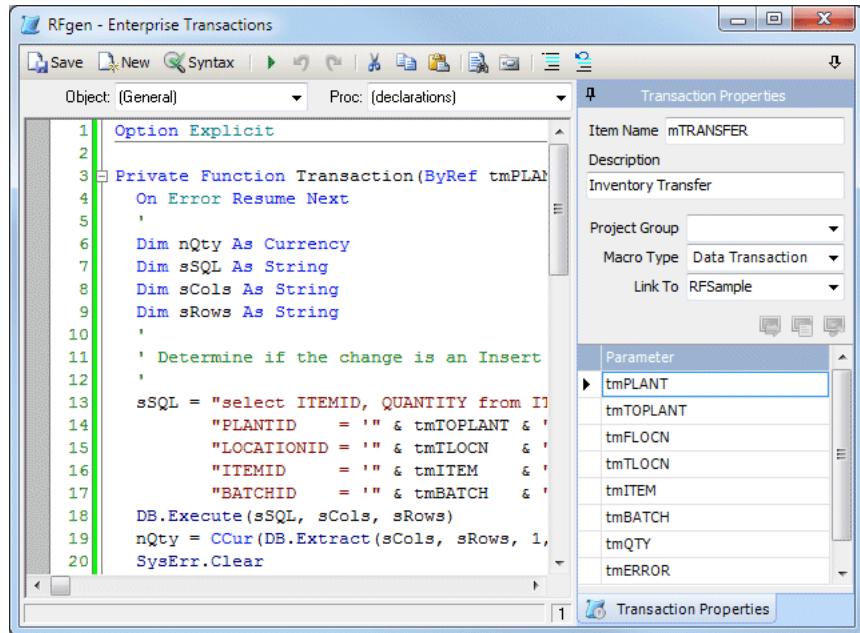
The **Load Module** option makes the BAS file either globally loaded and available all the time or just another module that must be referenced from each application that needs to use its functions.

**Add / Remove References** allows the programmer to add Global Assembly Cache (GAC) references to the VBA module such as a custom class created outside of RFgen. This is very useful for .Net module types.

In the code environment the Object drop-down box will contain the RFgen object for the RFgen.bas module only. For all other modules, it will remain empty. The Procedure drop-down box will contain any user-created function or procedure names. In the case of the RFgen object (as shown above) the user will have access to the internal events.

## Transactions List

The transaction design window has three purposes: to create and edit Timed Event macros, Host Transaction and Data Transaction macros.



A Timed Event macro is a macro that runs on a timer configured under the Configuration / Transaction Management / Transaction Management Events dialog. If you want some script to run without a user being present, create a Timed Event macro and enable the Transaction Management capabilities. There are no passing parameters for Timed Event macros.

A Data Transaction macro is a script that can accept parameters passed in and out and can execute in a queued or non-queued manner. You would create a Data Transaction macro: if multiple applications could take advantage of the same process, you need a history of the data being processed kept by the Transaction Manager, applications are run in a Mobile environment and later uploaded to the server for processing or queuing is implemented for all applications.

To create a Data Transaction macro, follow these steps:

1. Enter an **Item Name**, **Description**, optionally a **Project Group** that lets the user group like items together in the navigation tree, set the **Macro Type** to Data Transaction and set the **Link To** option to the appropriate data connection.
2. Add parameters for each value being passed in to the macro. The Location and Length columns do not apply to Data Transaction macros. A maximum of 31 can be used due to the integration with

the VBA environment. To work around this you may concatenate multiple values separated by a unique string like “|” and only use 1 parameter.

3. Select the Transaction function from the Procedure drop-down and a shell function will be created for you.

A Host Transaction macro can be created and linked to a Host Screen macro. This is described in the Screen Mapping section.

### Transaction Module Editing Menu Bar



The **Save** option saves the current script to the solution database. The **New** option clears all fields and provides a blank window for creating a new transaction macro. The **Syntax** option performs a syntax check of the script. The **Play** button is used to test Screen Mapping host Transaction macros to be sure the keyboard recording is correct. Also provided are **Undo**, **Redo**, **Cut**, **Copy** and **Paste**.

The **Find** icon allows the user to find text or replace text within the current application.

To allow references to global scripts that are in a VBA module but are not globally available select the **References** menu item. This window displays existing VBA modules. (See the VBA Modules tree) Check those modules you wish to include with the current macro. Modules designated Win32.bas and RFgen.bas are automatically included for each macro.

The **Comment** and **Un-Comment** options allow quick removal or addition of code blocks.

The **Dock** icon, (the down arrow) displays on every un-docked window allowing the user to re-dock back to the Mobile Development Studio main window. Right-clicking the top-most tab and selecting UnDock will undock the window.

### Hosts List

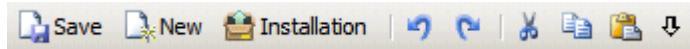
The Hosts List displays all screen mapping related macros. Please see the screen mapping section for more information.

# Resources Tab Overview

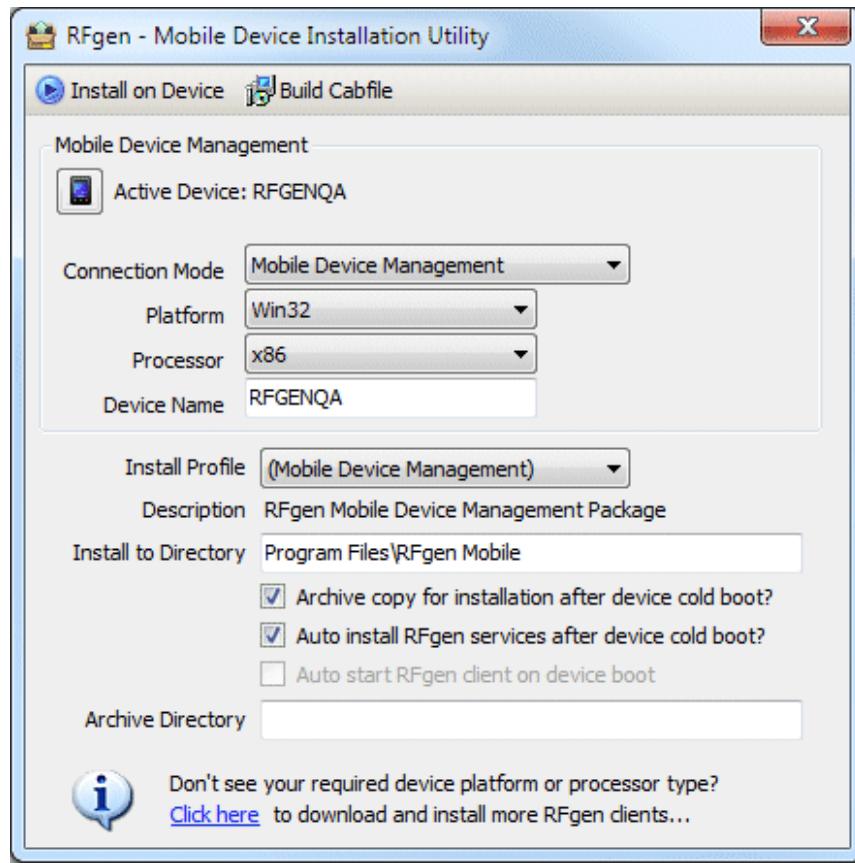
The Resources tab group together the items that make up the non-scripting portion of the solution. In this section, there are mobile device profiles, image resources, mobile theme configuration, text translations and Vocollect specific tasks that interface the Vocollect hardware devices to RFgen's data connectors.

## Device Profiles

The Device Profile Tree is used to prepare a mobile device for use in the wireless / wired environment or the mobile environment.

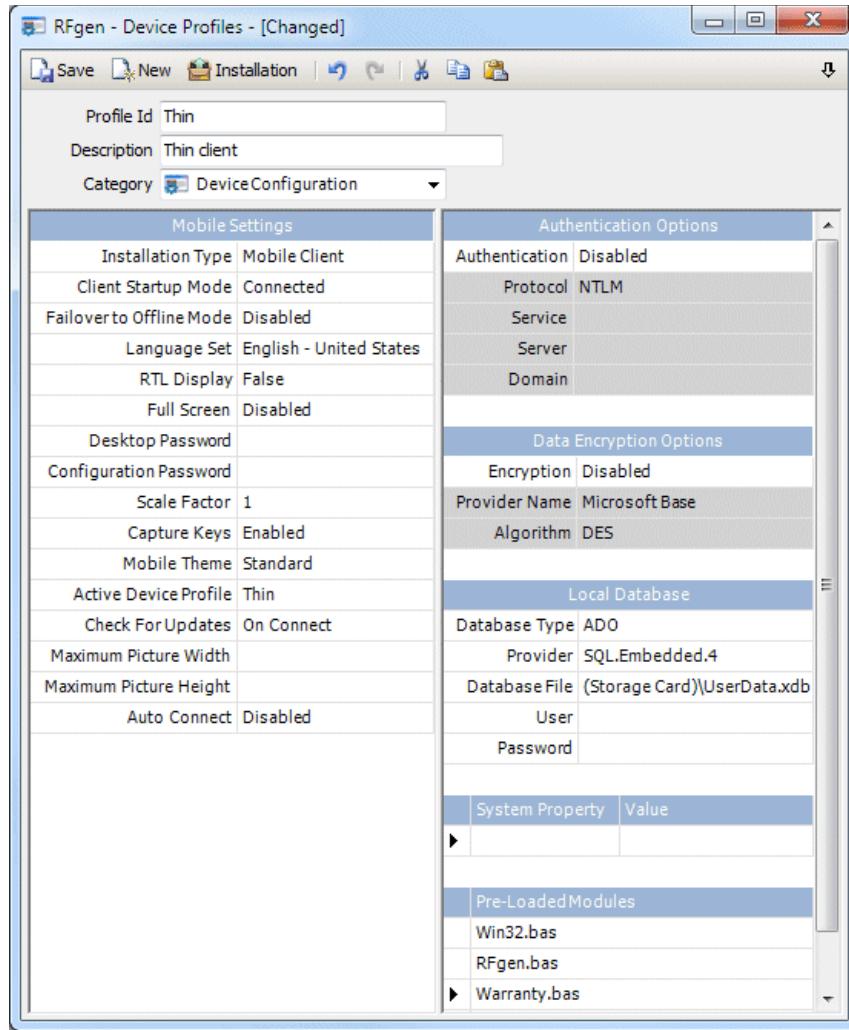


The **Save** option saves the current profile to the solution database. The **New** option clears all fields and provides a blank window for creating a new profile. The **Installation** brings up the MDM installation utility window.



This is the same window as found under the Devices menu option.

Also provided are **Undo**, **Redo**, **Cut**, **Copy** and **Paste**. The **Dock** icon, (the down arrow) displays on every un-docked window allowing the user to re-dock back to the Mobile Development Studio main window. Double-clicking the top-most tab will undock the window.



The **Profile ID** is the identifier of the configuration and gives the user the ability to open, edit and save a certain configuration to avoid recreating it for each mobile device. The **Description** explains the purpose of the profile and displays in the tree when saved.

## Category: Device Configuration

This category contains some basic setup information starting with the Client Settings Group.

For **Installation Type**, choose the type of client that will reside on the handheld. Your choices are Thin Client or Mobile Client.

**Thin Client** The traditional wireless real-time interface to the server where the mobile device is restricted to the RF environment.

**Mobile Client** This option allows the user to leave the RF environment and manually or automatically switch to and from a connected state. See the Mobile Devices Section for more detailed information.

**Client Startup Mode** refers to which state the mobile client will use when started, connected, disconnected or roaming. The Thin Client option will only have the 'Connected' choice.

**Failover to Offline Mode** refers to the mobile client requesting to switch from thin client state to a disconnected state when the RF signal is lost. The user can deny this request and wait for RF coverage to return.

**Language Set** defaults to English. Any language may be chosen from this menu as long as the language is installed on the mobile device.

**RTL Display** is for displaying all elements on the form in a right-to-left format. Languages such as Arabic and Hebrew are examples.

**Full Screen** determines if the display on the mobile device is in a window (smaller) or if the application is maximized for the screen (larger.)

**Desktop Password** is designed to prevent unauthorized users of the mobile device from leaving the client and returning to the device desktop. Setting the Full Screen mode to Enabled is recommended.

**Configuration Password** is designed to prevent unauthorized use of the mobile device configuration screen where the local settings are kept. Swiping down from the top edge will open the configuration windows and with this option enabled with a password a password box will be displayed.

**Scale Factor** is designed to fix a problem on some devices where they do not report their DPI (dots per inch) correctly. If the high resolution screen forces the application screen into a very small space, change this value from 1 to 2 to double the rendering of the application screen.

**Capture Keys** if enabled, will transmit all keystrokes to the server and not the operating system. If the operating system has taken the F3 key

for example to control volume then it would not get sent to the client unless this option is enabled. Other devices use a button to scan a barcode. If this were enabled that button to scan may not function so this feature would need to be disabled. The use of the feature depends on the requirements of the device and application.

**Mobile Theme** is the theme resource to be used on the device. It contains all the look-and-feel display options.

**Active Device Profile** is the name of the profile to be used when syncing a mobile client.

**Check For Updates** has three options, None, On Connect and Daily. On Connect means that every time the client (thin or mobile) makes a connection to the server it will synchronize. For thin client just the theme and some images will be updated. For mobile mode all the solution objects for the specified profile will be exchanged.

**Maximum Picture Width** and **Maximum Picture Height** are used when taking advantage of the device's camera. If a picture is taken and its size is greater than the maximum allowed, the image will be resized down to the max width or height, whichever comes last so that the aspect ratio is not changed. The image will not be cropped or reshaped.

The **Auto Connect** property, when enabled, will attempt to connect to one of the servers by going through the server list automatically, attempting for five seconds before moving to the next one. If this option is not enabled then a list is presented to the user and they must choose the correct server. If only one server is configured (the typical scenario) then this option has no value.

The right pane is for all the optional settings that may be used by the client.

#### Authentication Options / Data Encryption Options

These options are taken from the configuration settings. It is best to leave these consistent with the server settings. All parameters are fully documented under the Configuration menu.

#### Local Database

**Database Type** lets the user select between SQLite and OLEDB database types for use in the mobile environment.

**Provider** specifies which provider is used on the mobile device. Although you can freely enter text in this property, it is pre-populated with the solution's suggested values.

**Database File** is the database path and file location on the mobile device where the file should be created. The file name by itself will be placed in the solution directory. Specifying a path such as APPS\RFSample.xdb will place the file in that location.

**User** if required to access this mobile database, is entered here. There is no relation to the authentication required for the server's databases.

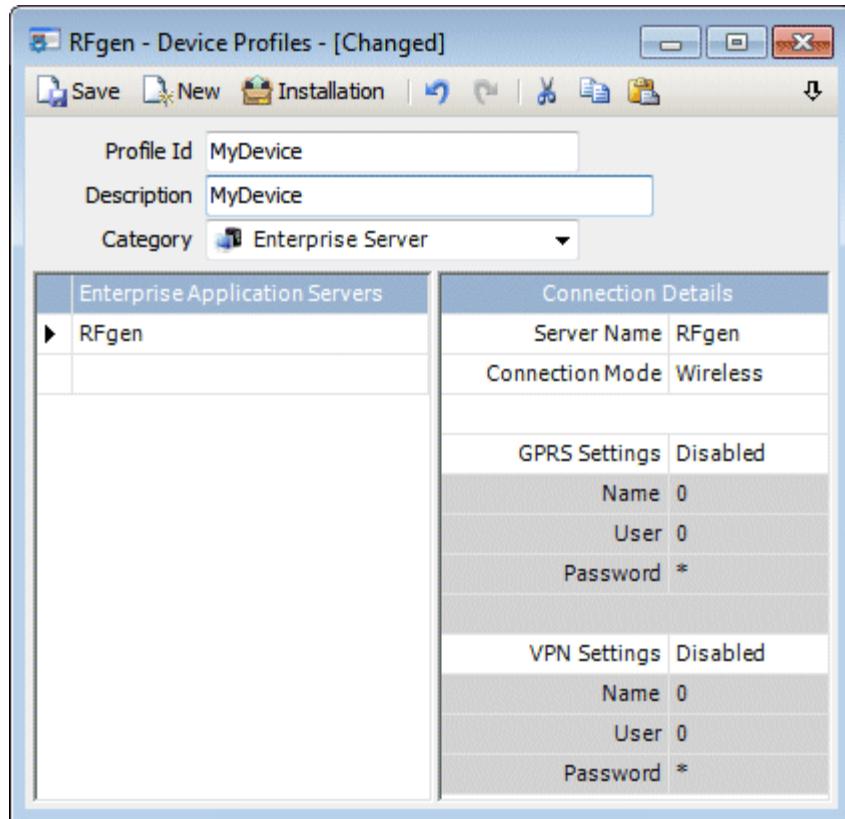
**Password** if required is entered here.

The **System Property** Ids and values are taken from the configuration settings. This list can be altered as needed for the mobile applications.

The **Pre-Loaded Modules** list is taken from the configuration settings. If they are not necessary or if supplemental BAS files are required for the mobile applications, this list can be changed as needed.

### **Category: Enterprise Server**

The Enterprise Server category records which servers the mobile device will have access to and what the connection type will be.



The option of multiple servers provides the mobile user a choice of which server they want to connect to. In the left pane simply list the name of each server. The Connection Details on the right will determine exactly how the client makes the connection.

The **Server Name** is reiterated and is changeable in either the left or right pane. The **Connection Mode** is either Wireless (which assumes the client is already on the private LAN or WAN) or Cellular. If cellular is chosen, then there are additional options for making a cellular connection. The client will auto-detect if WiFi is available and switch from cellular to WiFi if possible.

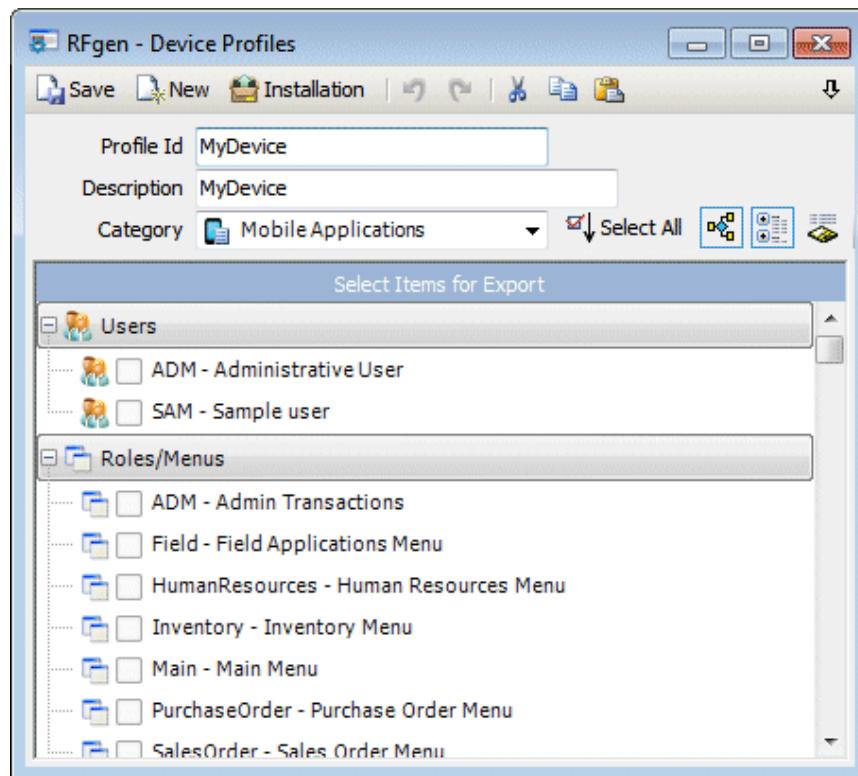
Depending on the platform of the mobile device, making a cellular connection can vary greatly. For example, if a VPN is configured on the device and it automatically starts the GPRS connection, then the client only needs to start the VPN, not both.

Note: The most effective way to determine what the client needs is to perform the connection manually first, to see what the working device configuration is and then, tell the client to start one or both connections by name.

**GPRS Settings** is either enabled or disabled. If one is configured on the device that the client must start itself, then fill in the Name of that connection. A **User** and **Password** may not be necessary. If the client must start the **VPN session** then reference it by **Name**. Again a **User** and **Password** may not be necessary.

## Category: Mobile Applications

The Mobile Applications category applies only for a Mobile mode client. This window allows the selection of the users, menus, applications, VBA modules, and table, macro and business function schemas to be transferred to the mobile device to be used in a disconnected state only.



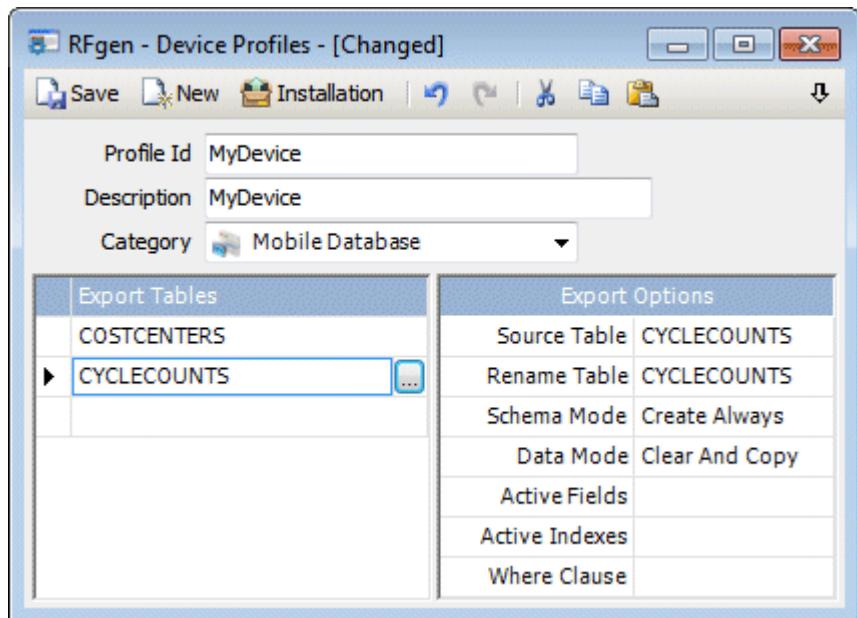
Select the required items or use the toolbar to help in the selection.



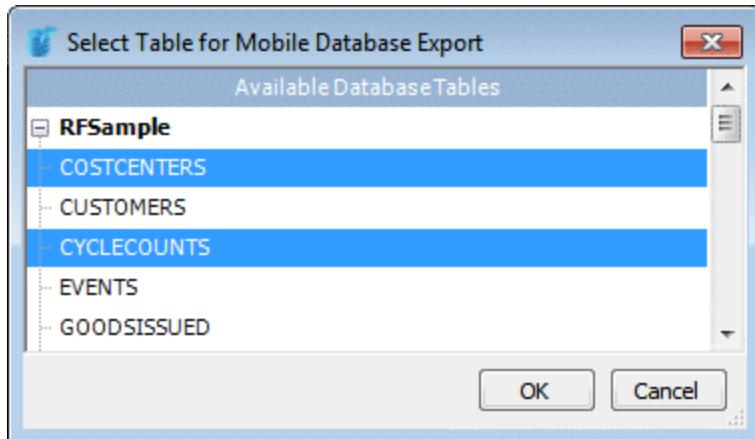
Select All obviously selects any visible item in the list. The next toolbar icon will automatically select other items that are required for the user-selected item to function. For example, selecting a user, all menus, applications, etc. will be selected automatically. The last 2 buttons toggle between displaying the whole list and only the selected items.

### Category: Mobile Database

The Mobile Database category applies only for a Mobile mode client. Here, you select which database tables from any source that is necessary for the mobile applications to function.



Select the export tables necessary and then configure each table individually.



You may select multiple tables for faster entry. Click on Select Table to return to the previous screen.

The Export Options allow the user to configure each table.

The **Source Table** is the name of the table in the database referenced through the connector.

The **Rename Table** property allows the user to change the name, as it will exist on the mobile device.

**Schema Mode** sets how the table will be created. There are three options: Data Only, Create Always (default) and Create on Change. Data Only means that RFgen will not create the table at all. It must exist in the database already. The Data Mode property will determine how the data is filled in. Create Always means the table will be created on the mobile database and Create on Change means that the existing table schema on the mobile database will be compared to the equivalent table schema on the server. If there is a difference, the existing table schema on the mobile database will be replaced with the updated table schema from the server.

The **Data Mode** property has 3 options: Clear and Copy (default), Copy Only and Schema Only. Clear and Copy will delete the contents of the table and re-populate it with the new data from the server. Copy Only will place the data from the server on top of the existing data. Where it is the same, there is no change. Where it is different and depending on the keys, either it will overwrite data or insert new records. Schema Only will always send an empty table to the mobile database.

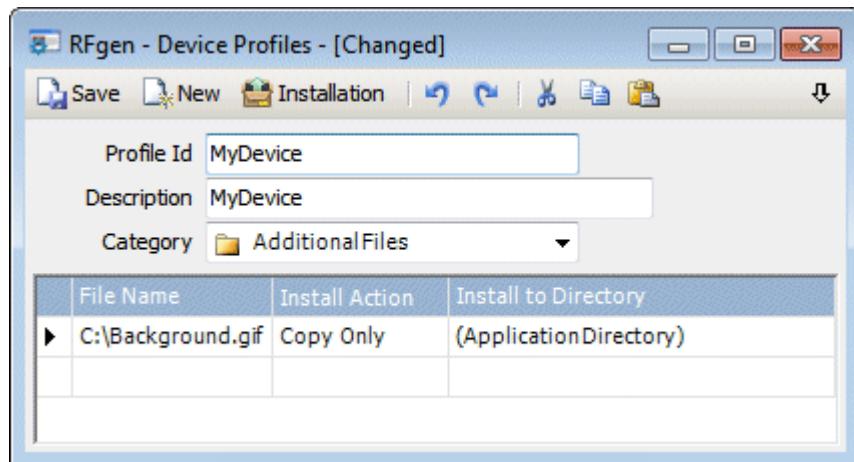
**Active Fields** are the fields that should be copied from the server database and used in the mobile database. This allows the user to have a smaller table structure, if the server's table contains fields that are not required for the mobile data collection effort. Leaving it blank assumes all the fields are necessary.

**Active Indexes** specifies the indexes on the mobile database for more efficient data retrieval.

The **Where Clause** specifies a subset of the data from the server's table in case the server contains more data than is necessary on the mobile database.

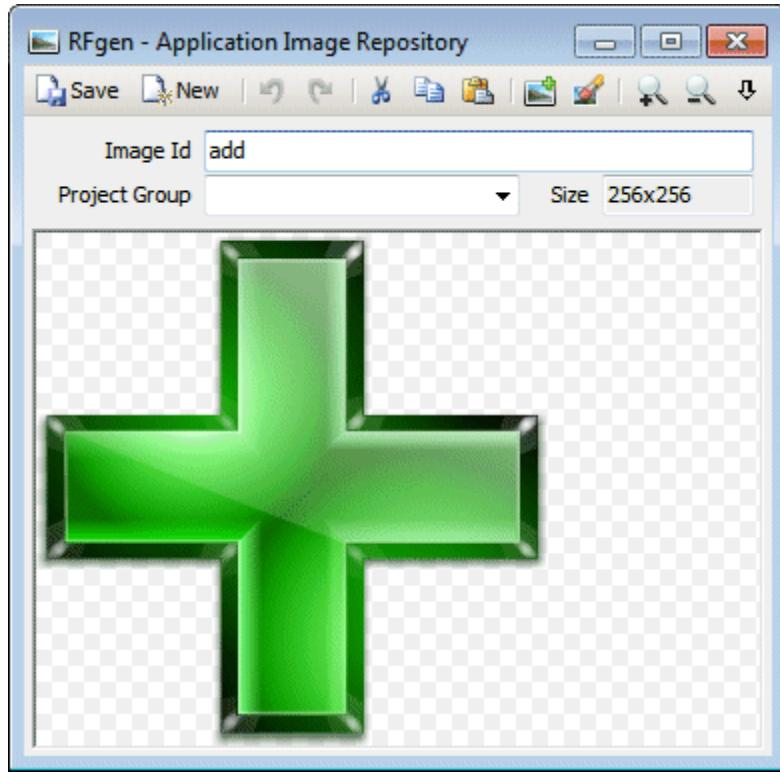
### Category: Additional Files

Additional files allow the user to copy other miscellaneous files to a specified directory and dictate whether they are registered or installed. The Install Action options are to copy the file or to copy and install the file. The Install to Directory drop-down offers a number of places the file can be placed.

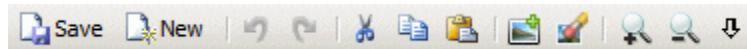


### Image Resources

The Image Resource provides for a collection of pictures of different formats to be stored inside the solution database. The images can be referenced by Image prompts at design-time or runtime or used as part of the configuration for mobile device backgrounds. This window allows the user to drag and drop an image for quick selection.



The **Image ID** is referenced from the GUI properties or from the code to extract this image from the application database. The **Project Group** can optionally be used to group similar image resources for better management.



The **Save** option performs a save of the image to the database. The **New** option clears the picture box and provides a blank window. Also provided are **Undo**, **Redo**, **Cut**, **Copy** and **Paste**.

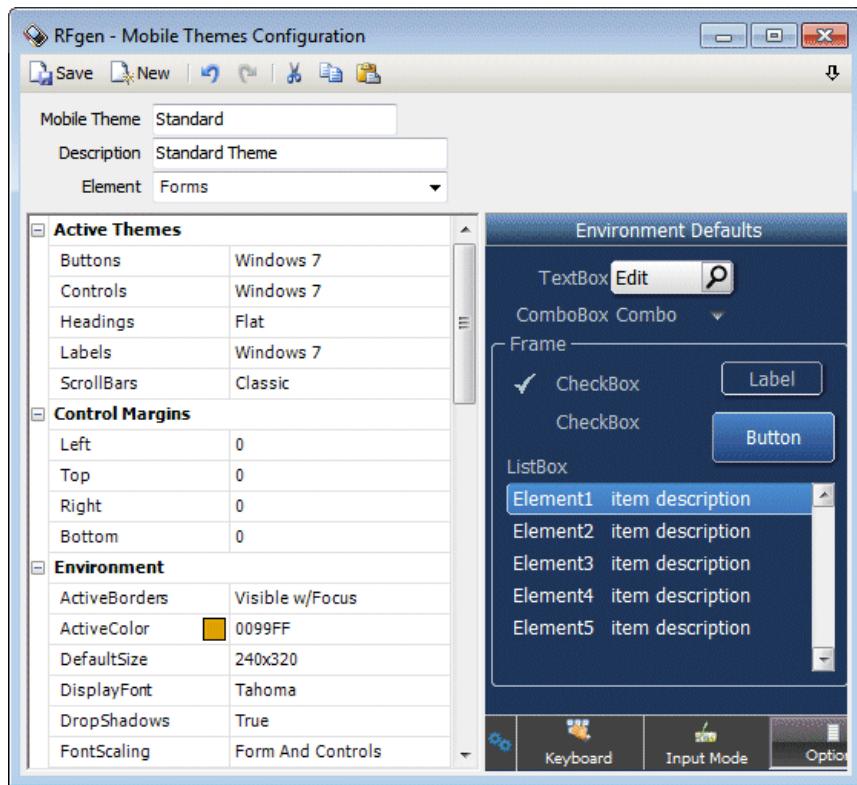
**Browse** for a new image opens the file explorer to select an existing image. **Erase** current image clears the display.

The plus and minus **magnification** buttons let the user zoom in and out but this does not change how the image will be rendered on the application screen.

The last icon, (the down arrow) displays on every un-docked window allowing the user to re-dock back to the Mobile Development Studio main window. Double-clicking the top-most tab will undock the window.

## Configuring Mobile Device Themes

The themes are broken into different elements. The first controls the look of the forms.



### Active Themes:

Buttons – specifies some default color schemes and the basic shapes of the borders on the button controls

Controls – specifies some default color schemes and the basic shapes of the borders on the prompts

Headings – specifies some default color schemes and the basic shapes of the borders on the title of the form

Labels – specifies some default color schemes and the basic shapes of the borders on the label controls

ScrollBars – specifies the theme of the form level scroll bars

### Control Margins

Left – the number of pixels of padding between the left side of a prompt and any other prompts or form edge

Top – the number of pixels of padding between the top side of a prompt and any other prompts or form edge

Right – the number of pixels of padding between the right side of a prompt and any other prompts or form edge

Bottom – the number of pixels of padding between the bottom side of a prompt and any other prompts or form edge

### Environment

ActiveBorders – in general, the editable prompts will either default to a standard, unchanging field, display an Active Border when it has the focus, No border at any time, appears standard only when it has focus or completely Transparent at all times.

ActiveColor – the color of the border of the prompt with focus if the ActiveBorders property is set to Active Border.

DefaultSize – the size in pixels for the shape of the form. There are default values that can be selected based on the typical size of the device but a custom value can also be entered. This value can be overwritten when designing the form itself.

DisplayFont Font Name – the default font for all applications using this theme. It applies to all prompts on this application screen and menus.

DropShadows – set to True or False, a drop shadow will appear when the prompt is displayed in the Standard, Active Border or Visible w/Focus modes.

FontScaling – controls the auto scaling of a form that stretches like in the Desktop Client. Designing a form for a handheld device and then displaying it on a tablet device for example, the form should scale up to

the larger size. The options are **Form And Controls**: will auto scale the form size and control size plus locations to adjust to the larger / smaller font in the theme; **Controls Only**: the client will not scale up the form size, but all control size plus locations will adjust to the new font size; **No Scaling**: no changes are made to the size of the application when the client size changes.

Form Body	
BackColor(1)	<input type="color"/> 5A351F
BackColor(2)	<input type="color"/> 5A351F
BackGradient	Solid
FontSize	10
FontStyle	Normal
ForeColor	<input type="color"/> CDC7B6
ImageAlignment	Disabled
ImageName	
Form Heading	
BackColor(1)	<input type="color"/> C5A377
BackColor(2)	<input type="color"/> 52311C
BackGradient	Vertical
FontSize	10
FontStyle	Normal
ForeColor	<input type="color"/> FFFFFF
Padding	0

### Form Body

BackColor(1) – the primary color of the background of an application screen depending on the BackGradient value

BackColor(2) – the secondary color of the background of an application only used when the BackGradient option is set to an option other than Solid. In this case the background will fade between the two colors.

BackGradient – set to Solid or a direction for the fade effect

FontSize – the default font size for all applications using this theme

FontStyle – the default font style for all applications using this theme such as Bold, Italic, Underline or a combination

ForeColor – the default prompt label fore color for all applications using this theme

ImageAlignment – If the application will have a background image like a watermark, this setting positions that image resource on the screen in one of many possible places.

ImageName – the name of the image resource to be used on the application background

### Form Heading

BackColor(1), BackColor(2), BackGradient, FontSize, FontStyle and ForeColor are applied the same as in other examples.

Padding – Adds padding to the heading on the search page

MenuStrip		
BackColor(1)	<input type="color"/>	424142
BackColor(2)	<input type="color"/>	424142
BackGradient		Solid
FontSize		8
FontStyle		Normal
ForeColor	<input type="color"/>	FFFFFF
Height		36
IconMode		Image + Text
LineColor	<input type="color"/>	C0C0C0
Visible		False
Width		75
MenuStrip Images		
Width		16
Height		16
MenuStrip Pressed		
BackColor(1)	<input type="color"/>	7A7A7A
BackColor(2)	<input type="color"/>	303030
BackGradient		Vertical Split
ForeColor	<input type="color"/>	FFFFFF

## MenuStrip

BackColor(1), BackColor(2), BackGradient, FontSize, FontStyle and ForeColor are applied the same as in other examples.

Height – the height of the Menu Strip buttons

IconMode – Options are Image + Text, Image Only and Text Only

LineColor – sets the line color surrounding the menu strip

Visible – Set to True will start sessions with the menu strip visible, False it will be invisible until the Show Menu (configurable in the F-Keys) command is used.

Width – sets the width of the individual buttons within the menu strip

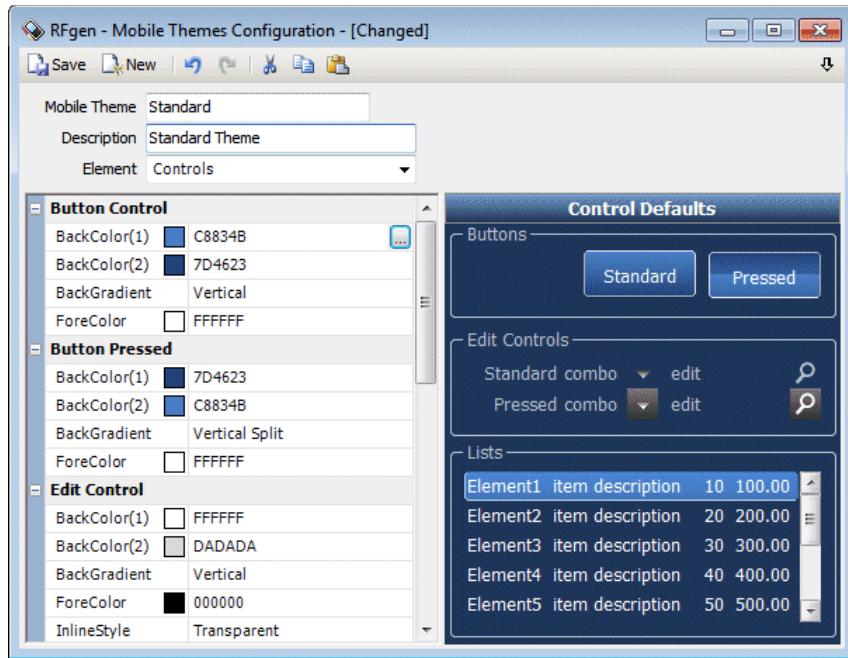
## MenuStrip Images

Width / Height – Sets the width and height of the images within the menu strip button

## MenuStrip Pressed

BackColor(1), BackColor(2), BackGradient and ForeColor are applied the same as in other examples but are used here to show the depressed state of the button

The second element is called controls and covers the look of specific prompt types.



### Button Control & Button Pressed

BackColor(1), BackColor(2), BackGradient and ForeColor are applied the same as in other examples.

### Edit Control

BackColor(1), BackColor(2), BackGradient and ForeColor are applied the same as in other examples.

InLineStyle – describes how the combobox or search buttons appear within the edit field. Options are Standard, No border and Transparent.

<b>Edit Inline Button</b>	
BackColor(1)	<input type="color"/> FFFFFF
BackColor(2)	<input type="color"/> C0C0C0
BackGradient	Vertical
ForeColor	<input type="color"/> 000000
<b>Edit Inline Pressed</b>	
BackColor(1)	<input type="color"/> 7A7A7A
BackColor(2)	<input type="color"/> 303030
BackGradient	Vertical
ForeColor	<input type="color"/> FFFFFFFF
<b>List Control</b>	
BackColor(1)	<input type="color"/> 28180D
BackColor(2)	<input type="color"/> C0C0C0
BackGradient	Solid
ForeColor	<input type="color"/> FFFFFFFF
LineColor	<input type="color"/> 7F7F7F
RowPadding	0
ScrollBars	Vertical
ShowLines	None
<b>List Heading</b>	
BackColor(1)	<input type="color"/> FFE6E6
BackColor(2)	<input type="color"/> FF8484
BackGradient	Vertical
FontStyle	(Default)
ForeColor	<input type="color"/> 000000
Padding	4
<b>List Selected Item</b>	
BackColor(1)	<input type="color"/> D78B4E
BackColor(2)	<input type="color"/> BB7943
BackGradient	Vertical
ForeColor	<input type="color"/> FFFFFFFF

### Edit InLine Button & Edit InLine Pressed

BackColor(1), BackColor(2), BackGradient and ForeColor are applied the same as in other examples.

### List Control

BackColor(1), BackColor(2), BackGradient, ForeColor and LineColor are applied the same as in other examples.

RowPadding – sets the row height in pixels for Listbox and MenuList controls

ScrollBars – set to None, Horizontal, Vertical or Both to have the Listbox and MenuList controls show or hide the default scrollbar. This is only the default and can be overwritten in the form design.

ShowLines – will hide or show the lines between rows and columns. The options are None for hiding all the lines, Horizontal for showing only the lines between rows, Vertical for showing only the lines between columns, and Both for showing lines between rows and columns.

### List Heading

BackColor(1), BackColor(2), BackGradient, FontStyle and ForeColor are applied the same as in other examples.

Padding – pads the heading for all list controls

### List Selected Item

BackColor(1), BackColor(2), BackGradient and ForeColor are applied the same as in other examples.

The third element is called Menus and covers the look of the menu screens.

<b>Menus</b>	
BackColor(1)	<input type="color" value="#5A351F"/>
BackColor(2)	<input type="color" value="#000000"/>
BackGradient	Solid
BorderStyle	Transparent
ForeColor	<input type="color" value="#FFFFFF"/>
ListStyle	Buttons
NormalizeText	True
Overflow	Vertical
RowPadding	4
ScaleDownText	2
ScrollBars	None
ShowHeading	Always
<b>AutoEntries</b>	
AddItems	Both
BackupImage	green_left_arrow
BackupText	Back
LogoutImage	Stop
LogoutText	Logoff

## Menus

BackColor(1), BackColor(2), BackGradient and ForeColor are applied the same as in other examples.

BorderStyle – set to Standard, No Border or Transparent to change how the menu section looks around other prompts or the title bar.

ListStyle – is the options for the menu display. They are Standard text list, Image List, Buttons and Desktop.

NormalizeText – trims spaces from the captions so they will center better

Overflow – specifies which way the remaining items will be displayed. If there are more items than will fit on the device's screen this option can be set to horizontal or vertical which means the user can swipe bottom-to-top or right-to-left to access the remaining data.

RowPadding – changes the height of the rows when the menu's ListStyle property is set to Standard.

ScaleDownText – reduces the size of the caption text by a factor and only applies to the Buttons and Desktop line list styles.

ScrollBars – set to None, Horizontal, Vertical or Both to have the menu show or hide the default scrollbar.

ShowHeading – turns the list heading on or off. The options are **Never** show the list heading, **Always** show the list heading or only show the list heading **If Not Empty** (it has been given a value).

#### AutoEntries

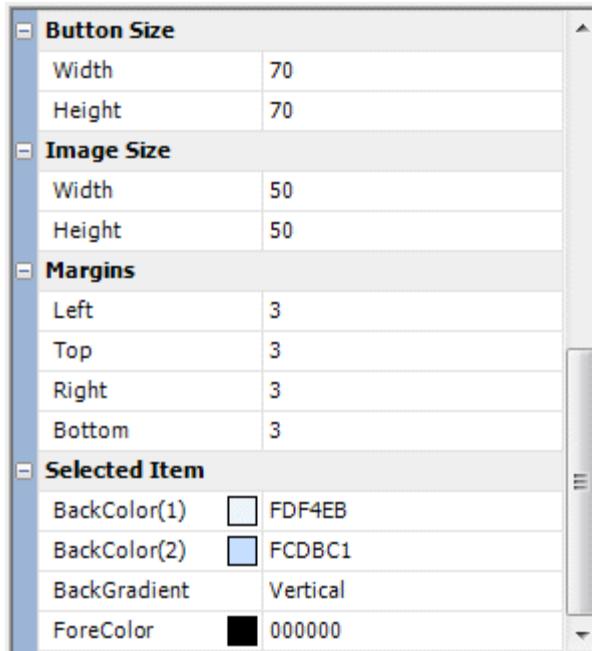
AddItems – select if the menu should have additional options like a Logout, Backup menu option or both.

BackupImage – If the AddItems property includes the Backup option, this property selects from the Image Resources for an icon.

BackupText – The text under or next to the icon describing the backup option

LogoutImage – If the AddItems property includes the Logout option, this property selects from the Image Resources for an icon.

LogoutText – The text under or next to the icon describing the logout option



### Button Size

Width – the width of the space that contains the image and the text. This applies to Button and Desktop modes.

Height – the height of the space that contains the image and the text. This applies to Button and Desktop modes.

### Image Size

Width – the width of the icon used in the menu. This applies to all menu styles except Standard.

Height – the height of the icon used in the menu. This applies to all menu styles except Standard.

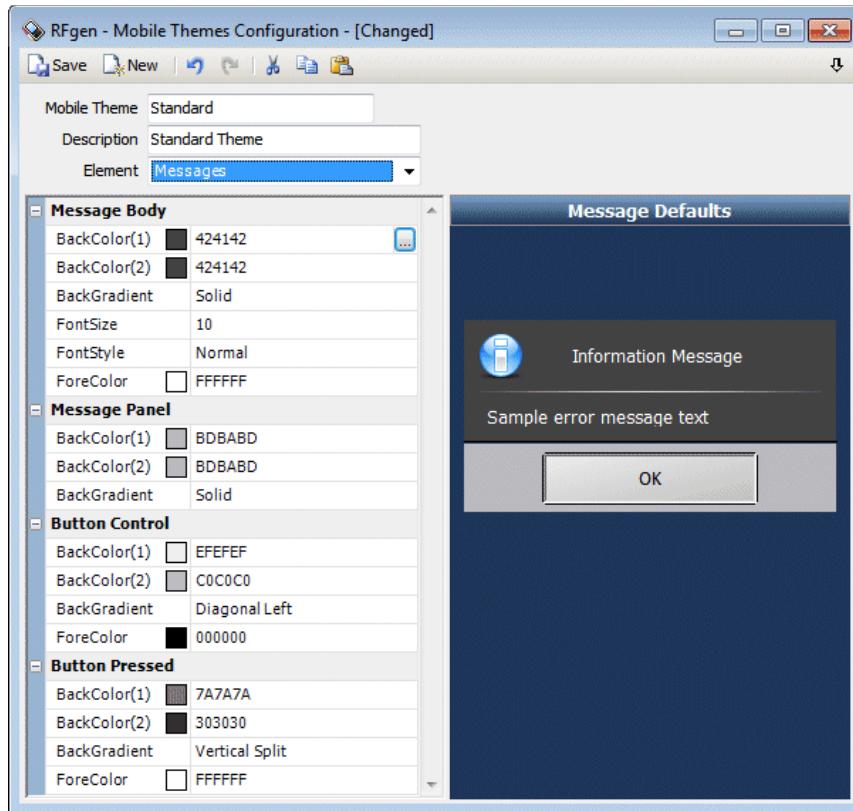
### Margins

Left, Top, Right, Bottom – the padding in pixels between the buttons or rows in the menu. This applies to all menu styles except Standard.

### Selected Item

BackColor(1), BackColor(2), BackGradient and ForeColor are applied the same as in other examples.

The forth element is called Messages and covers the look of the message box popup window.



### Message Body

BackColor(1), BackColor(2), BackGradient, FontSize, FontStyle and ForeColor are applied the same as in other examples.

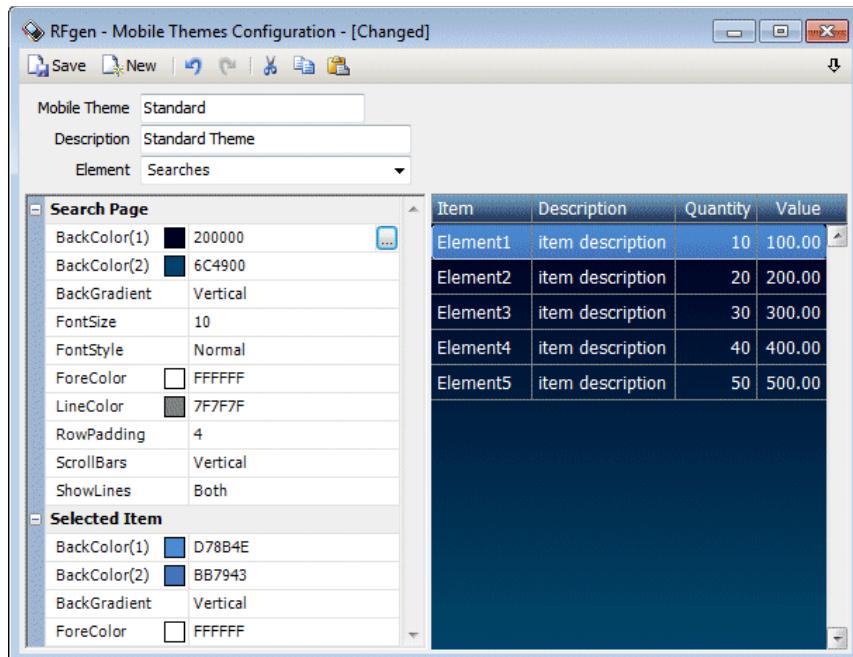
### Message Panel

BackColor(1), BackColor(2), and BackGradient are applied the same as in other examples.

### Button Control & Button Pressed

BackColor(1), BackColor(2), BackGradient and ForeColor are applied the same as in other examples.

The fifth element is called Searches and covers the look of the search screens reached from the form.



### Search Page

BackColor(1), BackColor(2), BackGradient, FontSize, FontStyle and ForeColor are applied the same as in other examples.

LineColor – the color of the lines within the grid if they are shown.

Row Padding – is the number of pixels above and below the text in a row that are added for padding.

ScrollBars – how many scroll bars are supported in the search list screen. The options are none, horizontal, vertical and both.

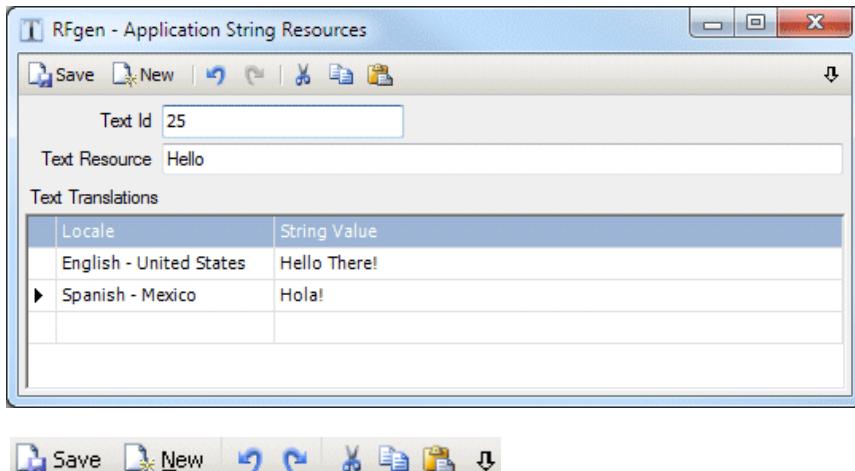
ShowLines – set to None, Horizontal, Vertical, or Both to show the grid lines in the search pages.

### Selected Item

BackColor(1), BackColor(2), BackGradient and ForeColor are applied the same as in other examples.

## Translations

The Translations Resources are meant to easily create a list of customizable strings that may be referenced from the code. Error messages, warnings or just general text are examples of strings that could be referenced. The translations resource contains any required translation from one language to another and the appropriate conversion is based on the locale of the device or user using the application.



The **Save** option performs a save of the text resource to the database. The **New** option clears the grid and provides a blank window for creating a new set of strings. Also provided are **Undo**, **Redo**, **Cut**, **Copy** and **Paste**. The **Dock** icon, (the down arrow) displays on every un-docked window allowing the user to re-dock back to the Mobile Development Studio main window. Double-clicking the top-most tab will undock the window.

The **Text Id** is the identifier of the resource referenced in the code. The **Text Resource** is the name of the resource referenced in the code. Either one can be used in the code but both are required.

Select the **Locale** and enter a **String Value** that should be returned when the code references the Text Resource id.

For an example see the App.GetString command.

The locale can be set in a few ways. Configuration / Desktop Preferences menu has a place to set the default locale and App.Locale = 1033 is the script command to set or change the locale possibly based on a user's property.

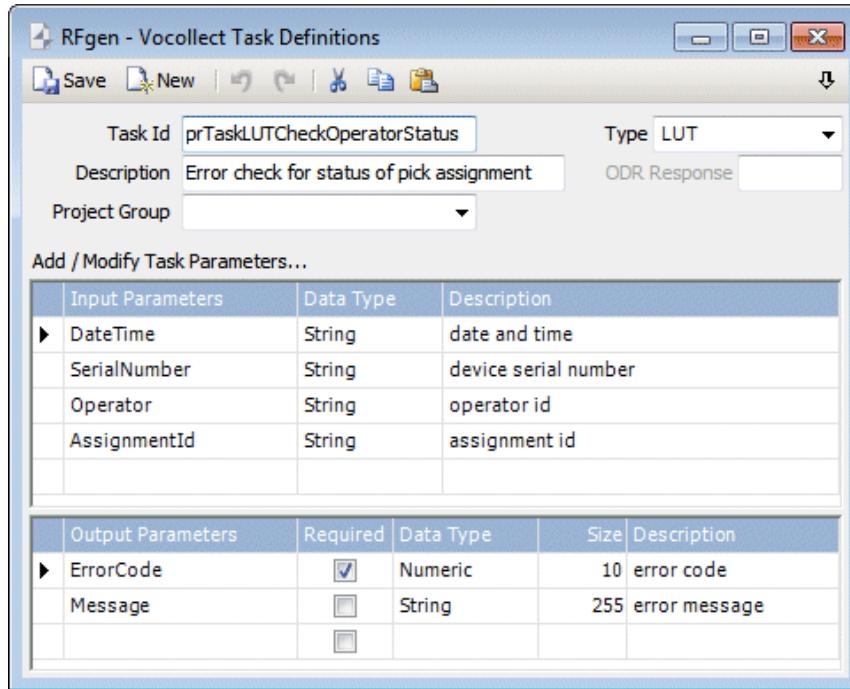
## Vocollect Tasks

Vocollect tasks are pre-defined scripts that execute on Vocollect hardware and interact with RFgen to provide the voice solution together with the backend data connection solution.

Either right-click on the tree or double-click an existing task to bring up the design window. Alternatively, you can use the Import utility from the RFgen System Files group and choose predefined Vocollect tasks rather than creating them manually.



The **New** option clears all fields and provides a blank window for creating a new Vocollect task. The **Save** option performs a save of the task to the database. Also provided are **Undo**, **Redo**, **Cut**, **Copy** and **Paste**. The **Dock** icon, (the down arrow) displays on every un-docked window allowing the user to re-dock back to the Mobile Development Studio main window. Double-clicking the top-most tab will undock the window.



The **Task ID** is RFgen's identifier but in the case of the pre-defined tasks Vocollect provided the names. The **Description** clarifies the task and displays in the tree. The **Type** is either LUT or ODR. LUT is two-way communication between the Vocollect device and RFgen and ODR represents one-way communication. For ODR, there are no output parameters. The **ODR Response** field can contain any value and it represents an acknowledgement bit from RFgen to make the Vocollect device stop polling RFgen for a response. Types are input only and are used for tasks such as updating a count or updating a status. The **Project Group** lets the user group like items together in the navigation tree.

The Task Parameters are defined by Vocollect and are configured here. RFgen contains several pre-entered tasks which can be found by choosing the Utilities / Import Mobile Applications menu, selecting Import From Cached System Files, selecting one or more of the Vocollect Tasks and clicking on Import.

# Screen Mapping

The Screen Mapping module enables mobile applications to be mapped against multiple host systems like the AS/400, IBM Mainframe, UNIX systems and other character-based ‘legacy’ applications. In practice, screen mapping applications use keystrokes recorded for a host screen’s navigation and data entry, along with the collected data and play back of the keystrokes, while replacing the recorded data with the newly collected data. Accuracy in staging and applying keystrokes is of the utmost importance. The recording capabilities provide this needed level of accuracy. The solution provides three host protocols: TN5250, TN3270, and VT220 in order to interact with legacy hosts.

The screen mapping applications may be created by means of an automatic recording processes, and point and click, drag and drop development methods. The automatic recording processes create Visual Basic for Applications (VBA) macros (i.e., scripts) that utilize pre-built screen mapping extensions for system navigation and data handling. An intuitive set of VBA extensions have been designed to interact with any character-based legacy application. Users may, of course, modify scripts as desired or create new scripts. **Screen mapping supports transaction queuing so that when a host is offline, data collection may continue uninterrupted.** The system thus allows true 24/7 support for critical data collection operations.

## How to Make Screen Mapping Work

The Screen Mapping module is included with this base system. When loaded and authorized, the system functions (simultaneously) as both an ODBC database server and a legacy host terminal server.

To properly function with a host AS/400, IBM mainframe, UNIX, or other legacy-based system, the server must be part of a communications network, capable of interacting with a host via TCP/IP networking protocols.

## Theory of Operation

Screen Mapping works by identifying a Main Menu in the host system that is used as a base reference point for navigation to and from transaction screens. This is the starting point for transaction processes.

The navigation process (i.e., series of keystrokes) required to proceed from a login state to the Main Menu may be automatically recorded by the system. A small visually unique portion of the main menu screen is marked for identification purposes. If required, multiple areas of the main menu may be marked. This allows the server to know that it has arrived at the requested destination.

The next step is the navigation from the main menu to the transaction screen. Transaction screens are the displays that users use to input data into the host system. Again, the navigation process (i.e., required keystrokes) to reach a screen and then return to the main menu may be automatically recorded.

When a transaction screen has been identified, the next step in the process is to identify the fields on the screen where the data will be entered. Screen fields are marked and each is given a field name. These fields may optionally be used in much the same way as ODBC fieldnames are used by the system; i.e., they may be used to create transactions by dragging the host screen's marked fields on to an application screen. Doing this, means that the user can scan and enter all the data on the mobile device and the server already knows how to log on to the host system, how to navigate to the proper screen and where to place the collected data on the host screen, all without having to program the scripts yourself.

## Programming Philosophy

Programming Screen Mapping applications differ from typical data collection applications in that problems, if and when encountered, need to be handled automatically by the application program without the involvement of the remote data collection users. For example, the RFgenSM module contains built-in commands, such as 'SM.GetText', that can be used to search for specified text in a host screen (at a specific 'Col, Row' location, or anywhere on the screen). This, plus other diagnostics, allow programmers to positively identify the correctness of 'happenings' within a host application.

## Design Considerations

Before starting a screen mapping project, users should consider certain project design issues related to the following topics: Screen Mapping Level, Logon Security, Data Integrity, Keyboard/Special Key Configuration, and Runtime Environment/Variables.

## Screen Mapping Levels

The Screen Mapping interface is divided into 3 levels of usage: **Low**, **Medium** and **High** levels.

**Low level** use is represented by scripting in the VBA environment all aspects of interaction between the server and the host system using the SM object's methods and procedures. These commands allow the developer complete control over the host session. Examples include sending/receiving text, control keys, cursor positioning, "WaitFor" statements, "Find" statements, etc. It is entirely possible for the user to write/program complete solutions using only these low-level commands. These commands are documented in the Screen Mapping Extensions section.

**Medium level** use is typified by the creation Host Screen macros and / or Data Entry macros using the recording capabilities. At any time in the VBA script, one of these macros can be called and the host screen can be made to navigate or transact instantly. This capability simplifies the programming of the navigation requirements within a host system. Calling a transaction macro will place all collected data into the host screen's fields and submit the screen to the host for processing. Transaction macros can have input / output parameters. These parameters are used to send and receive data from the host screen. Because of the solution's unique design, transaction data can be stored while the host is offline, and send to it for processing later when the connection is re-established.

Medium level usage entails the development of a VBA script to call the pre-recorded macros. One Screen Mapping command 'SM.CallMacro' is oftentimes sufficient to update the host.

**High level** use of the Screen Mapping capabilities is represented by the automatic recording of the 'Host Screen' and 'Transaction Macros' discussed above. Host transaction fields are then embedded in applications in much the same manner (e.g., drag and drop) as table fields from ODBC databases. Using embedded methods, data automatically posts to the host once all input fields have been entered (note: posting was accomplished manually as the last step in Medium level usage, not automatically as with embedded fields).

**A final note:** All automatically recorded macros are created using base low-level commands. Thus, users have complete access to all VBA scripts, including modifying them as desired. An example of user modifications might include checking for error conditions (such as bad data) and/or warning, error, or informational messages. Copying and

pasting the recorded macro script and placing it in the application directly is a quick way to build a low level solution.

## Logon Security Considerations

There are a number of ways the programmer can implement security. One important thing to remember with screen mapping is **that the end-user is never on-line with the host system**. The end-user has no way of interacting with the host system that you haven't provided for. With this in mind, the following are a few examples of login security methods:

The developer can create a "Login" Transaction Macro that is linked to the Login Host Screen. If a new user needs to sign on, this can be accomplished through a simple call to the "Login" macro.

The user ID and password specified when the user logged in could be provided to the script. The login would occur when the user called the first macro.

A generic login could be specified, and the user changed dynamically using a system function such as "sign-on" or "change-to". This command could be executed as part of a single Transaction Macro, or as a separate one called only when the user changes.

## System Integrity Considerations

Screen mapping actions that cause a host system to be updated should be acknowledged by the host before another command is sent. The host interface is designed to automatically accommodate for this as much as possible. The script or macro waits for the host to not be busy before reading from or writing to the host screen. In a host-busy condition (input inhibited), The server will wait for the timeout period specified in the settings for the condition to clear. However, screen mapping VBA extensions should be incorporated to provide ways of acknowledging successful actions. For instance, the following commands may be used to provide programmer control over host sessions:

**SM.WaitForText** – This function looks for a unique text string on the screen for a specified amount of time. If the text is found, it returns True, otherwise, it times out with a value of False.

**SM.WaitForCursor** – This function waits until the screen input cursor stops at the desired location for a specified amount of time. If the cursor stops at the desired location, it returns True, otherwise, it times out with a value of False.

**SM.WaitForScreen** – This function looks for the desired screen identifier for a specified amount of time. If the application is in the correct screen, it returns True, otherwise, it times out with a value of False.

**SM.WaitForHost** – This function is designed for vt220 connections. As the vt220 protocol does not typically include input inhibit conditions, this function will return True once the host responds to the previous action command, otherwise, it will time out with a value of False.

**SM.CallMacro** – This function has a “Queue-Offline” argument, which, if enabled, will store the transaction if the host is offline. These “store-and-forward” transactions will be processed automatically once the host connection is re-established.

**Note:** the ‘**SM.WaitFor...**’ commands are primarily useful (and perhaps required) for VT hosts.

See Screen Mapping Extensions section for more information.

## Keyboard/Special Key Configurations

We understand that not all terminal emulations use the same keyboard layout. Accordingly, a developer is provided with 2 options to send special keys to the host:

The first is a pop-up window that is accessed by clicking on the ‘Hot-Key’ icon at the top of the Host Session window. You can then select the desired special key from the list in the window and transmit it to the host.

A second option is to re-map selected keys on your keyboard to transmit the special keys instead. These re-mappings are defined by clicking on the Session menu item at the top of the Host Session window. To re-map a key simply press the key(s) you want to re-map and then select the desired special key to send from the drop-down list.

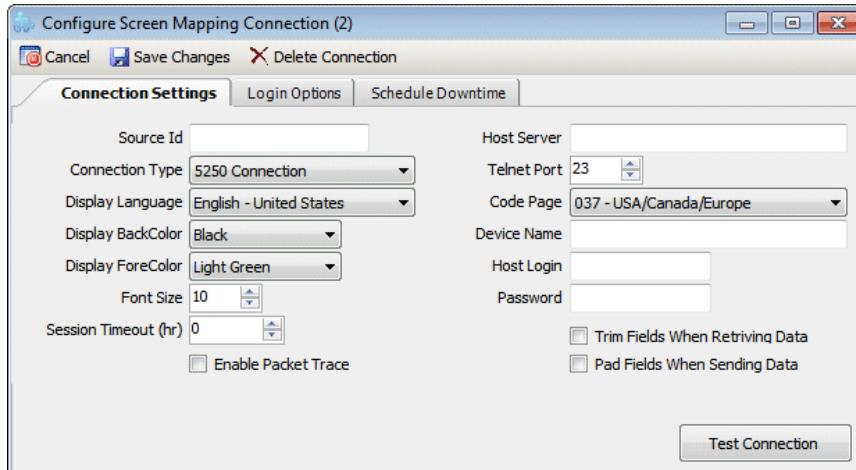
See the VT220 Key Mapping section for more information on Hot-Keys.

## Runtime Environment Variables

A programmer can make use of any of the standard language extensions while creating or modifying macros. In addition, any global variables specified in the “Win32.bas” or “RFgen.bas” modules are available for use in any macro. However, the user cannot call another macro from within a macro.

# Configuring the Host Connection

In the Mobile Development Studio, click the menu option Enterprise Connections / Add New Data Connection / Add New Screen Mapping Connection. The following window will appear.



The first entry is the **Source Id** used to reference the data connection only. This can have any value but spaces and extended characters are not recommended.

Choose the **Connection Type** (VT220, TN5250 or TN3270); i.e., the protocol used to communicate with your host system. Notice that there is an additional option called Console Application. This type is designed to launch a console application rather than use a telnet server and then pass that display through the server to the device using the HostScreen prompt control. One example would be the SAP console application (SAPCNSL.EXE) running on the server and being displayed and allowing interaction with the user on a mobile device. Simply specify a process or executable name to run and any passing parameters necessary.

The preferred option is UTF-8 but if a legacy system's output is language specific then the **Display Language** field should be changed to make the screen render correctly. The Language field can be left as (Default) if a code page is specified or if UTF-8 is used.

Preferences for the emulation screen include **Back Color**, **Fore Color** and **Font Size**. These are only for development since the screens are hidden during production.

The **Session Timeout** value (in hours) will disconnect and reconnect to the legacy server at the specified interval. This may be required if the legacy server is configured to not allow a connection that never times out.

In the case of communication errors the **Enable Packet Trace** option can be set and a trace log of the communication will be captured. This is used by support staff to diagnose issues on behalf of the customer. Please contact support if this switch is necessary.

Next, type in the **Host Server** name or IP address. The **Telnet Port** is the port that the server uses to communicate with your host. The default for a telnet server is port 23.

If TN5250 or TN3270 are selected, you may enter a **Code Page** for specifying the language being used in the protocol and an **IBM Device Name** for the host system. Code pages were selected for loading when you loaded the screen mapping software. These fields are hidden in the VT setup.

For VT220 the **Data Stream** field can be set to either Standard or UTF-8 to accommodate the type of packet data coming from the host system.

When using the connection type 3270 or 5250, the **Device Name** field is designed to make each connected device appear unique to the host system. Leaving it blank, the host system will not distinguish between the connecting clients. Fill this field in with a name and the server will automatically add a three digit, zero padded number to each client so the host system will see each connecting session as a unique device.

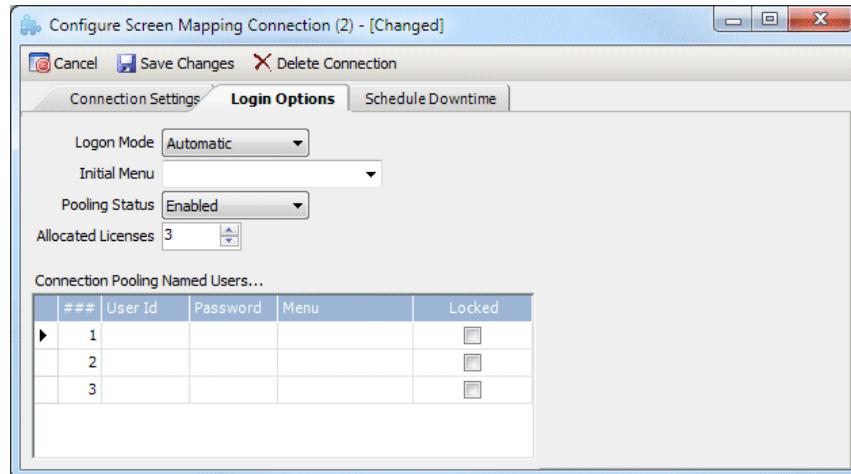
The **Host Login** and **Password** fields are used only if SSH is used when connecting to the host system. Under the VT220 options, if Connect via SSH is checked then the Host Login and Password are required.

**Trim Fields When Retrieving Data** set to enabled will auto trim spaces from the host output fields. If a variable is defined for a section of the host screen (like where error messages are displayed), this feature will trim the text for easier use in message boxes, for example.

The **Pad Fields When Sending Data** option when enabled will use spaces to pad any input. A variable defined for a region of the host screen where input will take place also has a length property assigned at the time the field was defined. If the data is 3 characters, but is placed in a host screen field designed for 10 maximum characters, the server can pad the input data to fill up the host screen input field.

There are some additional properties for the VT220 mode only. **Echo Characters Locally** means that the server will print the typed characters on the telnet screen because the host is accepting the keystrokes but not showing them to the user. **Wrap Text at End of Line** will force the server to place the additional text on the next available line if it doesn't fit in the current field. Most host system will do this automatically. **Destructive Backspace** means that the server will receive a backspace command and apply it to the screen as a command that removes the last character. Some systems would move the cursor but not remove the character. **Send Whole Key Packets** forces the server to submit keystrokes in one packet instead of two in some cases. Most host systems already support keystrokes coming in as one or more packets. **Send Return + Line Feed** will add a carriage return plus a line feed to the Enter keystroke when communicating with the host. **Connect via SSH** will establish an SSH (secure) connection to the host from the server. If this option is turned on then the **SSH User Name** and **Password** fields will be required.

The **Test Connection** button will verify all settings before saving the connection. This is not required.



On the **Login Options** tab are options for specifying how the host will log in and if the connection should be pooled.

The **Login Mode or Auto Login** (VT220) can be either Automatic or Manual. Automatic means that the defaults will be used and when the session is started, the default user, password and main menu will be used to log in. Manual means that the session will be started and the script must pass the user, password, and navigation for the main menu.

The **Initial Menu** is the Hosts macro to be used to log the host system in. This is typically the launching point for all navigation to host transactions.

Connection Pooling can be enabled and the maximum connections allowed in the pool can be selected. This selection will determine how the server and its clients will interact with your host system.

The options for the **Pooling Status** are:

Disabled – Setting connection pooling to disabled will cause the server to spawn a connection to the host system for each active mobile device. Each connection will be linked to a particular device on a one-to-one basis, and will be shut down when that device disconnects. Note: there is no limitation on the number of connections allowed.

Enabled – Setting connection pooling to Enabled will cause the server to spawn a single connection to your host system. As each device requires access to the host system, they will go to the pool and retrieve one of the available connections. When they are finished, the device will release the connection back to the pool. If no connections are available, the server will start a new connection (up to the specified maximum) and add it to the pool. After 10 minutes of non-use, an opened pooled connection will be terminated releasing resources on the server and potentially licenses on the host system. Keep in mind that unless the SM.BeginTrans and SM.CommitTrans commands are used, it would be possible for one user to position the screen in one place while another user also uses that pooled connection to perform their tasks causing both users to get failures.

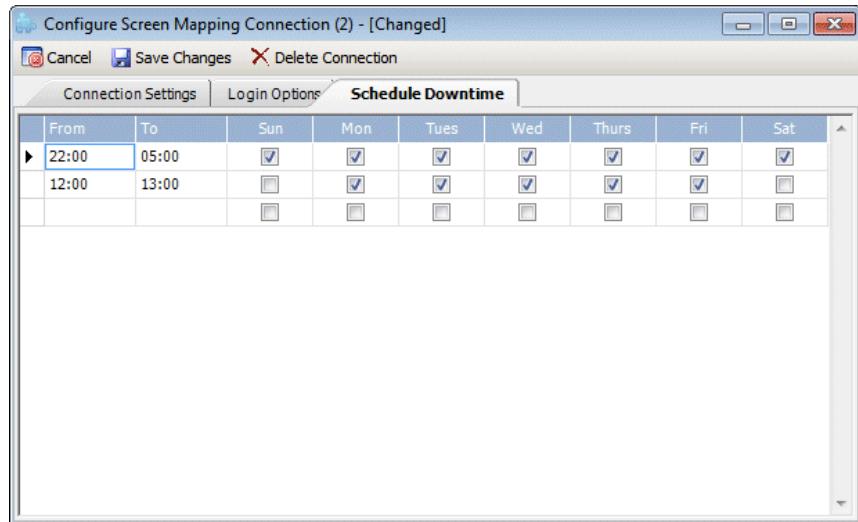
The **Connection Pooling Named Users** grid dictates how each host session is started. You may also override the default settings by configuring a specific pooled session separately.

Session	Each of the individual pooled connections are listed separately. This provides for specific settings for each connection.
User Id	If the host system requires that unique names be used or creating multiple logins with the same user is prevented, each pooled connection can have its own user ID. Session, user, and password information can be obtained at runtime with the commands SM.SessionUser, SM.SessionPwd, and SM.SessionID.
Password	This is the corresponding password used for each unique user ID.
Menu	Each session can have its own main menu. When a session is requested and no main menu is specifically assigned or the "(Default)" value is used, the next available session will execute the requested main menu based on the scripts and chosen transaction. If a session is requested and the next available session does have a main menu assigned, and it is not the required one, other sessions will be evaluated for a matching main menu. If one is found and available, it will be used.
Locked	<p>The ability to lock a session means that the session can ONLY be used with the specified main menu and will not allow other main menus, even if all other available sessions are in use. For example, there are 10 pooled sessions, five locked on main menu A and five locked on main menu B. If a session with main menu A is requested and all five sessions for main menu A are currently used, the server will look to the sessions assigned to main menu B. If they are not locked, the server will take one of them. Since they are locked into main menu B, in use or otherwise, the server will wait for one of the first five to be released.</p> <p>The purpose of locking a set number of sessions to a specific main menu is to ensure that there is always some bandwidth available for certain transactions. Not locking them means that they will be marked with a preference for a type of transaction (the use of a specific main menu), but will switch to another main menu when necessary.</p>

For example, there are 10 pooled sessions available and the first five have one main menu assigned and the last five have a second main menu assigned. When a session with the second main menu is requested, the 6<sup>th</sup> session handle will be used. This is only significant because of the Locked property.

## Scheduled Downtime

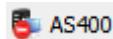
The Scheduled Downtime option in the ‘Screen Mapping Connection’ Window allows users to schedule ‘down time’ for a host connection; i.e., **the server will disable the connection** during the time that a host is offline. The following panel will appear.



In this example the connection will be unavailable between 10PM and 5AM the next day and noon (12PM) and 1PM on the weekdays. The check boxes for each day refer to the starting time only. There is no harm in having overlapping down times. (Time is shown in a 24-hour notation.)

During the down time, transactions can be ‘queued’ for automatic posting to your host when the connection becomes available.

When you ‘Save’ the screen mapping configuration entries, a session with your host should be available by right-clicking on the screen mapping connector and selecting Show Host Connection and the host name will appear as a ‘connection indicator’ at the bottom of the Mobile Development Studio window. A red circle in the connection indicator...



indicates that a connection has not been established with your identified host.

With a host connection established, your screen mapping development project is ready to be started.

## VT220 Key Mapping

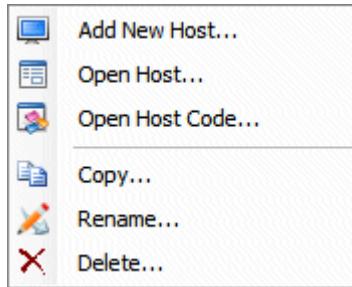
The default key mapping for VT sessions can be edited if certain keys are not working correctly with the host. Use the Import utility from the menu, choose Cached System Files, and import the VT220 Key Map Template. A file called VT220.VKM will be placed in the install directory and can be edited with a text editor.

A screenshot of a Microsoft Notepad window titled "VT220.vkm - Notepad". The window contains a list of key mappings in a tab-separated value (TSV) format. The columns are labeled with the key name and its corresponding value. The key names include standard keyboard keys like BACKSPACE, BREAK, and various function keys (F1-F16), along with specific terminal control codes like TERMINAL\_ID and DELETE. The values are represented as strings of characters or escape sequences, such as 'VT220' for TERMINAL\_ID or '27 + '[A' for CURSOR\_UP.

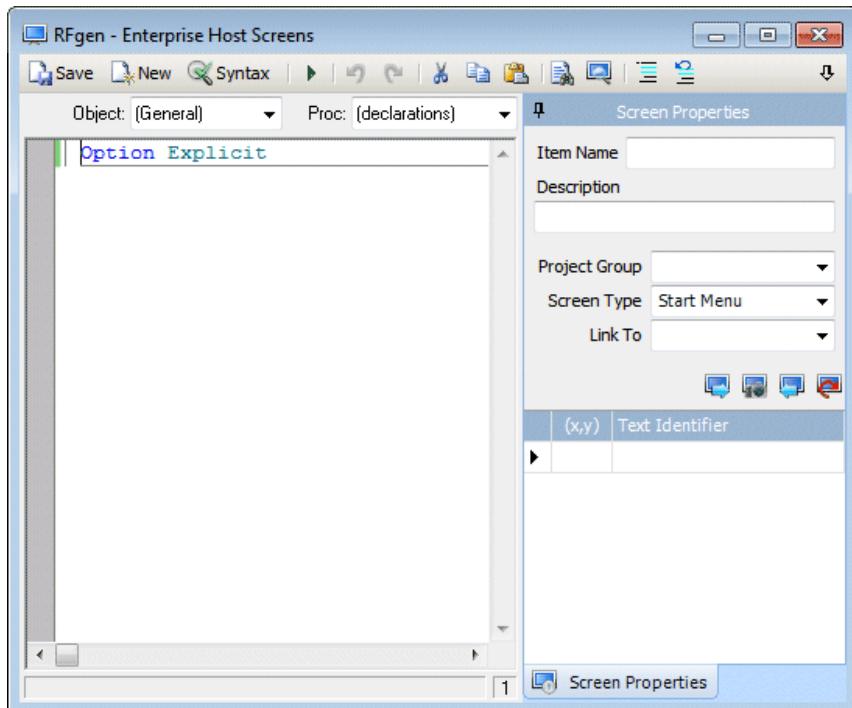
Key Name	Value
TERMINAL_ID	'VT220'
BACKSPACE	8
BREAK	3
CURSOR_UP	27 + '[A'
CURSOR_DOWN	27 + '[B'
CURSOR_RIGHT	27 + '[C'
CURSOR_LEFT	27 + '[D'
DELETE	127
DO	27 + '[29~'
ENTER	8
ESCAPE	27
F1	27 + 'OP'
F2	27 + 'OQ'
F3	27 + 'OR'
F4	27 + 'OS'
F5	27 + '[15~'
F6	27 + '[17~'
F7	27 + '[18~'
F8	27 + '[19~'
F9	27 + '[20~'
F10	27 + '[21~'
F11	27 + '[23~'
F12	27 + '[24~'
F13	27 + '[25~'
F14	27 + '[26~'
F15	27 + '[28~'
F16	27 + '[29~'

## Hosts Tree

Double-click on the Hosts heading to create a new screen mapping macro or right-click on the title of an existing macro to make additional changes.

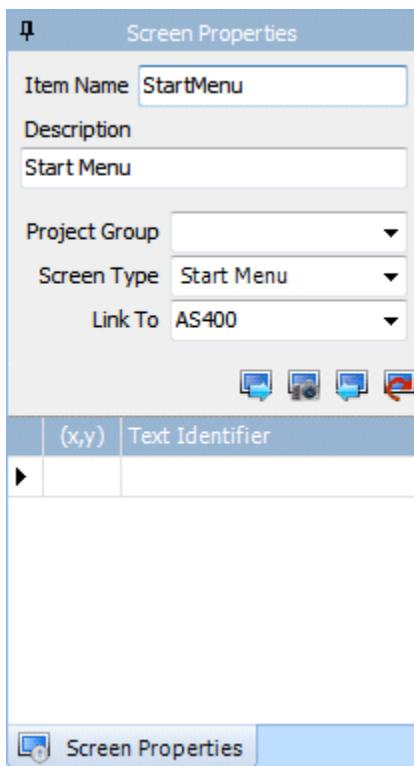


Double-clicking on an existing macro opens the Host Screen Editing window.



There are two types of macros that can be created here. **Start Menu** macros take the data connection as it is when first connected and logs in and navigates to a main menu used as a generic starting point for all screen mapping transactions. **Host Screen** macros are used when a specific transaction is chosen by the mobile user to navigate the host system to the proper screen meant to accept specific data (ex.: Cycle Count screen). Additionally, this macro can be used to play back the keystrokes of a user entering the collected data into the screen itself. This macro stores the x,y coordinates of the fields on this host screen and places the collected data in the proper places before sending additional keystrokes to submit the data (ex.: F8). At that time the host screen processes the data just as if the user entered the data directly to the host screen.

## Building a Start Menu Macro



Begin by entering the **Item Name** of the Start Menu macro. This will be the name given to the primary main menu used to log the host system in. Fill in the **Description** field, select a **Project Group** if desired, select

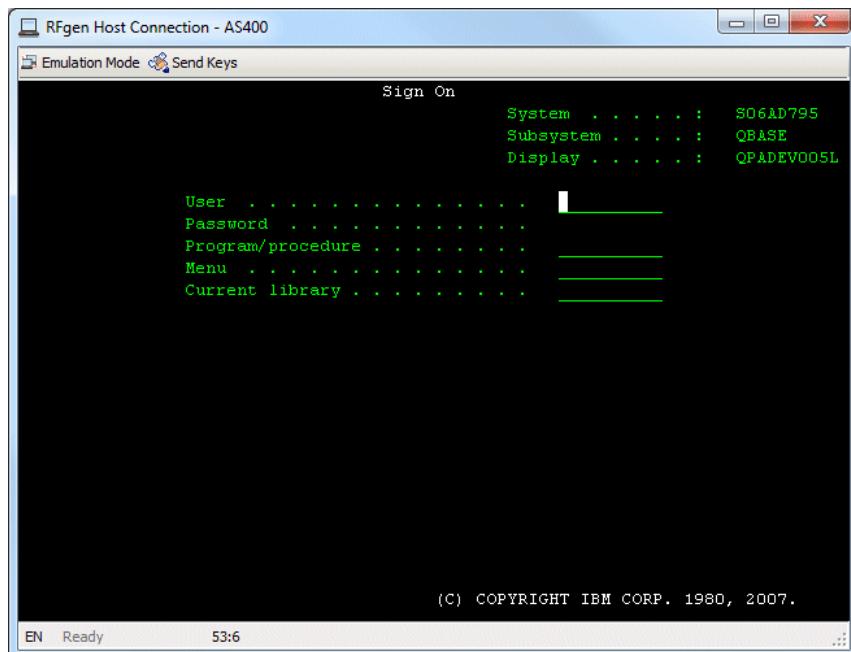
Start Menu for the **Screen Type** and **Link To** the name of the screen mapping data connection.

Next select the Recording menu option.



There are four different scripts that this macro can contain. **Record system sign-on** records the keystrokes to successfully log in to the host system and navigate to the main menu. **Identify system menu** records the x,y coordinates of some text on the host screen, so when the system attempts to reach this page it will compare the host screen to what is known to be the proper screen. **Record system sign-off** contains the keystrokes recorded for exiting the main menu and going back to the login screen. **Record error recovery** records the keystrokes to do whatever the user must to get the host back to the main menu. Usually the safest solution is to back out as far as will ever be needed and then log in again.

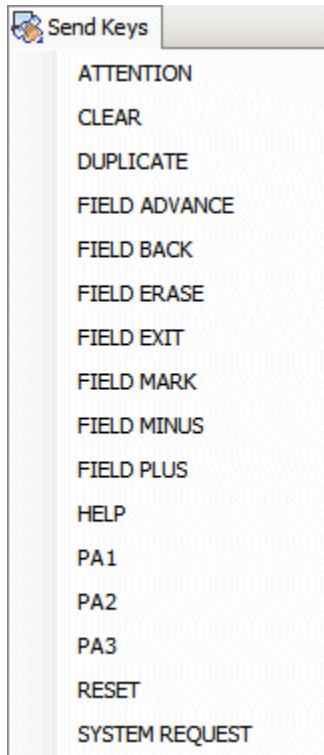
When recording these macros the Host Session window will appear. In the case of this host system, it is an IBM AS/400 connected to RFgen using a TN5250 telnet protocol.



The first toolbar icon represents the mode the window is in. Examples are Emulation Mode, Recording Mode and Identifying Mode.

## The Send Keys Selection

Selecting a 'Hot Key' from the available list is an alternative to re-mapping your keyboard. The following list of keys will appear.



Clicking on a menu item will send the selected key to your host. If you are currently recording a macro, the selected key will become part of that macro.

## Start Menu Macro – Record System Signon

Choose the Recording menu option and select Logon to the Main Menu. The Host Session window will display with a column on the right for recording and editing all keystrokes.

Enter the keystrokes necessary to display the main menu. During the recording phase the user may right-click on the host system to bring up a menu of additional commands that may be inserted into the script.

See the Recording Options section below for a complete description of these commands.

Add FindText Statement
Add GetArea Statement
Add GetCursor Statement
Add GetText Statement
Add WaitForCursor Statement
Add WaitForCursorMove Statement
Add WaitForHost Statement
Add WaitForText Statement
Add WaitForWrite Statement
<hr/>
Select Current Field

The Scripting Commands column on the right can also be edited in case a mistake is made, variables need changing or timeout values need to be adjusted. If a step is forgotten, such as waiting for text to appear on the screen before performing the next keystroke, simply position the insert arrow on the row to be preceded by the missing command and perform that command. For example, highlight a section of text, right-click and select the Add WaitForText Statement option. If a keystroke was pressed accidentally, simply delete it from the list. Be sure to put the insert arrow back at the bottom before continuing.

Scripting Command
SM.SendTextAlt 7, 19, "15"
SM.SendKey KeyEnter
SM.SendTextAlt 7, 19, "18"
SM.SendKey KeyEnter
SM.SendTextAlt 7, 19, "8"
SM.SendKey KeyEnter
SM.SendTextAlt 21, 8, "30"
SM.SendKey KeyTab
SM.SendTextAlt 4, 12, "1001"
SM.SendKey KeyTab
SM.SendTextAlt 31, 12, "25"
SM.SendKey KeyTab
SM.SendKey KeyTab
SM.SendTextAlt 48, 12, ".."
SM.SendKey KeyCursorDown
SM.SendTextAlt 48, 13, "1.."
SM.SendKey KeyEnter
▶
 Save  Cancel

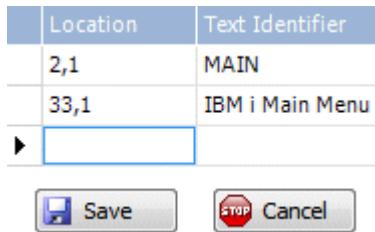
When completed, click Save. Cancel may be clicked at any time to cancel the recording session. An internal macro called **LogOnToMainMenu** is recorded by this step.

## Start Menu Macro – Identify System Menu

In order for the server to positively identify the main menu, a portion of the screen needs to be ‘marked’. From the recording toolbar button options choose Identify system menu.

Next, select unique text for this screen on the host system by left-clicking and dragging across the text and then select the Mark Field menu option. If necessary, multiple selections can be made.

The marked region and its coordinates are placed in a grid on the right side of the host screen window where the whole list is captured and can be edited.



When complete, click 'Save' to save all marked areas. These identifiers will appear in the Screen Properties window.

### Start Menu Macro – Record System Signoff

From the recording toolbar button options choose record system signoff. The Host Session window will display with a column on the right for recording and editing all keystrokes.

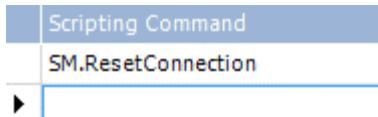
Enter the keystrokes necessary to logoff from the main menu. During the recording phase, the user may right-click on the host system to bring up a menu of additional commands that may be inserted into the script. This was described in the first step.

When completed, click Save. Cancel may be clicked at any time to cancel the recording session. An internal macro called **LogOffFromMainMenu** is recorded by this step.

### Start Menu Macro – Record Error Recovery

This step records the keystrokes that will navigate the host system out of any possible screen or menu back to the known main menu. If the host system pops up additional screens like system messages that the scripts do not take into account, then the server would notice that none of the recorded identifiers match were the host is and run this script.

From the recording toolbar button options choose record error recovery. Enter the keystrokes necessary to get back to the main menu. Possibly, an easier solution would be to type in the SM.ResetConnection command as shown.



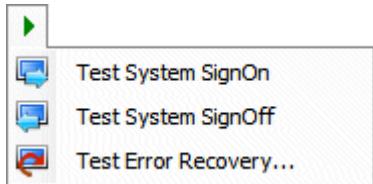
When the system resets, it automatically re-runs the LogOnToMainMenu macro taking the host to the main menu. In this case, the complete **AbortNavigation** macro would look like this:

```
Private Function AbortNavigation() As Boolean
    On Error Resume Next
    '
    SM.ResetConnection
    '
    AbortNavigation = SM.WaitForScreen("Base", 10)
End Function
```

Be sure the host system is not adversely impacted by using the SM.ResetConnection command.

## Start Menu Macro – Test Scripts

After the first four steps have been completed, the recorded macros should be tested.



The drop-down menu from the 'Test' button allows you to select the macro to be tested. A message box will appear showing the success or failure of the macro.

Important:

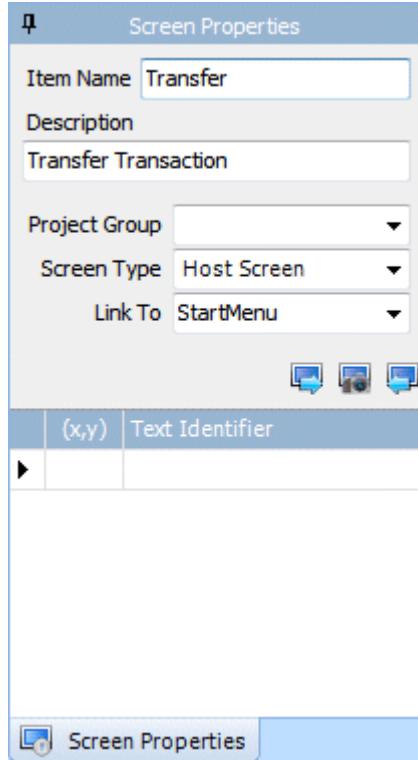
1. To test the **LogOnToMainMenu** macro, your host screen should first be positioned at your login screen.

2. To test the **LogOffFromMainMenu** macro, your host screen should be positioned at your Main menu.
3. To test the **AbortNavigation** macro, your host screen should be positioned anywhere in the menu structure but not on an application screen.

Be sure to save all the work that has been done before moving on to the Host Screen macro recording steps.

## Building a Host Screen Macro

The Host Screen macro contains just the navigation to and from a transaction screen. A Transaction macro is linked to this macro so when a form tries to update the host screen the Transaction macro will use this Host Screen macro to perform the required navigation.



Begin by entering the **Item Name** of the Host Screen macro. Fill in the **Description** field, select a **Project Group** if desired, select Host Screen

for the **Screen Type** and **Link To** the name of the start menu just defined.

Next select the Recording menu option from the toolbar buttons.

There are three different scripts that this macro can contain. **Go to the Application Screen** records the keystrokes to successfully navigate the host system to the particular transaction screen from the already defined main menu. **Identify the Application Screen** records the x,y coordinates of some text on the host screen so when RFgen attempts to reach this page, it will compare the host screen to what is known to be the proper screen. **Return to the Main Menu** is the keystrokes recorded for exiting the transaction screen and going back to the main menu.

## **Host Screen Macro – Go to the Application Screen**

Choose the Recording menu option and select Go to the Application Screen. The Host Session window will display with a column on the right for recording and editing all keystrokes.

Before proceeding, please ensure that you are on the host's Main menu (as previously identified).

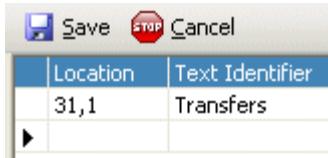
Enter the keystrokes necessary to navigate to the transaction screen. During the recording phase, the user may right-click on the host system to bring up a menu of additional commands that may be inserted into the script. When completed, click 'Save'. 'Cancel' may be clicked at any time to cancel the recording session. An internal macro called **GoToScreen** is recorded for the screen by this step.

## **Host Screen Macro – Identify the Application Screen**

In order for the server to positively identify the application screen a portion of the screen needs to be 'marked'. From the Recording menu option, choose Identify the Application Screen.

Next, select unique text for this screen on the host system by left-clicking and dragging across the text and then select the Mark Field menu option. If necessary, multiple selections can be made.

The marked region and its coordinates are placed in the grid on the right side of the host screen window where the whole list is captured and can be edited.



When complete, click 'Save' to save all marked areas. These identifiers will appear in the Host Screen Editing window.

## Host Screen Macro – Return to the Main Menu

From the Recording menu option choose Return to the Main Menu. The Host Session window will display with a column on the right for recording and editing all keystrokes.

Enter the keystrokes necessary to return to the Main menu. When completed, click 'Save'. 'Cancel' may be clicked at any time to cancel the recording session. An internal macro called **ReturnToMainMenu** is recorded for the application screen by this step.

## Host Screen Macro – Test Scripts

After each step is completed, or at the end of all steps, the recorded macros should be tested. Click on the Scripts menu option and the script window will appear.

```
Private Function GoToScreen() As Boolean
    On Error Resume Next
    '
    Dim iCol As Integer
    Dim iRow As Integer
    Dim sText As String
    Dim bSuccess As Boolean
    '
    SM.SendTextAlt 7, 19, "15"
    SM.SendKey KeyEnter
    SM.SendTextAlt 7, 19, "18"
    SM.SendKey KeyEnter
    SM.SendTextAlt 7, 19, "8"
    SM.SendKey KeyEnter
    '
    GoToScreen = SM.WaitForScreen("This", 10)
End Function
```

The drop-down menu from the ‘Run’ button allows you to select the macro to be tested. A message box will appear showing the success or failure of the macro.

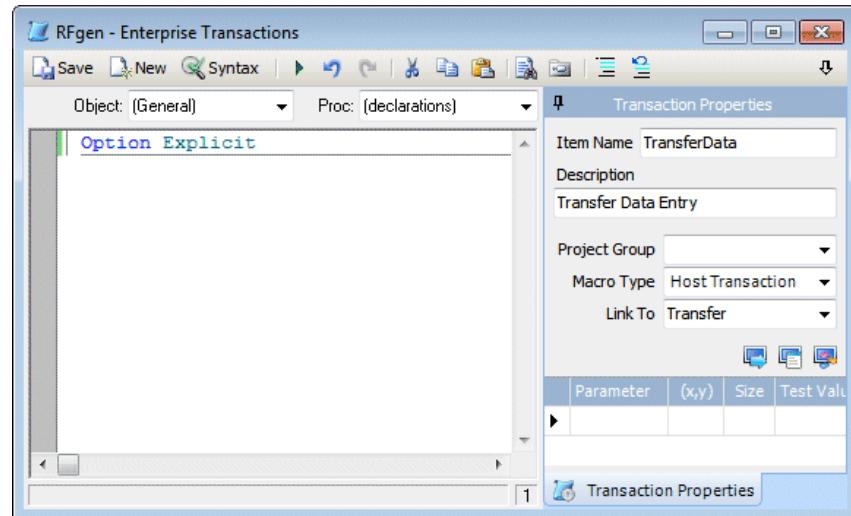
Important:

1. To test the **GoToScreen** macro, your host screen should first be positioned at your main menu.
2. To test the **Transaction Script** macro, your host screen should be positioned at your transaction screen. This macro was created when recording the Enter a Sample Transaction option.
3. To test the **ReturnToMainMenu** macro, your host screen should be positioned at your transaction screen.

Many keystrokes may have been used to navigate between screens or around the transaction screen itself that may not be necessary. By default, the text written to the screen uses the coordinates to locate the proper input field so any additional tabs, for example, to move between fields are not necessary and can be deleted. The reserved word “This” and “Base” are internally substituted at runtime depending on what names were given to the transaction macro and the main menu macro. Be sure to save all the work before exiting.

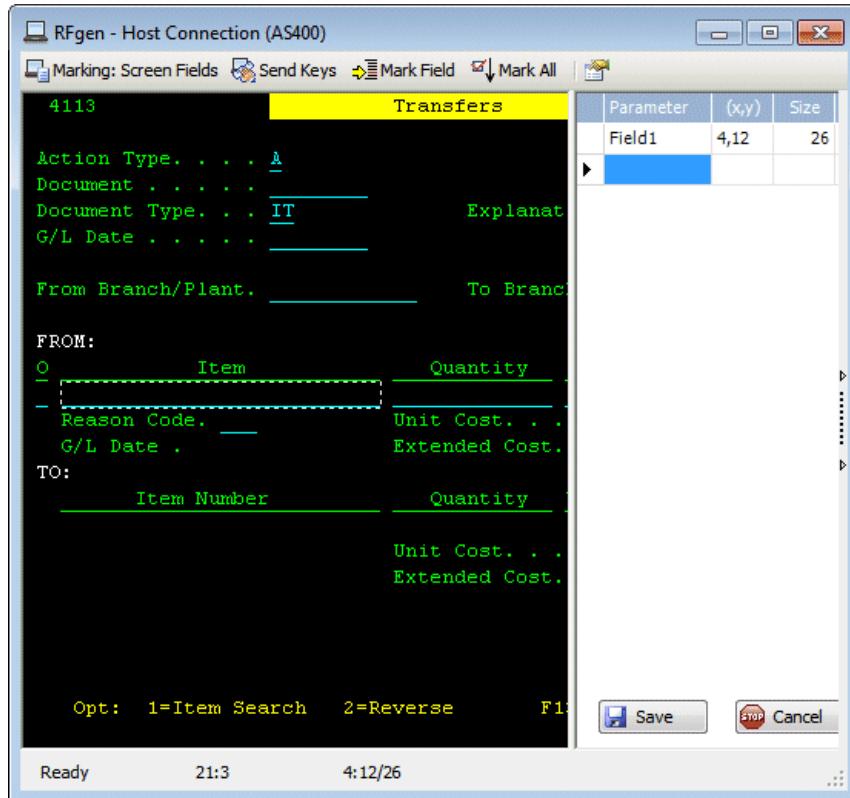
## Building a Host Transaction Macro

The Host Transaction macro contains just the recording of the sample transaction. This macro is linked to the Host Screen macro so when called it can determine the navigation steps. Under the Transactions section create a new macro, give it a name and description, select Host Transaction macro type and link it to the Host Screen macro previously recorded.

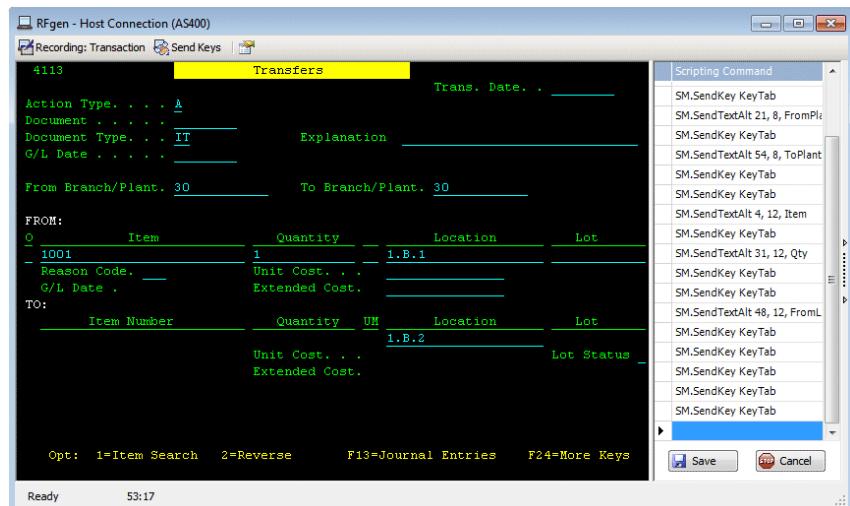


Click the first toolbar button above the parameter grid to navigate the host screen to the correct location.

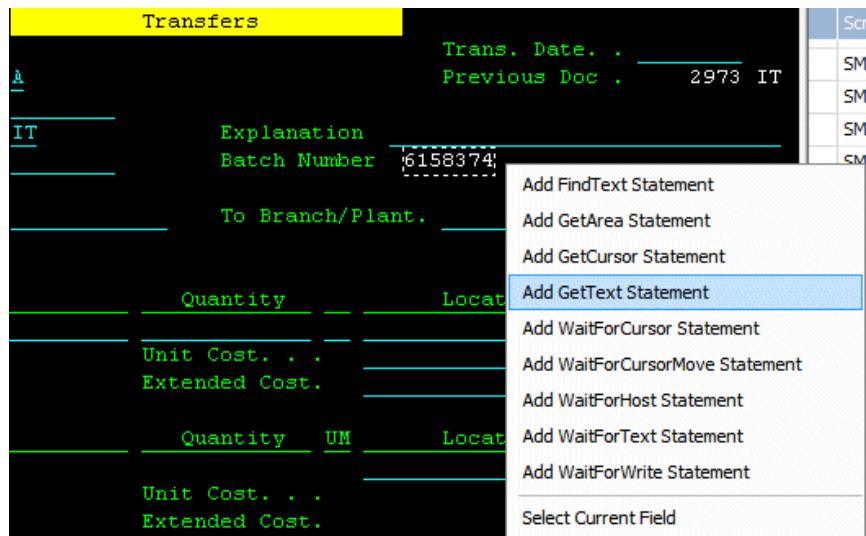
Next click the second toolbar button to mark all the fields on the screen that will be required for data entry.



Finally click on the third toolbar button to record a sample transaction.



After recording and submitting the data there may be indicators that it was successful or possibly failed. In this case there is a Batch number generated if the submission was successful so I select the batch number area and right-click to add the GetText command. In the code I will make sure something is retrieved. If nothing is returned then the bottom of the screen will contain an error message and the same GetText command can be used at those coordinates to get the text and display it to the user in a message box.



After the recording there may be a need to clean up the code if extra keystrokes were used and not necessary.

The screenshot shows the RFgen software interface for creating enterprise transactions. The main window displays a VBA-like macro code:

```
Private Function Transaction(ByRef Item As Variant, )
    On Error Resume Next
    '
    Dim iCol As Integer
    Dim iRow As Integer
    Dim sText As String
    Dim bSuccess As Boolean
    '
    SM.SendKey KeyTab
    SM.SendKey KeyTab
    SM.SendKey KeyTab
    SM.SendKey KeyTab
    SM.SendKey KeyTab
    SM.SendKey KeyTab
    SM.SendTextAlt 21, 8, FromPlant
    SM.SendKey KeyTab
    SM.SendTextAlt 54, 8, ToPlant
    SM.SendKey KeyTab
    SM.SendKey KeyTab
    SM.SendTextAlt 4, 12, Item
```

To the right, the "Transaction Properties" dialog is open, showing the following settings:

Item Name	TransferData
Description	Transfer Data Entry
Project Group	[dropdown]
Macro Type	Host Transaction
Link To	Transfer

Below the properties are two tables: "Parameter" and "(x,y)".

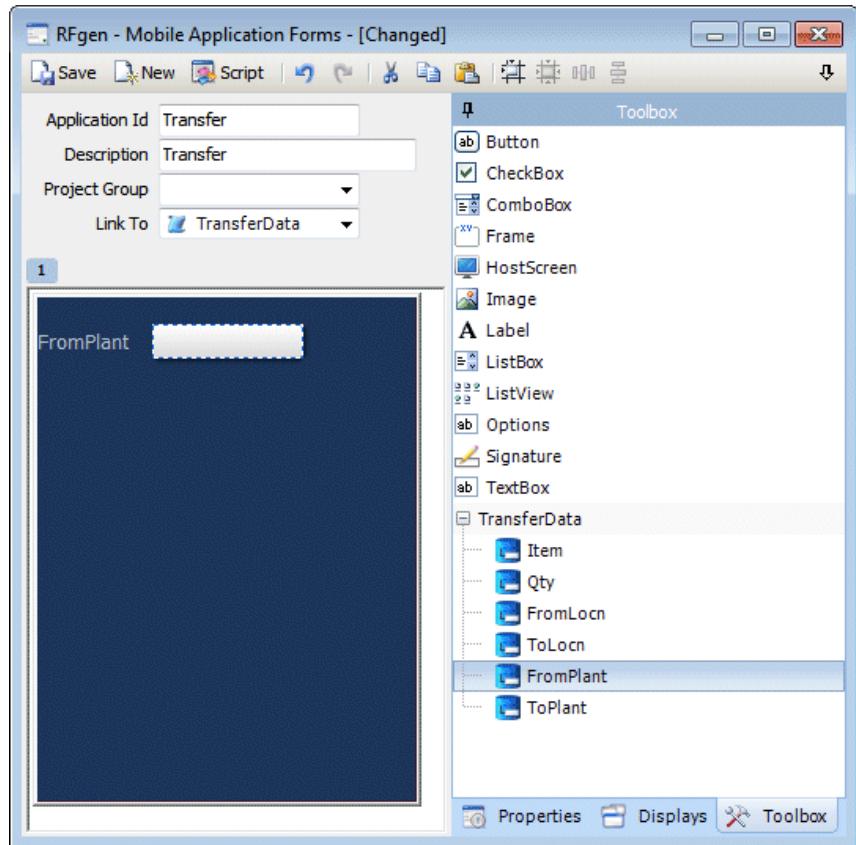
Parameter	(x,y)	Size
Item	4,12	26
Qty	31,12	13
FromLocn	48,12	20
ToLocn	48,17	20
FromPlant	21,8	12
ToPlant	54,8	12

	(x,y)	Size

Notice that there are several tab keystrokes but the SendTextAlt command references X,Y coordinates. This makes all the tab characters unnecessary and worth deleting.

Now click on the Play toolbar icon to test the complete data entry portion of the macro assuming you can continue to reuse the same sample data.

Save and exit the Transaction Macro design screen. The last step is to create an application that links to this macro.



Creating a new application and linking it to the data entry macro provides all the marked fields in the toolbox window. Drag them on to the screen, save the application, place the new application on a menu and test the result. Set the host screen itself either to the main menu or the login screen to test all macros working together.

This application does not require any VBA scripting because the macros recorded all the steps. There is the option of taking the code out of the macros and placing them directly in the application itself if more direct control is required. An example might be the login screen on the host requires named user instead of a generic user. In this case the recording can be taken and placed in the RFLogin form. However in doing so the Host Screen macro does not have a Start Menu macro to rely on so the Host Screen and Host Transaction macro code would need to be placed in the application itself and the macros deleted.

# Recording Options

While recording a host macro, you can highlight a region or field with the mouse and right-click on the highlighted area. A popup menu appears with the following selections. These options are used to add control statements to the current macro during the recording process.

- Add FindText Statement
- Add GetArea Statement
- Add GetCursor Statement
- Add GetText Statement
- Add WaitForCursor Statement
- Add WaitForCursorMove Statement
- Add WaitForHost Statement
- Add WaitForText Statement
- Add WaitForWrite Statement

---

- Select Current Field

**Add FindText.** SM.FindText is used to determine if a specified text string is currently displayed on the host screen. It can be used to look for the text in a specific screen location or to search the entire host screen for the text. See Screen Mapping Extensions for details.

**Add GetArea.** This command gets the text off the screen in any rectangular area. With the option to trim the result, selecting a column of data from the screen, parsing it and using it is much easier than getting all the data with several SM.GetText commands.

**Add GetCursor.** Used to determine where the cursor is currently located on the host screen. See Screen Mapping Extensions for details.

**Add GetText.** This selection adds SM.GetText which is used to retrieve text from the host screen at the column/row position established by the area highlighted by the mouse. See Screen Mapping Extensions for details.

**Add WaitForCursor.** This does the same as SM.WaitForText, only with the cursor; i.e., the script waits until the cursor reaches the specified location, before executing the next statement. See Screen Mapping Extensions for details.

**Add WaitForCursorMove.** This function is also used to time your commands to the host session. With this command, you specify only an amount of time in seconds. If the cursor has changed positions within that time, a True result is returned. Otherwise, it will timeout and return False. See Screen Mapping Extensions for details.

**Add WaitForHost.** SM.WaitForHost is used to time your commands to a vt220 host session. With it, you can delay sending text or keys to the host session, or retrieving data from the host session until the host has responded to the last command sent. See Screen Mapping Extensions for details.

**Add WaitForText.** Adds an SM.WaitForText statement to the macro, with the column/row position and the length being established by the area highlighted by the mouse, i.e., the script waits until the text appears at the specified location. See Screen Mapping Extensions for details.

**Add WaitForWrite.** This function waits for a specified number of seconds for data to be entered at a specific location and returns a True or False. If data was written within the wait time, True is returned. If the number of seconds expires first, False is returned. See Screen Mapping Extensions for details.

**Select Current Field.** Selects the current input field and records its attributes.

## Building a Screen Mapping Application

Once the recorded macros are built, there are two ways they can be implemented.

Create an application with data fields, collect and validate the data and use the Embedded Procedure object to pass that data to the Host Screen macro. (In the code window there is a right-click option to insert embedded code. Select Transaction Macros as the data source and then pick the appropriate Host Screen macro. See the Embedded Procedure section for more details.) The macro passes back either a True or False (based on setting the function name “Transaction” equal to one value), indicating the success or failure of the macro to complete its script. Alter the Host Screen macro’s script to send back an appropriate Boolean value. This method isolates the code responsible for interacting with the host system which is ideal for version control and frequent updates to application code unrelated to the host system.

Take the code generated in the macro and place it in the application itself. This gives the application total control and has another advantage. If the login screen is a host screen that must allow different user credentials, then the solution cannot rely on the automatic logging in and navigation to the main menu. An application can be created that collects the user credentials and screen maps the data to the host login screen. Having already recorded the macro, that code can be placed in the application directly and the macro may be deleted to avoid confusion. Taking this path requires that all applications be responsible for navigating and populating screens and fields on the host because the host is not automatically taken to a main menu, a generic starting point for the Host Screen macros. Since the Host Screen macros are linked to a Start Menu macro and they have been replaced by an application that performs that task after a custom login, Host Screen macros will not work. The solution is to create a Transaction macro with the same code, since it does not rely on links to previous macros.

# Mobile Enterprise Service Management

When started, the Server enables multiple communication sessions between your server and your remote devices (up to the number of authorized users for your particular installation). For telnet clients, if other telnet servers are activated on your server, then they must be deactivated, or the server must be reconfigured to listen on a port other than the standard telnet port 23. The menu bar 'Configuration / Service Configuration' may be used to specify a different port. The service will administer to all types of clients (Telnet, GUI-based devices, XML, Vocollect, etc.) simultaneously.

The Mobile Services Manager is the graphical user interface to the service that administers the remote devices. It allows you to:

- Start and stop the service
- Authorize permanent usage of the system
- Configure the system for running as a Windows task or as a service
- Start the Enterprise Dashboard
- Start the Transaction Management Dashboard
- Broadcast a message to all online users
- Configure the environment and connections
- View event logs



← Note the Mobile Services Manager icon in your Windows system tray

After installation, each time you boot your system, the icon (shown above) will appear on your windows task bar. Right-click on the icon and the following menu will appear.



Here, you may open the Dashboard if the service is currently running and Start, Suspend or Shutdown the service. If you stop the service, the Dashboard will automatically be stopped.

In order to start and stop the Server from the command line, navigate to the install directory using the DOS 'dir' and 'cd' commands. Then use the following commands:

```
rfscm500 -startup  
rfscm500 -shutdown
```

Load Balancing runs as part of the service and is configured by letting each server know of other servers. Once this list is complete in each server, load balancing is automatic. Even though each server is licensed for a specific number of licenses, the combined number of licenses is the total allowed even if one of the servers should fail. For a period of seven days the remaining servers will accommodate the total number of licenses before reverting back to the number of licenses it was originally designed to run. Note that telnet connections and Vocollect connections are not load balanced.

When you click 'Open' the Service Manager (or double left-click the system tray icon), the Service Manager window shown below will appear.



The screen shows version information, service configuration settings and lists the authorized connectors and user count. This is also where you may authorize the product. To Start or stop the service, flip the switch to the on or off position. Note that when remote data collection devices are active, stopping the service will terminate all device processing.

If there is a **Heap Size** warning an icon will appear in the System Tray, here on the server dashboard, and also in the menu bar of the Studio. This means that for the authorized number of users, the Windows Heap memory for services may not be sufficient.

The System Database, Configuration, Connections and Event Logs menu items are all identical to those options in the Mobile Development Studio. They are included here in the service management screen in case a change is required that does not warrant prior testing. Note that the service may stop and restart depending on the change made.

## To Permanently Activate the Service

The service will not run longer than two hours without a permanent authorization code. To test longer than two hours, the service will need to be stopped and restarted. To authorize the service for permanent

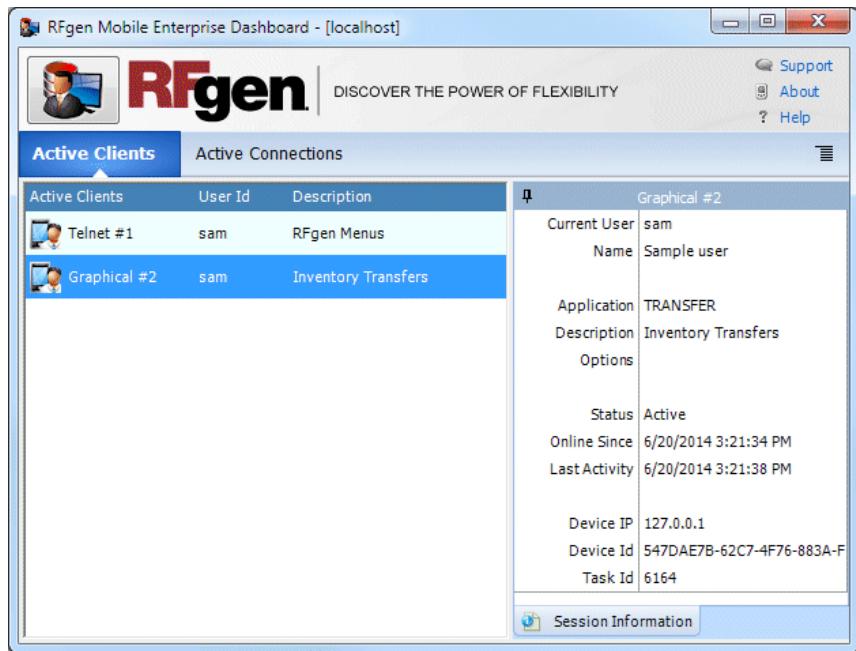
operation, click on the Authorizations button. A window similar to the following will appear.



Enter the Customer Id, Serial Number and click the Web Authorization but only if the server has never been authorized or was once authorized and the System Id has not changed. Depending on System Id being generated by the software and previous System Ids having already been authorized, there is a chance a call must be made to RFgen to get an authorization code. This is intentional. The Authorization Code is unique to this RFgen installation. Licensed users may obtain an authorization file by calling or emailing RFgen Software support department (telephone: 916-939-4065 or email: 'support@rfgen.com').

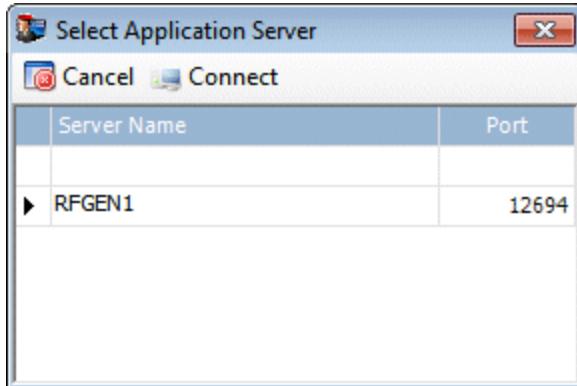
# Mobile Enterprise Dashboard

The dashboard allows the administrator to view and manage the remote sessions running under the Server. A window similar to the one shown below will appear when the administrator is started.



As devices log in, they are displayed in the window. If there are other computers on the network, they may also telnet into the system as data entry devices. This capability may be used as a system resource. For example, a hardwired (networked) user who connects to the system will receive the same screen that appears on the screens of remote devices.

To point the Enterprise Dashboard to a different server, click on the big icon in the upper left corner.



Select a server and click Connect to view that server's list of connected clients.

## Active Clients View

Queue sessions are shown for each queue that is setup. Graphical and Character sessions are displayed for each connected user and represent the type of device they are using.

This view appears when the dashboard is first started. As data entry devices log in, they are displayed in this main window. The window displays an icon for each device logged in and a right pane can be expanded to show details of that client's session. The details are:

**Current User** – is blank until a user logs in and then shows the user ID and name of the operator.

**Name** – The full name of the logged in user

**Application** – shows the menu or application screen name currently being viewed by the user.

**Description** – the description of the current form.

**Options** – any passed in parameters to the current form from the menu

**Status** – shows either Disabled or Active depending on the suspend status of the client connection.

**Online Since** – shows when the connection was established.

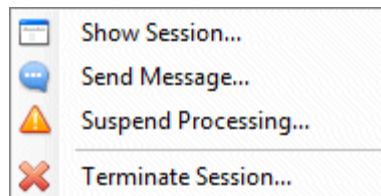
**Last Activity** – shows when the very last keystroke was made by the user.

**Device IP** – is the assigned IP address of the device.

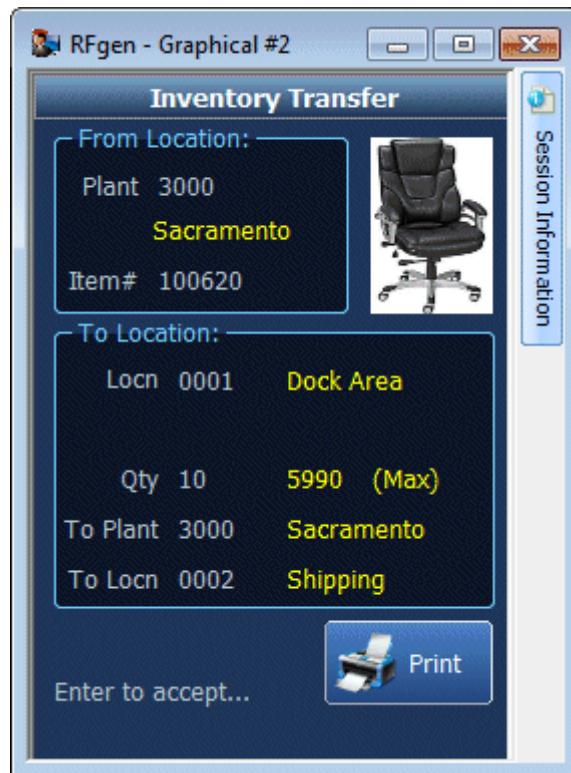
**Device Id** – This will be a GUID identifying graphical devices since in some environments the IP address alone is not enough to uniquely identify a client session.

**Task ID** – is the process identifier of the client session executable that can be located in the processes list of the Task Manager.

To monitor or interact with an active client session, simply right-click on the client session. A selection menu will appear.

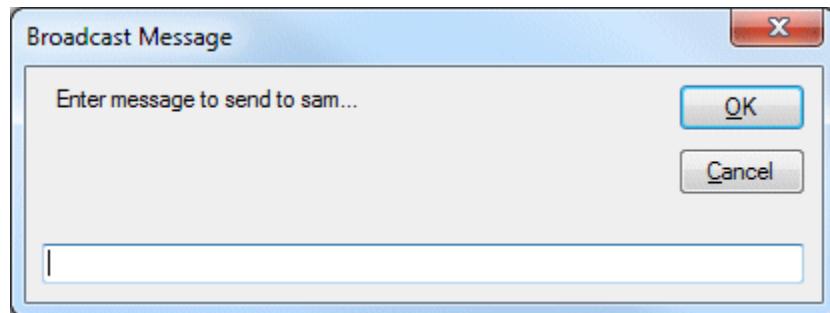


If you click on the **Show Session** selection, the selected client device screen window will appear.



Here, the remote device is being displayed. While the session window is open, all activity for the device will appear in the window. To 'take control' of the session, click inside the screen display area and interact with the prompts.

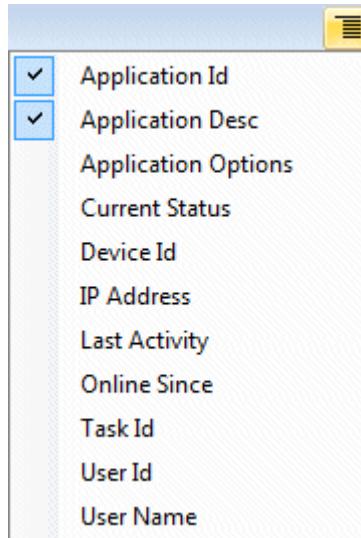
**Send Message** allows a user at the system console to send a message to the device user. The following message window appears.



To temporarily stop this session from collecting data, choose **Suspend Processing** from the menu.

To terminate this session, click **Terminate Session**. This action will terminate the communication session for the remote device.

On the far right there is a toolbar button to configure the displayed columns. The displayed columns offer an alternate view of the tree data in a record style rather than a grid pane. The following columns are available for selection.



All of this information is also available Show Session Information pane.

## Active Connections View

Clicking on the Active Connections view displays a window that contains each data connection in use. Entire connections or individual clients attached to a connection can be managed by right-clicking on the item.

The screenshot shows the RFgen Mobile Enterprise Dashboard interface. At the top, there's a logo with a person at a computer and the text "RFgen" in large red letters, followed by "DISCOVER THE POWER OF FLEXIBILITY". To the right are links for "Support", "About", and "Help". Below the header, there are two tabs: "Active Clients" and "Active Connections", with "Active Connections" being the active tab. On the left, under "Active Data Connections", there are sections for "Server Managed Connections" (which includes "Access Transaction Database - TM" and "Connection #1", which is highlighted with a blue selection bar), and "Client Managed Connections" (which includes "SAP - SAP" and "Session #2"). On the right, a detailed view of "Connection #1" is shown in a table:

Connection #1	
Status	Active
Online Since	3/12/2012 2:31:46 PM
Last Activity	3/12/2012 2:31:46 PM
Average Execution	0 seconds
Maximum Execution	0 seconds
Task Id	1480

For each client session there are also the connection statistics that may be viewed in the Session Information pane. Some applications may only take advantage of a few or just one connection depending on the work being performed. The data connection statistics are:

**Status** – shows either Disabled or Active depending on the server's ability to make the connection to the backend database or system.

**Online Since** – When the connection was last established

**Last Activity** – When the last use of that connection took place

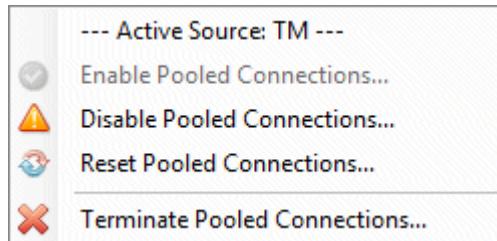
**Average Execution** – is how long, on average, each request to the backend database or system takes.

**Maximum Execution** – is the time taken by the longest running single request to the backend database or system.

**Task Id** – a unique process ID used by Windows so that it may be found in the Windows Task Manager

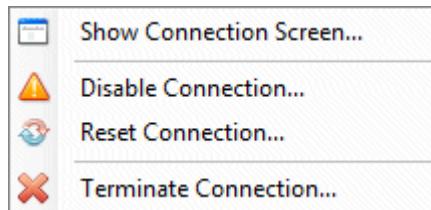
If Transaction Management is in use, the ability to suspend and resume processing of a queue is done here by right-clicking on the proper queue and choosing the proper option.

Right-clicking on a Server Managed Connection (meaning that pooling is enabled) the connector itself will display this menu.



For the selected data connection, the administrator may disable or enable the pooled connections being shared among the thin client mobile users. Resetting the pooled connections is a quick way to destroy and recreate handles to the backend connections with a minimal interruption to the users. These options should be used carefully and only in emergency situations. The Terminate Pooled Connections option tells the server that all pooled handles should be permanently destroyed. Again, this allows the administrator to have control over the environment, but should be used only when necessary.

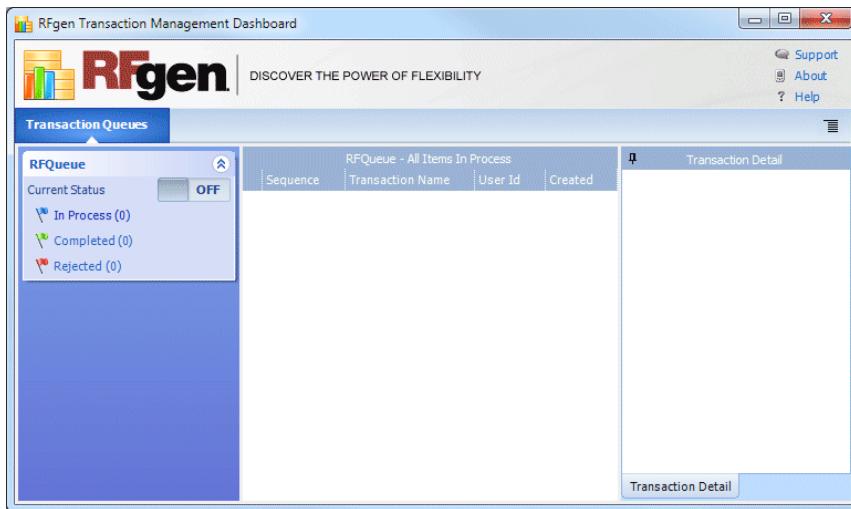
Right-clicking on a used session will display this menu.



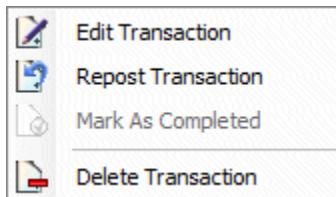
These options will only affect the client device using this session and not all clients sharing the connector.

# Transaction Management Dashboard

The Transaction Management Dashboard is used to manage the queues and queue processing. Each queue can be started or stopped individually and completed or failed transactions can be edited and resubmitted.

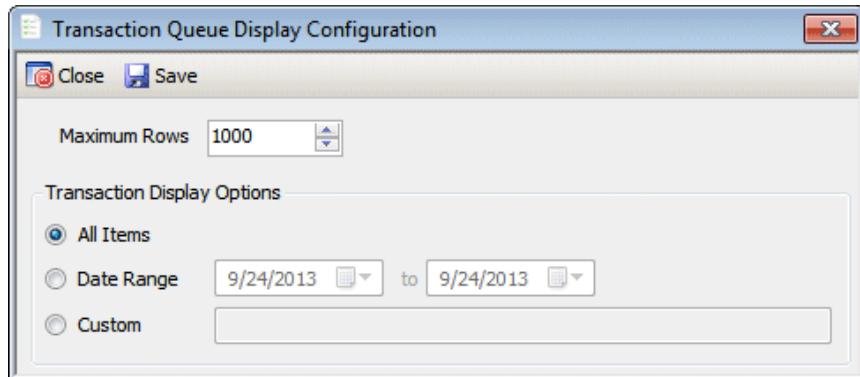


Three types of logs are available: “In Process” transactions are data collection entries waiting to be posted to the host application (typically because the host has been offline or not available); “Completed” transactions and “Rejected” transactions may also be displayed. Transactions may be edited, reposted, marked as completed or deleted by means of the right-click menu option from the desired record.



The Transaction Information pane on the right side will show details about the queued transaction as well as the values of the passed in parameters.

The Display Options are used to narrow down the records being displayed in this window. Click the toolbar button on the far right to get this configuration screen.



Over time the list of completed transactions can become very large.

**Maximum Rows** will limit the display to the first configured number of entries. To see the most recent entries, use the data range option and set the Maximum Rows to a high value.

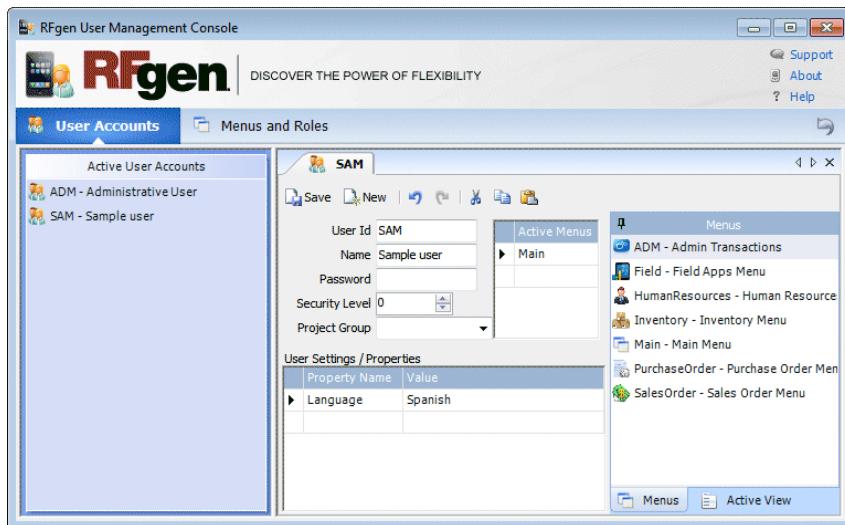
Transaction Display Options – **All Items** shows an unrestricted list of entries and **Date Range** will limit the entries to a date-based on their created date.

The **Custom** option is an ability to specify your own Where clause for the lookup. The actual names of the fields in the Queue database must be known as well as the type of field. An example would be:  
where SeqNo = 1

(See *TM.GetItemsEx* for examples of table fields and types.)

# User Management Console

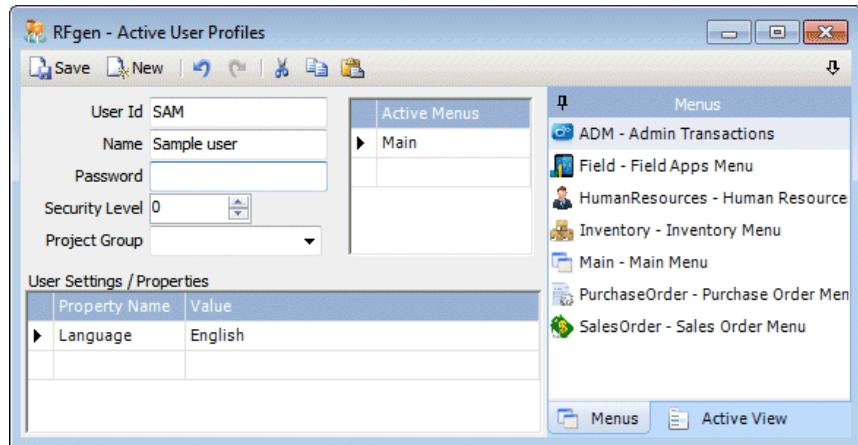
The User Management Console contains the same edit capability as the Mobile Development Studio for the users and menus. The purpose of this console is to allow a person to manage or maintain the users in the RFgen network without providing access to the forms or code. If users should need to be added, modified or deleted and potentially their menus changed this console will work. If additional changes are required to applications, code, macros or resources then the full development environment is required.



## User Accounts

Right-click on an existing user or on the title to add a new user. Double-click or right-click on a specific user to make additional changes.

Double-clicking on an existing user opens the User Editing window.



Here a user code of 'SAM' is illustrated. A password is optional for a user account. SAM's startup menu is 'Main Menu'. Security Level is a numeric value between 0 – 100 that will be compared to the menu's required security level before allowing that user access to the following menus or forms. Other menus are embedded in the Main Menu for SAM's use (created using the Menus tree).

To assign a menu to the user double-click one or more menus to add to the Active Menus list.

The **Project Group** property allows the users to be grouped together into categories such as "Admin" or "SecondShift" or any other named grouping. This has no impact on production. It is just an option for making the management of the development database easier.

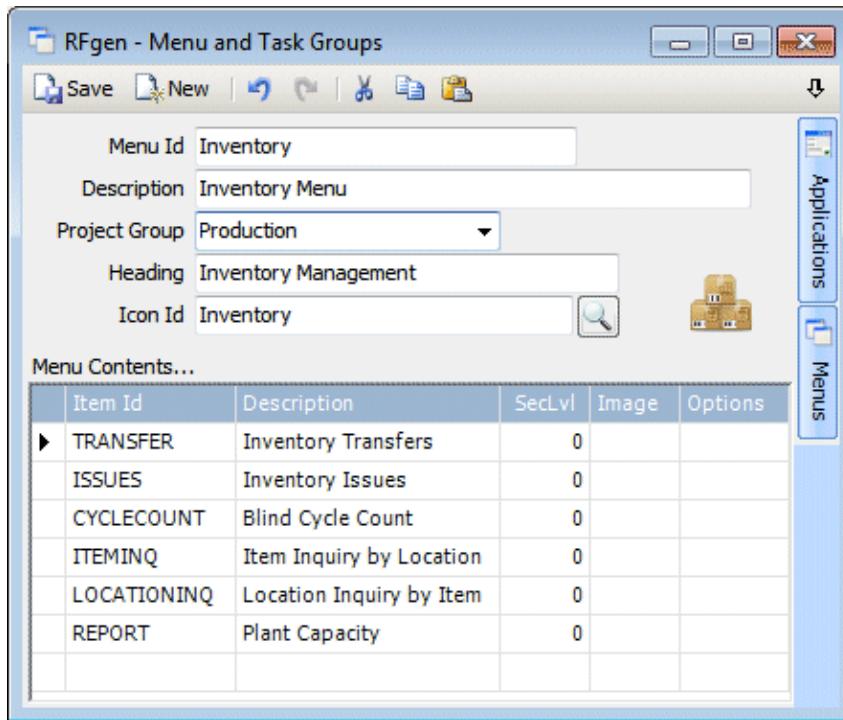
The Optional User Settings / Properties section allows the administrator to include any property and value for the selected user for the purposes of retrieving that data at run time. This has no effect on the user logging in. This data can be retrieved using the VBA extension command App.UserProperty.

When done, click the Save menu item.

## Menu Tab

Right-click on an existing menu or on the title to add a new menu. Double-click or right-click on a specific menu to make additional changes.

Double-clicking on an existing menu opens the Menu Editing window.



Menus are groups of existing applications and other menus that will be made available to RFgen Users who are assigned available menus.

The **Menu Id** is the name of the menu that is used when assigning them to users. The **Description** is required and only shows in the Menu list.

The **Project Group** property allows the menus to be grouped together into categories such as "Test" or "Production" or any other named grouping. This has no impact on production. It is just an option for making the management of the development database easier.

The **Heading** value will be placed at the top of the menu and is optional. The **Icon Id** references an Image resource to represent this menu in case the menu is presented to the user with graphical elements. Graphical menu display options include an icon and text description, Windows desktop style and a mobile device graphical button view.

To select an item to appear on your menu, right-click on the first column of the row you want to insert into or delete. The user may also just press

the Insert or Delete keys on the keyboard. When the correct row is selected double-click the items in the Applications or Menus lists. When done with your selections, click on the Save menu item.

In the case where one form is used for multiple purposes, variables can be defined in the Menu grid itself. Then those variables can be referenced when the application is loaded to determine how to use the application.

In a case where you want the user to select from two variations of the same application, add the application to the menu twice, give them different descriptions and then use two different command **Options**. Using the format –VARNAME=VALUE will create variables with values that can be referenced at runtime. The **Image** column allows for multiple instances of an application to have different menu icons.

In the Form\_Load event, set a global variable equal to App.GetValue("VARNAME") and based on the result, show or hide prompts or change the logic of the application. To add more than one variable to the command Options add a space and repeat as shown below.

Making the application name a dash, supports comment lines within a menu.

Menu Contents...					
	Item Id	Description	SecLvl	Image	Options
	TRANSFER	Inventory Transfers	0	Transfer	-PLANT=10 -LOCN=DOCK
	ISSUES	Inventory Issues	0		
	CYCLECOUNT	Blind Cycle Count	0		
	-		0		
	-	[ Inquiries ]	0		
	ITEMINQ	Item Inquiry by Location	0		
	LOCATIONINQ	Location Inquiry by Item	0		
	REPORT	Plant Capacity	0		

In this case, what is displayed on the menu is a blank line and then a comment line indicating that the following list of items are their own group. The user has no ability to select label entries on the menu. The highlight bar will skip over these entries. This does not do anything if in the Button or Desktop menu modes.

# Mobile Devices

Mobile devices can be used in a few different ways, as a thin client or mobile client device. Thin client refers to having a wired or wireless connection from the mobile device to the server while using an application.

## Windows Desktop Client

Mobile devices are generally Windows CE, Windows Mobile, Android, or iOS based devices but others such as tablets or full PCs can run a full version of the Windows OS. The Windows Desktop Client is exactly the same client and configuration program but compiled to run on a full Windows system.

## Thin Client

While in thin client mode, the user interacts with a session running on the server. Since all the processing takes place on the server, the mobile device cannot be a point of failure or lose data. If the wireless device goes out of range of the network, the mobile device screen will appear to stop since the server cannot order the screen to refresh. The client on the mobile device will continue attempts at reconnecting and will then resend the last piece of data entered. Unlike standard telnet, RFgen has added a “Guaranteed Packet Delivery” system to the protocol to ensure no loss of data and an always-synced application. The advantage to the thin client is real time updates to backend systems as well as complete validation data available to ensure the collected data is as accurate as possible. The disadvantage is the need for a wired, wireless, or cellular connection available while collecting data. This client does not require any authorization process.

## Mobile Client

Mobile mode is where the mobile device contains a copy of the applications, users, menus, and database values for validation as well as storage. Keep in mind there will be space limitations on the device. The mobile client can be implemented in two ways based on the Startup Mode configured:

1. Connected to the server as in a thin client

2. Disconnected from the server where everything takes place on the device
3. Roaming in and out of range of the server where database access is first attempted against the server, but if the client does not have a connection the local database is used.

When the mobile client is configured to start in a connected state, it is in essence a thin client. The Batch Failover property will allow this mode to either switch over to a disconnected state or remain in a connected state and wait for connectivity to return when wireless coverage is lost.

When a mobile client is configured to start in a disconnected state, all aspects of the data collection operate on the device. Applications, menus, users and others are loaded from the device, validation data on the local database is used for validation and completed transactions are queued on the device pending an upload to the server. Only an RFgen server can extract what was stored on the mobile device. The one distinction between connected (or Thin Client) and disconnected is that an extra option must be provided to tell the mobile client to upload the data to the server when it is in range. See the Device object methods to accomplish this.

When a mobile client is configured to start in a roaming state, then the Batch Failover option has no effect. If the client detects connectivity to the RFgen server, database related commands, embedded procedures and macro calls / queuing are automatically redirected to the server. If there is no connectivity to the RFgen server, all activity is directed locally just like a disconnected state. To keep data integrity, it is recommended that the Server.SendQueue command be placed in the RFgen\_OnConnect event. That way all queued work will go to the server before more transactions can be performed that will automatically go to the server. It is also recommended that the Server.SyncApps command also be placed here to keep the applications, menus, users and others always up to date.

The mobile client does require an authorization code to function longer than the demo period. This is described in the Mobile Client section.

## Mobile Device Management

On the mobile device there is an icon  that represents the MDM (Mobile Device Management) service. Clicking on this icon may give various options depending on the implementation. The purpose of this

service is to allow requests from the server to be performed on the device when it is not in the cradle.

Some core capabilities include the ability to detect that a server upgrade has occurred and to auto-update the client environment. Further, if you've deployed in a mobile (off-line capable) environment, MDM provides support for "Application Synchronization" requests. In this scenario, if you've changed any applications that are in the mobile device's profile, it can automatically detect the application / profile changes, build a custom deployment package and remotely update the device – all without the intervention of IT personnel.

In the case of the Windows Desktop Client, the MDM is installed when the Windows Desktop Client is installed. Then when using the Mobile Device Installation Utility to deploy a Thin or Mobile client profile to the desktop client, select the active device, platform of Win32, choose the Install Profile name and click the Install on Device menu option. This is the same procedure for mobile devices as well.

## Auto Upgrade Clients

The server supports automatically upgrading the mobile and wireless clients, including desktop clients through the wireless environment without user interaction when the server software is upgraded. First, be sure the Mobile Clients installation package is installed on the production server so it will have the necessary EXEs to draw from when sending the new files to the mobile device. Second, each client needs to be running the MDM service so a process outside the client process on the device can execute the commands to replace the client application. The MDM is installed the same way as the profile.

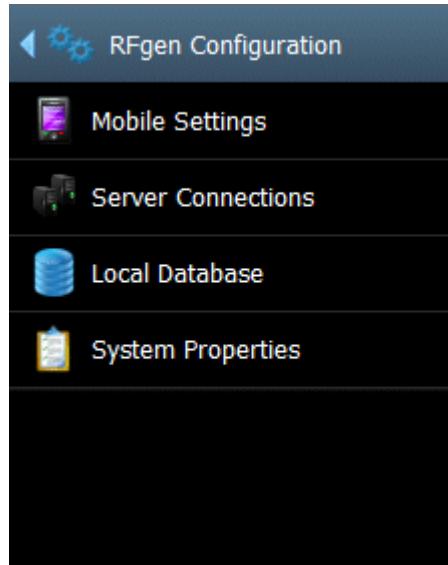
Finally, when a client detects that it is necessary to upgrade the user may be given a choice to perform the upgrade.

## RFgen Mobile Configuration

This executable is installed in the RFgen Mobile folder to allow the user to customize the emulator differently than the options chosen in the Mobile Development Studio at the time the CAB file was created. On a Windows platform start by clicking the Windows Start button on the mobile device and choosing the program. Other platforms will likely have an "App" installed.



Select the RFcfg 5 program from the list.



There are multiple configuration pages available. The first one is the Mobile Settings page.

Mobile Settings	
Client Startup Mode	Connected
Failover to Batch Mode	<input type="checkbox"/>
Active Device Profile	
None	
Check For Updates	
Never	
License Information	
Identity	
50-4EFD-91A7	
Authorization Code	
None	
Device Settings	
Language Set	
1033 - English, United States	
RTL Display	<input type="checkbox"/>
Full Screen	<input type="checkbox"/>
Virtual Keyboard	<input checked="" type="checkbox"/> OFF
Capture Keys	<input checked="" type="checkbox"/> OFF
Scale Display	
1	
Display Theme	
None	
Configuration Password	
None	
Desktop Password	
None	

## **Client Startup Mode**

For Mobile Installation Types, this option determines if the Mobile client will start in a connected state or a disconnected state.

## **Failover to Batch Mode**

For Mobile Installation Types, this determines if the mobile client will ask the user if they wish to move from a connected state to a mobile mode when the thin client detects that the device is no longer in communication with the server.

## **Active Device Profile**

This property is the name of the profile to be used when syncing a mobile client.

## **Check for Updates**

This property has three options, Never, On Connect and Once Daily. On Connect means that every time the client (thin or mobile) makes a connection to the server it will synchronize. For thin client just the theme and some images will be updated. For mobile mode all the solution objects for the specified profile will be exchanged.

## **License Information Group**

### **Identity**

The Identity is the parameter generated by the device that will be used to permanently authorize it for use in the mobile mode.

### **Authorization Code**

This field will automatically fill in when the authorization process is started and completes successfully. This process can be started by bringing up the menu strip, clicking the configuration icon, selecting the Options menu option and selecting the Authorizations menu item.

Manually entering a code here and saving the configuration will also work if access to the server or the server's Internet access is not available. To obtain a code call support with the Identity value and they will provide the code.

## **Device Settings Group**

### **Language Set**

The default is English. Any language may be chosen from this menu.

### **RTL Display**

This option is for displaying all elements on the form in a right-to-left format. Languages such as Arabic and Hebrew are examples.

### **Full Screen**

This option determines if the display on the mobile device is in a window (smaller) or if the application is maximized for the screen (larger).

### **Virtual Keyboard**

This feature will automatically open the SIP (Soft Input) keyboard for other configuration fields that require text input like the authorization code or server DNS name.

### **Capture Keys**

This option if enabled will transmit all keystrokes to the server and not the operating system. If the operating system has taken the F3 key for example to control volume then it would not get sent to the client unless this option is enabled. Other devices use a button to scan a barcode. If this were enabled that button to scan may not function so this feature would need to be disabled. The use of the feature depends on the requirements of the device and application.

### **Scale Display**

This option is designed to fix a problem on some devices where they do not report their DPI (dots per inch) correctly. If the high resolution screen forces the application screen into a very small space, change this value from 1 to 2 to double the rendering of the application screen.

### **Display Theme**

This is the theme resource to be used on the device. It contains all the look-and-feel display options.

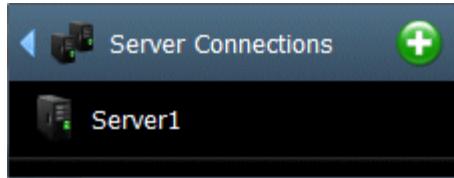
### **Configuration Password**

Setting the configuration password will then prompt the user for that password anytime the configuration options are accessed, either from the CFG executable or the client itself.

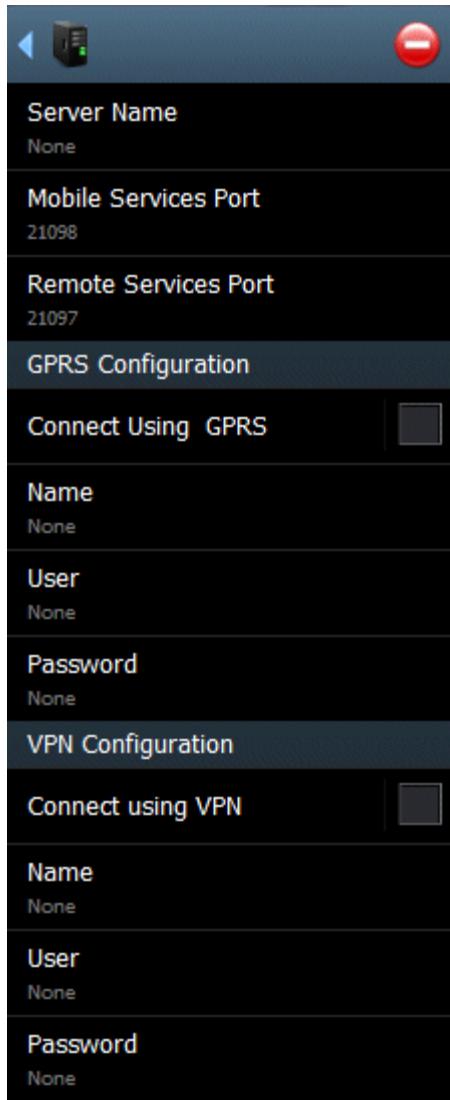
### **Desktop Password**

This option is designed to prevent unauthorized users of the mobile device from leaving the client and returning to the Device desktop.

### **Server Connections Group**



If no server connections exist, click the Plus icon to add a server. After entering the data you can back out and add additional servers if desired.



### Server Name

A telnet client can connect to any server. Adding to this list prompts the user to choose which server they wish to connect to before starting the login process. The name can either be the DNS name or the IP address.

### Mobile Services Port

This is the port number the server is using for graphical telnet traffic.

## **Remote Services Port**

This is the port number the server is using for passing data between the client and server.

## **GPRS Configuration**

If the device is going to take advantage of the cellular network for updates back to the server, these settings can be used to automatically establish the connection when commands are used to sync or send data.

### **Connect Using GPRS**

Toggle this option if you want the client to establish a cellular connection when server updates are attempted. The cellular connection setup must already be created on the device.

#### **Name**

The name of the connection is the same name used to create the connection in the mobile device's operating system.

#### **User and Password**

A User and Password may not be necessary. If the client must start the cellular session then reference it by Name.

## **VPN Configuration**

If the device is going to take advantage of a VPN for updates back to the server, these settings can be used to automatically establish the connection when commands are used to sync or send data.

### **Connect Using VPN**

Toggle this option if you want the client to establish a VPN connection when server updates are attempted. The VPN connection setup must already be created on the device.

#### **Name**

The name of the VPN connection is the same name used to create the VPN in the mobile device's operating system.

## **User and Password**

A User and Password may not be necessary. If the client must start the VPN session, then reference it by Name.

## **Local Database Group**

Usually only needed for Mobile clients (not Thin clients), this section specifies the details of the local database setup on the device.

### **Database Type**

This option is to specify if a local database is to be used or not. If a database table was included In the CAB file based on the profile then this option should read as database type selected in the profile.

### **Provider**

This field specifies which provider is used on the mobile device.

### **Database File**

The database file refers to where on the mobile device the file should be created. The file name by itself will be placed in the installation directory. Specifying a path such as APPS\RFSample.xdb will place the file in that location.

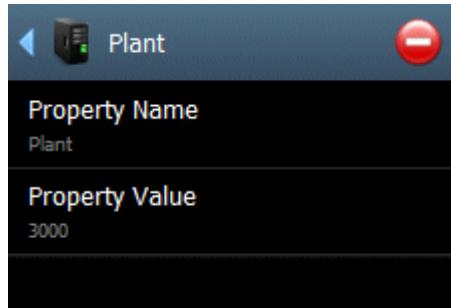
### **User**

If a user is required to access this mobile database, enter it here.

### **Password**

If a password is required to access this mobile database, enter it here.

## **System Properties Group**



## System Properties

Entering a property name and property value is like creating a global constant with a read-only value. For example, if this installation was designed for multiple warehouses but this particular installation was for a warehouse called 'Main Street' then entering the property name of 'Warehouse' and a value of 'Main' would allow the programmer to identify which warehouse was being used. The command used is `System.EnvironmentProperty`.

## RFgen Mobile Client

From the Windows Start button on the mobile device, click Programs and then the RFgen client. A splash screen displays the version of the client installed on the mobile device. This version should match the software running on the server.



After the splash screen, the RFLogin screen is the typical first form. In this case, the RFLogin screen shown is from the sample applications shipped with the software.



To show the soft menu buttons press the specific F key or perform an action that executes the language extension `MenuStrip.Show`. The configurable menu buttons appear as well as the client configuration menu option. Click the configuration menu option for features such as authentication, resetting the connection or exiting the application.

In the case of the mobile client, the menu may have the Authorization option. If the client has wireless access to the server and the server has Internet access (and the client is allowed to be authorized) then this option will authorize the client device and the menu option will disappear. The other options are to reset the connection between the client and server which makes sense if the client is wirelessly connected and exit the client.

This form can be bypassed and the user could be presented with a menu or another form if desired. If a desktop password has been configured, the user must enter that password before being exited back to the desktop.

# Visual Basic Scripts

Visual Basic for Applications (VBA) scripts allow technical personnel to provide additional functionality to RFgen data collection applications by responding to events using Visual Basic programming statements (scripts). VBA Scripts allow the developer to enhance the capabilities offered by standard RFgen applications and other objects. In fact, developers may take total control over the client device by responding to field/system events, handling all data display functions, and even sending SQL statements through the RFgen database connection.

Programmable events include 'field' events (scripting is related to the current prompt) and application events (scripting will be executed whenever this event occurs; e.g. when an application is loaded). In addition, built-in VBA 'extensions' give the developer easy access to manipulate the client device, and quick access to the connected database.

## Global User Defined Subroutines and Functions

To use global subroutines and functions: create a module file using the VBA Modules and put your 'global' subroutines and functions in that file. On the Properties tab in the RFgen Application window, you will see a list of available VBA modules. Set each module to TRUE that you want that application to be able to access.

You will then be able to call any subroutines or functions from these modules in your RFgen Applications.

Note: There are 2 built in modules that are automatically referenced by RFgen Applications. These are:

RFgen.bas – The RFgen.bas is typically used to contain functions and procedures that need to be accessible from any transaction

Win32.bas – The Win32.bas is typically used to store global variables.

# Using External ActiveX files in the VBA Environment

RFgen allows the incorporation of external ActiveX .EXE or .DLL modules by declaring an Object and using the CreateObject method.

Example:

```
Dim MyObject As Object  
Set MyObject = CreateObject("name.cls")  
' name is the name of the ActiveX module ' subsequent  
usage  
MyObject.property  
MyObject.method
```

The syntax is slightly different when using a DLL compiled using Visual Basic.

```
Dim MyDLL As Object  
Set MyDLL = CreateObject("ourDDC.DDCcom")
```

Where: ourDDC is the executable name DDCcom is the public class name containing the functions you wish to call.

## VBA Global Variables/Objects

Global variables and objects are available for use with RFgen, for storing and retrieving items related to a specific session.

The preferred method for using global variables is to declare the variables in the 'Win32.bas module, via the VBA Modules option. Any public variables declared in the Win32.bas module will be available to all RFgen applications and will maintain their values throughout the life of the RFgen session.

Examples:

```
Public sMyString As String  
Public iMyInt As Integer  
Public oMyObject As Object
```

# VBA Declarations

The VBA ‘General’ object is where the developer may declare variables that are available to all events in the current application. User defined functions or subroutines may also be entered here (see above).

# VBA Events

Field events are linked to and executed by individual data entry prompts in an RFgen application. Application events are global in nature (e.g., the application event occurs immediately before the corresponding field event). Edit properties are executed after the VBA event has finished. Using these events, the developer may respond to the users' actions as they choose. The events supported by RFgen are as follows:

## **Click**

The Click event occurs when the user clicks on a prompt or button with a mouse or pen. It is typically used to determine if a user has pressed a button on the screen.

Applies to: Prompts (graphical mode only)

Syntax: Click()

## **GotFocus**

The GotFocus event occurs as soon as the user (either moving forward or backing up) reaches the data entry field. This event is typically used to generate a default value that the user can either accept or reject.

Applies to: Applications, Prompts

Syntax: GotFocus(Rsp, AllowChange)

Rsp           (String) is the default value to display on the Client device.

AllowChange (Boolean) is set to True to allow the user to override the default, and is set to False to generate an OnEnter event bypassing user interaction at this field.

## **KeyPress**

The KeyPress event occurs when the user presses and releases a key on the keyboard. This event is typically used to capture incoming keystrokes and perform some action based upon user input. In addition, the device's telnet emulator must be capable of supporting ‘Character’

mode. If the emulator is set to transmit data in Line mode, RFgen will not see the data until the whole string is complete.

Applies to: Applications, Prompts

Syntax: KeyPress(KeyAscii)

KeyAscii (Integer) is the ASCII value of the character pressed on the client device.

## **Initialize**

The Initialize event occurs when an RFgen client is initially loaded. It will occur only once, and is typically used to initialize variables, open additional database connections or create links to ActiveX objects.

Applies to: Applications (RFgen.bas)

Syntax: Initialize()

## **Load**

The Load event occurs when the application is initially loaded. It will occur only once per form and is typically used to initialize variables.

Applies to: Applications (RFgen.bas)

Syntax: Load()

## **Lost Focus**

The LostFocus event occurs when the prompt that had the focus is giving it up to another prompt. This would occur after the OnEnter event is finished and before the next prompt's GotFocus event is executed.

Applies to: Application, Prompts

Syntax: LostFocus()

## **OnBackup**

The OnBackup event occurs when the prompt with the focus receives a command to go back to a previous prompt, as if the up arrow key was pressed.

Applies to: Application, Prompts

Syntax: OnBackup(Cancel)

Cancel (Boolean) is set to True if the backup movement should be stopped. Set it to False or do not change the default to allow the backup to continue.

## OnConnect

The OnConnect event occurs when the Mobile Client in a roaming state automatically discovers it has connectivity to a network and makes a connection to the RFgen server. The event only executes after 10 seconds of continuous connectivity to the RFgen server. To maintain data integrity it may be a good idea to include the Server.SyncApps and Server.SendQueue in this event.

Applies to: RFgen (RFgen.bas)

Syntax: OnConnect()

## OnCursor

The OnCursor event occurs whenever a cursor (arrow) key is pressed. This event is typically used to allow users to page through a list of data. The Up arrow is typically used to backup to the previous field (system default); however, to bypass system processing of this event, set Cursor = "" (null).

Applies to: Application, Prompts

Syntax: OnCursor(Cursor)

Cursor (String) is the character value of the key pressed. Possible values are ('U'p, 'D'own, 'R'ight, and 'L'eft).

Example:

```
Public Sub PartNo_OnCursor(Cursor As String)
    On Error Resume Next
    If Cursor = "U" Then
        'your logic
    End If
End Sub
```

## OnDisconnect

The OnDisconnect event occurs when the Mobile Client in a Roaming state is disconnected from the RFgen server because the client moved out range. This event does not execute when the Server.Disconnect command is issued.

Applies to: RFgen (RFgen.bas)

Syntax: OnDisconnect()

## OnEnter

The OnEnter event occurs when the user presses the Enter key on the Client device, or a default value specifies that no user input is allowed. This event is typically used to validate data entered, and/or adjust the display on the device. To reject the entry made by the user, set Cancel = True and RFgen will force the user to re-enter the field.

Applies to: Application, Prompts

Syntax: OnEnter(Rsp, Cancel, Message)

Rsp (String) is the value entered/scanned by the user on the Client device.

Cancel (Boolean) is set to False to accept the data entered and move to the next prompt, or is set to True to fail the edit check and force the user to re-enter data at the current field.

Message (String) is a message to display on the Client device.

## OnEscape

The OnEscape event occurs when the Escape key is pressed. This event is typically used to capture an incoming Escape keystroke and perform some action based upon user input.

Applies to: Application, Prompts

Syntax: OnEscape()

## OnFkey

The OnFkey event occurs whenever a function key (Fn) key is pressed. Function keys F1-F4 trigger pre-defined system events. However, to bypass system processing of these events set Fkey = '0' (zero) and RFgen will ignore the event.

Applies to: Application, Prompts

Syntax: OnFkey(Fkey)

Fkey (Integer) is the integer value of the key pressed. Possible values range from (1-10).

Example:

```
Public Sub Form_OnFkey(Fkey As Long)
    On Error Resume Next
    If Fkey = 5 Then
        ' In this example we are using the F5
```

```
' key to execute logic.  
' your logic  
End If  
End Sub
```

## OnInRange

The OnInRange event occurs when a Mobile Client notices that there is an IP address available on the network adapter. This event does not execute when using the Roaming Client because the OnConnect will execute anyway.

Applies to: RFgen (RFgen.bas)

Syntax: OnInRange()

## OnLocale

The OnLocale event occurs after OnConnect (only when a client application makes a connection) and passes in the locale number based on the client device's location. In the case of the United States, number 1033 is returned. Based on this value you may set global values or default login forms (See App.ChangeLoginForm)

Applies to: Application, Prompts

Syntax: OnLocale(nDeviceLocale)

nDeviceLocale (Long) is the value of the client device's locale.

Example:

```
Public Sub RFgen_OnLocale(ByVal nDeviceLocale As  
Long)  
    On Error Resume Next  
    Select Case nDeviceLocale  
        Case 1033  
            App.ChangeLoginForm("RFLoginEnglish")  
        Case Else  
            App.ChangeLoginForm("RFLogin")  
    End Select  
End Sub
```

## OnMenu

The OnMenu event occurs when the user clicks one of the Menu Strip buttons. If one of the default actions is clicked, no code is required.

Applies to: RFgen (RFgen.bas), Application and Prompt

Syntax: OnMenu (sMenuAction)  
sMenuAction      (String) is the Action name of the button clicked by  
the user

Example:

```
Public Sub Form_OnMenu(MenuAction As String)
    On Error Resume Next
    MenuStrip.Show(False) ' hide the menu after a click
End Sub
```

## OnOutOfRange

The OnOutOfRange event occurs when a Mobile notices that the IP address is no longer available on the network adapter. This event does not execute when using the Roaming Client because the OnDisconnect will execute anyway.

Applies to: RFgen (RFgen.bas)

Syntax: OnOutOfRange ()

## OnReadData

The OnReadData event occurs whenever data is successfully retrieved from a database by an RFgen application. This event is typically used to force an immediate repaint of the screen to display all linked prompts values to the user.

Applies to: Application

Syntax: OnReadData(TableName)

TableName    (String) is the name of the table that was successfully queried.

Example:

```
Public Sub Form_OnReadData(TableName As String)
    On Error Resume Next
    Screen.Refresh ' Repaint the current screen
End Sub
```

## OnRefresh

The OnRefresh event occurs when the screen is completely repainted (as when the user presses the F2 key). It is typically used to redisplay information to the user that was displayed via Screen.Print statements.

Applies to: Application

Syntax: OnRefresh()

## OnReturn

The OnReturn event occurs when the user returns from a called application that had its return flag set to True. It is typically used to process data from the called application.

Applies to: Application

Syntax: OnReturn(Name)

Name (String) is the name of the called application that is returning.

## OnRowColChange

The OnRowColChange event occurs when the focus moves between a column or row in a control that supports them.

Applies to: Application, Prompts

Syntax: OnRowColChange(Row, Col)

Row (Long) is the value of the row being selected

Col (Long) is the value of the column being selected

## OnScan

The OnScan event occurs when RFgen identifies a preamble string in the data stream coming from a client device. This event is typically used to validate that data was scanned, or to parse the data (PDF, RFID, etc.) into a more usable format. To reject the data scanned, set the ScanData = "" and RFgen will remove it from the data stream and prevent it from being entered into the current prompt.

Applies to: Application, Prompts

Syntax: OnScan(ScanData)

ScanData (String) is the value scanned by the user on the client device.

## OnSearch

The OnSearch event occurs only if a linked or unlinked text box has code in the OnSearch event and the user clicks on the generated search command button. This is only possible in graphical mode. Any VBA code can be placed in this event, even if it is not specifically related to searching a data source.

Applies to: Text prompts

Syntax: OnSearch(Rsp, Cancel)

Rsp        (String) The value in the textbox if any  
Cancel      (Boolean) Set to True to NOT run the OnEnter event and  
                put the focus back on the prompt

## OnTimer

The OnTimer event occurs at a user-defined interval when it is enabled. This event is used to perform some action based upon defined time interval.

Applies to: Application

Syntax: OnTimer()

Example:

First the timer interval is set and the timer is enabled.

```
Public Sub Form_Load()
    On Error Resume Next
    App.TimerInterval = 1000
    App.TimerEnabled = True
End Sub
```

```
'OnTimer event is given some logic.
Public Sub Form_OnTimer()
    On Error Resume Next
    'your logic
End Sub
```

## OnUpdate

The OnUpdate event occurs after the last prompt has been entered and just prior to RFgen updating the database. This event is typically used to validate that data was entered and process additional updates.

Applies to: Application

Syntax: OnUpdate(Cancel)

Cancel      (Boolean) is used to cancel the update.

## OnVocollect

This event is only used with Vocollect data collection hardware. When the Vocollect device activates a Vocollect task on an application, a recordset is sent to RFgen for processing. If RFgen needs to

communicate back to the user, RFgen passes to the Vocollect device another recordset.

Applies to: Application

Syntax: OnVocollect(rsIn, rsOut)

rsIn a DataRecord object that captures the collected data coming in from the Vocollect device  
rsOut a DataRecord object built in the RFgen VBA code that is sent to the Vocollect device

Example:

```
Public Sub prTaskODRUpdateStatus_OnVocollect (ByVal
rsIn As DataRecord, ByVal rsOut As DataRecord)
    On Error Resume Next
    '
    rsIn.Param("DateTime")
    rsIn.Param("SerialNumber")
    rsIn.Param("Operator")
    rsIn.Param("AssignmentId")
    rsIn.Param("LocationId")
    rsIn.Param("UpdateMode")
    rsIn.Param("UpdateStatus")
    '
    rsOut.Param("ErrCode") = 0
    rsOut.Param("Message") = ""
End Sub
```

## Terminate

The Terminate event occurs when an RFgen client is terminating. It will occur only once, and is typically used to close manually opened database connections or release links to ActiveX objects.

Applies to: RFgen (RFgen.bas)

Syntax: Terminate()

## Unload

The Unload event occurs when the application is terminating. It will occur only once, and is typically used to release any system resources manually opened that are specific to this application.

Applies to: Application

Syntax: Unload()

# VBA Language Extensions

VBA Extensions are additions to the standard Visual Basic for Applications scripting language. Using these extensions, the developer may easily control the client device data display, manipulate database records, communicate with other databases, control stored procedures, communicate with attached or Windows-based printers and much more. The extensions provided have been grouped into several categories and are as follows:

## Prompt Specific Extensions

These properties and methods are available at run time by using the name of the prompt or the RFPrompt wrapper function with specifying either the prompt name or prompt number. Be careful using the prompt number since adding or deleting prompts over time will create bugs in the script. Prompt numbers are best used when looping through the prompts to set a property like Visible = False. At design time, these properties can be found on the Fields Properties tab.

### Align

This property aligns the text of a label or the text in the field of a prompt either left, center, right or vertically. The vertical option only applies to labels.

Syntax: PromptID.Align = enValue

enValue (enAlign) an enumeration that contains AlignCenter, AlignLeft, AlignRight and AlignVertical.

Examples:

```
txtBox.Align = AlignCenter  
RFPrompt("txtBox").Align = AlignCenter  
RFPrompt(2).Align = AlignCenter
```

### Autosize

This property is only for the Label, Image and MenuList controls. The options are to size the control based on content, Fill the screen from its starting location on down, Horizontal size means with width of the screen, None for no auto sizing and Vertical which stretches on the vertical height of the control.

Syntax: PromptID.Autosize = enValue  
enValue (enSizeModes) an enumeration that contains  
AutosizeContent, AutosizeFill, AutosizeHorizontal,  
AutosizeNone and AutosizeVertical

Examples:

```
Image1.Autosize = AutosizeContent  
RFPrompt("Image1").Autosize = AutosizeContent  
RFPrompt(2).Autosize = AutosizeContent
```

## BackColor1

This property accesses the prompt's data field background color and is the primary background color. BackColor2 is only used to create gradients.

Syntax: PromptID.BackColor1 = vValue

Alternate: IValue = PromptID.BackColor1

nValue (Long) is the color.

vValue (Variant) sets the color.

Examples:

```
txtPart.BackColor1 = RGB(255,255,0) 'yellow  
txtPart.BackColor1 = &HFF0000 'blue  
txtPart.BackColor1 = QBColor(5) 'magenta  
RFPrompt("txtPart").BackColor1 = vbWhite  
RFPrompt(1).BackColor1 = vbWhite
```

## BackColor2

This property accesses the prompt's data field secondary background color. It is used to produce gradients from one color to another.

Syntax: PromptID.BackColor2 = vValue

Alternate: IValue = PromptID.BackColor2

nValue (Long) is the color.

vValue (Variant) sets the color.

Examples:

```
txtPart.BackColor1 = vbWhite  
txtPart.BackColor2 = vbRed  
txtPart.BackGradient = GradientDiagonalRight  
  
RFPrompt("txtPart").BackColor1 = vbWhite  
RFPrompt("txtPart").BackColor2 = vbRed  
RFPrompt("txtPart").BackGradient =  
GradientDiagonalRight
```

```
RFPrompt(2).BackColor1 = vbWhite  
RFPrompt(2).BackColor2 = vbRed  
RFPrompt(2).BackGradient = GradientDiagonalRight
```

## BackGradient

This property uses either just the BackColor(1) for solid backgrounds or both BackColor properties to create gradients in one of several directions in graphical mode.

Syntax: PromptID.BackGradient = enValue

enValue (enGradients) enumeration that contains GradientDefault, GradientDiagonalLeft, GradientDiagonalRight, GradientHorizontal, GradientNone, GradientVertical and GradientVerticalSplit

Examples:

```
txtPart.BackColor1 = RGB(0,0,255)  
txtPart.BackColor2 = vbWhite  
txtPart.BackGradient = GradientVertical  
RFPrompt("txtPart").BackGradient = GradientVertical  
RFPrompt(2).BackGradient = GradientVertical
```

## BorderStyle

This property affects the border associated with a specific prompt. The different options are Active Border where the border is given an outline color when it has the focus, Default which takes its value from the current theme, No Border so only the data field is visible, Standard is just the visible border without a color outline, Transparent hides the field allowing the text to be entered on the application's background and Visible On Focus where the field is transparent unless it has focus.

Syntax: PromptID.BorderStyle = enValue

Alternate: enValue = PromptID.BorderStyle

enValue (enBorderStyles) is the border style used to render the control object. The options are DisplayActiveBorder, DisplayDefault, DisplayNoBorder, DisplayStandard, DisplayTransparent, DisplayVisibleOnFocus

Examples:

```
txtPart.BorderStyle = DisplayVisibleOnFocus  
RFPrompt("txtPart").BorderStyle = DisplayNoBorder  
RFPrompt(1).BorderStyle = DisplayTransparent
```

## Caption

This property accesses the caption associated with a specific prompt.

Syntax: PromptID.Caption = vValue

Alternate: sValue = PromptID.Caption

sValue (String) is the caption of the prompt.

vValue (Variant) is the caption to display for the prompt.

Examples:

```
txtPart.Caption = "Part Number"
```

```
RFPrompt("txtPart").Caption = "Part Number"
```

```
RFPrompt(1).Caption = "Part Number"
```

## Checked

This property accesses the status of a checkbox object associated with a specific prompt.

Syntax: PromptID.Checked = bValue

Alternate: bValue = PromptID.Checked

bValue (Boolean) is the checked state of the prompt.

Examples:

```
chkPrint.Checked = True
```

```
bValue = chkPrint.Checked
```

```
RFPrompt("chkPrint").Checked = False
```

```
RFPrompt(2).Checked = False
```

## Defaults

This property accesses the default property associated with a specific prompt.

Syntax: PromptID.Defaults = vValue

Alternate: sValue = PromptID.Defaults

sValue (String) is the default expression for the prompt.

vValue (Variant) sets the default expression for the prompt.

Examples:

```
txtYesNo.Defaults = "N"
```

```
RFPrompt("txtYesNo").Defaults = "N;O" ' with override
```

```
RFPrompt(2).Defaults = "N;O" ' with override
```

The letter 'N' will be the default.

## DisplayOnly

This property accesses the display only property associated with a specific prompt. Setting it to True makes a prompt into a display only

field, while setting it to False allows users to access and change data at the prompt.

Syntax: PromptID.DisplayOnly = bValue

Alternate: bValue = PromptID.DisplayOnly

bValue (Boolean) is the display only state for the prompt.

Examples:

```
txtPart.DisplayOnly = True
```

```
RFPrompt("txtPart").DisplayOnly = False
```

```
RFPrompt(2).DisplayOnly = False
```

## Edits

This property accesses the Edits property associated with a specific prompt.

Syntax: PromptID.Editable = vValue

Alternate: sValue = PromptID.Editable

sValue (String) is the edit conditions for the prompt.

vValue (Variant) sets the edit conditions for the prompt.

Examples:

```
txtPart.Editable = "2N" ' only accept 2 numbers
```

```
RFPrompt("txtPart").Editable = "2N"
```

```
RFPrompt(2).Editable = "2N"
```

## ErrMsg

This property accesses the error message property associated with a specific prompt.

Syntax: PromptID.ErrMsg = vValue

Alternate: sValue = PromptID.ErrMsg

sValue (String) is the error message to display for the prompt.

vValue (Variant) sets the error message to display for the prompt.

Examples:

```
txtPart.Editable = "2N" ' only accept 2 numbers
```

```
txtPart.ErrMsg = "Must be 2 numbers."
```

```
RFPrompt("txtPart").ErrMsg = "Must be 2 numbers."
```

```
RFPrompt(2).ErrMsg = "Must be 2 numbers."
```

## FieldId

This property returns the field ID of a specific prompt. If the prompt is linked to a database field, that field name is returned.

Syntax: sValue = PromptID.FieldId  
sValue (String) is the field id/name for the prompt.

Examples:

```
Dim sValue As String  
sValue = txtPart.FieldId  
sValue = RFPrompt("txtPart").FieldId  
sValue = RFPrompt(2).FieldId
```

## Font.Bold

This property accesses the prompt's data field bold option.

Syntax: PromptID.Font.Bold = bValue  
Alternate: bValue = PromptID.Font.Bold  
bValue (Boolean) is True or False for bold or not bold.

Examples:

```
txtPart.Font.Bold = True  
RFPrompt("txtPart").Font.Bold = True  
RFPrompt(2).Font.Bold = True
```

## Font.Italic

This property accesses the prompt's data field italic option.

Syntax: PromptID.Font.Italic = bValue  
Alternate: bValue = PromptID.Font.Italic  
bValue (Boolean) is True or False for italics or no italics.

Examples:

```
txtPart.Font.Italic = True  
RFPrompt("txtPart").Font.Italic = True  
RFPrompt(2).Font.Italic = True
```

## Font.Size

This property accesses the prompt's data field text size.

Syntax: PromptID.Font.Size = nValue  
Alternate: vValue = PromptID.Font.Size  
vValue (Variant) is the font size.  
nValue (Long) sets the font size. Default is set in the theme.

Examples:

```
txtPart.Font.Size = 16  
RFPrompt("txtPart").Font.Size = 16
```

```
RFPrompt(2).Font.Size = 16
```

## Font.Underline

This property accesses the prompt's data field underline option.

Syntax: PromptID.Font.Underline = bValue

Alternate: bValue = PromptID.Font.Underline

bValue (Boolean) True or False for underline or no underline.

Examples:

```
txtPart.Font.Underline = True
```

```
RFPrompt("txtPart").Font.Underline = True
```

```
RFPrompt(2).Font.Underline = True
```

## ForeColor

This property accesses the prompt's data field fore color.

Syntax: PromptID.ForeColor = nValue

Alternate: vValue = PromptID.ForeColor

vValue (Variant) is the color.

nValue (Long) sets the color.

Examples:

```
txtPart.ForeColor = RGB(255,255,0) 'yellow
```

```
txtPart.ForeColor = &HFF0000 'blue
```

```
txtPart.ForeColor = QBColor(5) 'magenta
```

```
RFPrompt("txtPart").ForeColor = vbWhite
```

```
RFPrompt(2).ForeColor = vbWhite
```

## Format

This property accesses the format of a specific prompt. It is only an extension of the Format VBA command.

Syntax: PromptID.Format = vValue

Alternate: sValue = PromptID.Format

sValue (String) is the format mask to use when displaying data for the prompt.

vValue (Variant) sets the format mask to use when displaying data for the prompt.

Examples:

```
txtTime.Format = "hh:mm"
```

```
RFPrompt("txtTime").Format = "hh:mm"
```

```
c - General Date
```

```
dddddd - Long Date
dddd - Short Date
tttt - Long Time
hh:mm AMPM - Medium Time
hh:mm - Short Time
$#,##0.00 or ($#,##0.00) - Currency 0.00 - Fixed
#,##0.00 - Standard 0.00% - Percent 0.00E+00 -
Scientific
Yes/No - Return "No" if zero, else return "Yes"
True/False - Return "True" if zero, else return
"False"
On/Off - Return "On" if zero, else return "Off"
```

*For further examples get help on the VB FORMAT command.*

## **Height**

This property accesses the Height property for a prompt object. This value can be changed at run time if there is a need to change prompt sizes on the screen. Depending on the client environment the default values may be in either pixels or characters. The command SYS.UsePixels can be used to switch between pixels or characters to make getting or setting this property more convenient.

Syntax: PromptID.Height = nValue

Alternate: vValue = PromptID.Height

vValue (Variant) is the display length for a prompt object.

nValue (Long) sets the display length for a prompt object.

Examples:

```
Dim vValue As Variant
vValue = txtPart.Height
txtPart.Height = 10
```

```
vValue = RFPrompt("txtPart").Height
RFPrompt("txtPart").Height = 10
```

```
vValue = RFPrompt(2).Height
RFPrompt(2).Height = 10
```

## **Image.Bitmap**

This property accesses the image in the signature capture object and allows the user to write it to a file. This property can also read from a file and place the BMP formatted image in an Inage control. Be sure to have pressed Enter on the signature box before trying to read the picture from it. After the OnEnter event finishes the prompt will have a value.

Syntax: PromptID.Image.Bitmap = bImage  
Alternate: bImage = PromptID.Image.Bitmap  
bImage (Variant) is the byte array containing the image.

Example *reading from a file*:

```
Dim bImage() As Byte
Open "C:\sig.bmp" For Binary Access Read As #1
    ReDim bImage(LOF(1))
    Get #1,,bImage
Close #1
Image1.Image.Bitmap = bImage
```

Example *writing to a file*:

```
Dim bImage() As Byte
bImage = sigSignature.Image.Bitmap
Open "C:\sig.bmp" For Binary Access Write As #1
    Put #1,,bImage
Close #1
```

## Image.DisplayMode

This function places an image into the control and has options to tile, stretch, not alter or position the image in a number of ways. The options are: BottomCenter, BottomLeft, BottomRight, CenterCenter, CenterLeft, CenterRight, TopCenter, TopLeft, TopRight, Default, Stretched, Disabled and Tile.

Syntax: PromptID.Image.DisplayMode(enVal)  
Alternate: enVal = PromptID.Image.DisplayMode  
enVal (enImageAlign) is the alignment style for the image

Examples:

```
btnPrint.Image.DisplayMode = ImageAlignCenterCenter
RFPrompt("btnPrint").Image.DisplayMode =
ImageAlignTopLeft
RFPrompt(2).Image.DisplayMode = ImageAlignBottomRight
```

## Image.GetBitmap

This function retrieves a BMP object stored within the database and displays it for an image prompt.

Syntax: [bValue =] PromptID.Image.GetBitmap(SQL)  
SQL (String) is the ODBC call to retrieve the OLE object stored in the database.

bValue (Boolean) Optional – returns True or False for the success of the command

#### Examples:

```
Dim sSQL As String  
sSQL = "Select Image from Inv where PartNo = '100'"  
imgPartImage.Image.GetBitmap(sSQL)  
  
RFPrompt ("imgPartImage") .Image.GetBitmap (sSQL)  
RFPrompt (2) .Image.GetBitmap (sSQL)
```

### Image.LoadResource

This function retrieves an image object stored as an Images resource within the master database and displays it for an image prompt.

Syntax: [bValue =] PromptID.Image.LoadResource(sName)

sName (String) is the Image ID used to retrieve the Images resource object stored in the master database.

bValue (Boolean) Optional – returns True or False for the success of the command

#### Examples:

```
imgPartImage.Image.LoadResource ("stop")  
RFPrompt ("imgPartImage") .Image.LoadResource ("stop")  
RFPrompt (2) .Image.LoadResource ("stop")
```

### Image.Path

This property retrieves a graphic file and displays it in an image prompt. When using thin client mode the supported image types are BMP, DIB, GIF and JPG. When running in mobile mode, only BMP is supported.

Syntax: PromptID.Image.Path = vValue

Alternate: sValue = PromptID.Image.Path

sValue (String) is the path to the image.

vValue (Variant) sets the path to the image.

#### Examples:

```
imgPartImage.Image.Path = "C:\House.GIF"  
RFPrompt ("imgPartImage") .Image.Path = "C:\House.GIF"  
RFPrompt (2) .Image.Path = "C:\House.GIF"
```

### Index

This property returns the prompt number and is read only. The language extension App.PromptNo returns the same value.

Syntax: vValue = PromptID.Index  
vValue (Variant) is the variable containing the prompt's number.

Examples:

```
Dim vValue As Variant
vValue = txtPart.Index
vValue = RFPrompt("txtPart").Index
vValue = RFPrompt(2).Index
```

## Label.Align

This property aligns the text of a left, center, right or vertically. The vertical option only applies to labels.

Syntax: PromptID.Label.Align = enValue  
enValue (enAlign) an enumeration that contains AlignCenter, AlignLeft, AlignRight and AlignVertical.

Examples:

```
txtBox.Label.Align = AlignCenter
RFPrompt("txtBox").Label.Align = AlignCenter
RFPrompt(2).Label.Align = AlignCenter
```

## Label.AutoSize

This property is only for the Label, Image and MenuList controls. The options are to size the control based on content, Fill the screen from its starting location on down, Horizontal size means with width of the screen, None for no auto sizing and Vertical which stretches on the vertical height of the control.

Syntax: PromptID.Label.AutoSize = enValue  
enValue (enSizeMode) an enumeration that contains AutosizeContent, AutosizeFill, AutosizeHorizontal, AutosizeNone and AutosizeVertical

Examples:

```
txtBox.Label.AutoSize = AutosizeContent
RFPrompt("txtBox").Label.AutoSize = AutosizeContent
RFPrompt(2).Label.AutoSize = AutosizeContent
```

## Label.BackColor1

This property accesses the prompt's label background color and is the primary background color. BackColor2 is only used to create gradients.

Syntax: PromptID.Label.BackColor1 = vValue

Alternate: nValue = PromptID.Label.BackColor1

vValue (Variant) sets the color.

nValue (Long) is the color.

Examples:

```
txtPart.Label.BackColor1 = vbWhite  
txtPart.Label.BackColor1 = RGB(255,255,0) 'yellow  
txtPart.Label.BackColor1 = &HF0000          'blue  
txtPart.Label.BackColor1 = QBColor(5)        'magenta  
RFPrompt("txtPart").Label.BackColor1 = vbWhite  
RFPrompt(2).Label.BackColor1 = vbWhite
```

## Label.BackColor2

This property accesses the prompt's label secondary background color. It is used to produce gradients from one color to another.

Syntax: PromptID.Label.BackColor2 = vValue

Alternate: nValue = PromptID.Label.BackColor2

vValue (Variant) sets the color.

nValue (Long) is the color.

Examples:

```
Part.Label.BackColor1 = vbWhite  
Part.Label.BackColor2 = vbRed  
Part.Label.BackGradient = GradientDiagonalRight
```

## Label.BackGradient

This property uses either just the BackColor(1) for solid backgrounds or both BackColor properties to create gradients in one of several directions in graphical mode.

Syntax: PromptID.Label.BackGradient = enValue

enValue (enGradients) enumeration that contains GradientDefault, GradientDiagonalLeft, GradientDiagonalRight, GradientHorizontal, GradientNone, GradientVertical and GradientVerticalSplit

Examples:

```
txtPart.Label.BackColor1 = RGB(0,0,255)  
txtPart.Label.BackColor2 = vbWhite  
txtPart.Label.BackGradient = GradientVertical  
RFPrompt("txtPart").Label.BackGradient = GradientNone  
RFPrompt(2).Label.BackGradient = GradientVertical
```

## Label.BorderStyle

This property affects the border associated with a specific label. The different options are Active Border where the border is given an outline color when it has the focus, Default which takes its value from the current theme, No Border so only the label is visible, Standard is just the visible border without a color outline, Transparent hides the field allowing the text to be entered on the application's background and Visible On Focus where the field is transparent unless it has focus.

Syntax: PromptID.Label.BorderStyle = enValue

Alternate: enValue = PromptID.Label.BorderStyle

enValue (enBorderStyles) is the border style used to render the label object. The options are DisplayActiveBorder, DisplayDefault, DisplayNoBorder, DisplayStandard, DisplayTransparent, DisplayVisibleOnFocus

Examples:

```
txtPart.Label.BorderStyle = DisplayVisibleOnFocus  
RFPrompt("txtPart").Label.BorderStyle =  
DisplayDefault  
RFPrompt(1).BorderStyle = DisplayTransparent
```

## **Label.Font.Bold**

This property accesses the prompt's label bold option.

Syntax: PromptID.Label.Font.Bold = bValue

Alternate: bValue = PromptID.Label.Font.Bold

bValue (Boolean) is True or False for bold or not bold.

Examples:

```
txtPart.Label.Font.Bold = True  
RFPrompt("txtPart").Label.Font.Bold = True  
RFPrompt(2).Label.Font.Bold = True
```

## **Label.Font.Italic**

This property accesses the prompt's label italic option.

Syntax: PromptID.Label.Font.Italic = bValue

Alternate: bValue = PromptID.Label.Font.Italic

bValue (Boolean) is True or False for italics or no italics.

Examples:

```
txtPart.Label.Font.Italic = True  
RFPrompt("txtPart").Label.Font.Italic = True  
RFPrompt(2).Label.Font.Italic = True
```

## **Label.Font.Size**

This property accesses the prompt's label text size.

Syntax: PromptID.Label.Font.Size = nValue

Alternate: vValue = PromptID.Label.Font.Size

nValue (Long) sets the font size. Default is 10.

vValue (Variant) is the font size.

Examples:

```
txtPart.Label.Font.Size = 16  
RFPrompt("txtPart").Label.Font.Size = 16  
RFPrompt(2).Label.Font.Size = 16
```

## **Label.Font.Underline**

This property accesses the prompt's label underline option.

Syntax: PromptID.Label.Font.Underline = bValue

Alternate: bValue = PromptID.Label.Font.Underline

bValue (Boolean) True or False for underline or no underline.

Examples:

```
txtPart.Label.Font.Underline = True  
RFPrompt("txtPart").Label.Font.Underline = True  
RFPrompt(2).Label.Font.Underline = True
```

## **Label.ForeColor**

This property accesses the prompt's label fore (font) color.

Syntax: PromptID.Label.ForeColor = nValue

Alternate: vValue = PromptID.Label.ForeColor

nValue (Long) sets the color.

vValue (Variant) is the color.

Examples:

```
txtPart.Label.ForeColor = RGB(255,255,0) 'yellow  
txtPart.Label.ForeColor = QBColor(5) 'magenta  
txtPart.Label.ForeColor = &HFF0000 'blue  
txtPart.Label.ForeColor = vbWhite  
RFPrompt("txtPart").Label.ForeColor = vbWhite  
RFPrompt(2).Label.ForeColor = vbWhite
```

## **Label.Height**

This property accesses the Height property for a label object associated with a specific prompt. This value can be changed at run time if there is

a need to change prompt sizes on the screen. Depending on the client environment the default values may be in either pixels or characters. The command SYS.UsePixels can be used to switch between pixels or characters to make getting or setting this property more convenient.

Syntax: PromptID.Label.Height = nValue

Alternate: vValue = PromptID.Label.Height

nValue (Long) sets the display length for a prompt's label object.

vValue (Variant) is the display length for a prompt's label object.

Examples:

```
Dim vValue As Variant  
txtPart.Label.Height = 10  
RFPrompt("txtPart").Label.Height = 10  
vValue = RFPrompt(2).Label.Height
```

## Label.Left

This property accesses the column position for a label object associated with a specific prompt. This value can be changed at run time if there is a need to move prompts around on the screen. Depending on the client environment the default values may be in either pixels or characters. The command SYS.UsePixels can be used to switch between pixels or characters to make getting or setting this property more convenient.

Syntax: PromptID.Label.Left = nValue

Alternate: vValue = PromptID.Label.Left

nValue (Long) sets the starting column for a prompt's label object.

vValue (Variant) is the starting column for a prompt's label object.

Examples:

```
Dim vValue As Variant  
vValue = txtPart.Label.Left  
vValue = RFPrompt("txtPart").Label.Left  
RFPrompt(2).Label.Left = 1
```

## Label.Theme

This property gets or sets the theme for the label portion of the prompt.

Syntax: PromptID.Label.Theme = vValue

Alternate: vValue = PromptID.Label.Theme

vValue (Variant) is the name of the theme to set or retrieve

Examples:

```
Dim vValue As Variant  
txtPart.Label.Theme = "Standard"
```

```
vValue = txtPart.Label.Theme  
vValue = RFPrompt("txtPart").Label.Theme  
vValue = RFPrompt(2).Label.Theme
```

## Label.Top

This property accesses the row position for a label object associated with a specific prompt. This value can be changed at run time if there is a need to move prompts around on the screen. Depending on the client environment the default values may be in either pixels or characters. The command SYS.UsePixels can be used to switch between pixels or characters to make getting or setting this property more convenient.

Syntax: PromptID.Label.Top = nValue

Alternate: vValue = PromptID.Label.Top

nValue (Long) sets the row to display a prompt's label object.

vValue (Variant) is the row to display a prompt's label object.

Examples:

```
Dim vValue As Variant  
vValue = txtPart.Label.Top  
vValue = RFPrompt("txtPart").Label.Top  
RFPrompt(2).Label.Top = 3
```

## Label.Width

This property accesses the Width property for a label object associated with a specific prompt. This value can be changed at run time if there is a need to change prompt sizes on the screen. Depending on the client environment the default values may be in either pixels or characters. The command SYS.UsePixels can be used to switch between pixels or characters to make getting or setting this property more convenient.

Syntax: PromptID.Label.Width = nValue

Alternate: vValue = PromptID.Label.Width

nValue (Long) sets the display length for a prompt's label object.

vValue (Variant) is the display length for a prompt's label object.

Examples:

```
Dim vValue As Variant  
vValue = txtPart.Label.Width  
vValue = RFPrompt("txtPart").Label.Width  
RFPrompt(2).Label.Width = 10
```

## Left

This property accesses the column position for the data field portion of a prompt object. This value can be changed at run time if there is a need

to move prompts around on the screen. Depending on the client environment the default values may be in either pixels or characters. The command SYS.UsePixels can be used to switch between pixels or characters to make getting or setting this property more convenient.

Syntax: PromptID.Left = nValue

Alternate: vValue = PromptID.Left

nValue (Long) sets the starting column for a prompt's data field.

vValue (Variant) is the starting column for a prompt's data field.

Examples:

```
Dim vValue As Variant  
vValue = txtPart.Left  
vValue = RFPrompt("txtPart").Left  
RFPrompt(2).Left = 1
```

## List

This property contains the list collection. These properties and methods only apply to prompts that can contain a list such as ListBox, MenuList and Combobox.

### List.AddItem

This method is used to manually add items to a list control. You must specify the value to be returned should the user choose the selection. The Display Columns array is the data to be displayed across multiple columns if desired.

Syntax: [vIndex = ] PromptID.List.AddItem(vSelValue, vDispCols)

vIndex (Variant) Optional – the row index where the new entry was appended

vSelValue (Variant) the item to add to the list. If the user adds the pipe symbol to the selection value variable, it will cause additional columns to be created in the display.

vDispCols (Variant) the data to be displayed. This is in the format: "display1", "display2", "display3". The MenuList control will determine the widest value in each column and format the columns so they align properly regardless of font.

Example:

The following uses the AddItem method in the OnEnter event in one prompt to populate the list in another prompt.

```
Public Sub txtBox_OnEnter(Rsp As String, Cancel As Boolean, ErrMsg As String)
```

```
On Error Resume Next
lstCars.List.AddItem("2000", "2000 ABC Motor Co.")
lstCars.List.AddItem("2001", "2001 Widgets R Us")
lstCars.List.AddItem("2002", "2002 Goodfellow Inc.")
End Sub
```

## List.AddItemEx

This alternate method is used to manually add items to a MenuList control set to a graphical mode. You must specify the value to be returned should the user choose the selection. Image Resource name applies a graphic to the list entry. The Options parameter is used if the MenuList control is acting as a menu where the user can select forms to go to and each one requires additional passed information. See the Options column in the Menus / Roles List. The Display Columns array is the data to be displayed across multiple columns if desired.

Syntax: [vIndex = ] PromptID.List.AddItemEx(vValue, vImage, vOptions, vDispCols)

vIndex      (Variant) Optional – the row index where the new entry was appended  
vValue      (Variant) the item to add to the list  
vImage      (Variant) the image associated with the list entry  
vOptions    (Variant) the option for the menu item  
vDispCols   (Variant) the data to be displayed. This is in the format:  
              "display1",    "display2",    "display3".   The  
              MenuList control will determine the widest value in each  
              column and format the columns so they align properly  
              regardless of font.

Example:

The following uses the AddItemEx method in the OnEnter event in one prompt to populate the list in another prompt.

```
Public Sub txtBox_OnEnter(Rsp As String, Cancel As
Boolean, ErrMsg As String)
    On Error Resume Next
    lstCars.List.AddItemEx("1", "imgCheap", "", "VW
Bug", "Chevy Colt", "Dodge Dart")
    lstCars.List.AddItemEx("2", "imgMiddle", "", "Prius", "Camry", "Tacoma")
    lstCars.List.AddItemEx("3", "imgExpensive", "", "Corvette", "Tesla", "Lamborghini")
End Sub
```

## List.Cell

This method is used to read or change values or properties of a specific cell within a control that contains multiple rows and columns.

Syntax: PromptID.List.Cell(Row, Col).<method or property>

Row        (Long) specifies the row number in the grid

Col        (Long) specifies the column number in the grid

Examples:

```
RFPrompt("lstCars").List.Cell(2, 2).Value = "1969"
```

```
RFPrompt(2).List.Cell(2, 2).Bold = True
```

## List.Cell(x,y).BackColor1

This method is used to read or change the primary background color of a specific cell within a control that contains multiple rows and columns.

Syntax: PromptID.List.Cell(x,y).BackColor1 = vValue

Alternate: IValue = PromptID.List.Cell(x,y).BackColor1

IValue    (Long) is the color.

vValue    (Variant) sets the color.

Examples:

```
lstCars.List.Cell(2, 2).BackColor1 = RGB(255,255,0)
```

```
'yellow
```

```
lstCars.List.Cell(2, 2).BackColor1 = &HFF0000
```

```
'blue
```

```
lstCars.List.Cell(2, 2).BackColor1 = QBColor(5)
```

```
'magenta
```

```
RFPrompt("lstCars").List.Cell(2, 2).BackColor1 =
```

```
vbWhite
```

```
RFPrompt(1).List.Cell(2, 2).BackColor1 = vbWhite
```

## List.Cell(x,y).BackColor2

This method is used to read or change the secondary background color of a specific cell within a control that contains multiple rows and columns. It is used to produce gradients from one color to another.

Syntax: PromptID.List.Cell(x,y).BackColor2 = vValue

Alternate: IValue = PromptID.List.Cell(x,y).BackColor2

IValue    (Long) is the color.

vValue    (Variant) sets the color.

Examples:

```
lstCars.List.Cell(2, 2).BackGradient =
GradientVertical
lstCars.List.Cell(2, 2).BackColor2 = RGB(255,255,0)
'yellow
lstCars.List.Cell(2, 2).BackColor2 = &HFF0000
'blue
lstCars.List.Cell(2, 2).BackColor2 = QBColor(5)
'magenta
RFPrompt("lstCars").List.Cell(2, 2).BackColor2 =
vbWhite
RFPrompt(1).List.Cell(2, 2).BackColor2 = vbWhite
```

## List.Cell(x,y).BackGradient

This property uses either just the BackColor(1) for solid backgrounds or both BackColor properties to create gradients in one of several directions for a specific cell within a control that contains multiple rows and columns.

Syntax: PromptID.List.Cell(x,y).BackGradient = enValue  
enValue (enGradients) enumeration that contains GradientDefault, GradientDiagonalLeft, GradientDiagonalRight, GradientHorizontal, GradientNone, GradientVertical and GradientVerticalSplit

Examples:

```
lstCars.List.Cell(2, 2).BackColor1 = RGB(0,0,255)
lstCars.List.Cell(2, 2).BackColor2 = vbWhite
lstCars.List.Cell(2, 2).BackGradient =
GradientVertical
lstCars.List.Cell(2, 2).BackGradient =
GradientVertical
lstCars.List.Cell(2, 2).BackGradient =
GradientVertical
```

## List.Cell(x,y).Bold

This property accesses the prompt's data field bold option for a specific cell within a control that contains multiple rows and columns.

Syntax: PromptID.List.Cell(x,y).Bold = bValue  
Alternate: bValue = PromptID.List.Cell(x,y).Bold  
bValue (Boolean) is True or False for bold or not bold.

Examples:

```
lstCars.List.Cell(2, 2).Bold = True  
RFPrompt("lstCars").List.Cell(2, 2).Bold = True  
RFPrompt(2).List.Cell(2, 2).Bold = True
```

## List.Cell(x,y).ForeColor

This method is used to read or change the fore color of a specific cell within a control that contains multiple rows and columns.

Syntax: PromptID.List.Cell(x,y).ForeColor = vValue

Alternate: IValue = PromptID.List.Cell(x,y).ForeColor

IValue (Long) is the color.

vValue (Variant) sets the color.

Examples:

```
lstCars.List.Cell(2, 2).ForeColor = RGB(255,255,0)  
'yellow  
lstCars.List.Cell(2, 2).ForeColor = &HFF0000  
'blue  
lstCars.List.Cell(2, 2).ForeColor = QBColor(5)  
'magenta  
RFPrompt("lstCars").List.Cell(2, 2).ForeColor =  
vbWhite  
RFPrompt(1).List.Cell(2, 2).ForeColor = vbWhite
```

## List.Cell(x,y).Italic

This property accesses the prompt's data field italic option for a specific cell within a control that contains multiple rows and columns.

Syntax: PromptID.List.Cell(x,y).Italic = bValue

Alternate: bValue = PromptID.List.Cell(x,y).Italic

bValue (Boolean) is True or False for italic or not italic.

Examples:

```
lstCars.List.Cell(2, 2).Italic = True  
RFPrompt("lstCars").List.Cell(2, 2).Italic = True  
RFPrompt(2).List.Cell(2, 2).Italic = True
```

## List.Cell(x,y).Underline

This property accesses the prompt's data field underline option for a specific cell within a control that contains multiple rows and columns.

Syntax: PromptID.List.Cell(x,y).Underline = bValue

Alternate: bValue = PromptID.List.Cell(x,y).Underline  
bValue (Boolean) is True or False for underline or not underline.

Examples:

```
lstCars.List.Cell(2, 2).Underline = True  
RFPrompt("lstCars").List.Cell(2, 2).Underline = True  
RFPrompt(2).List.Cell(2, 2).Underline = True
```

## List.Cell(x,y).Value

This property accesses the prompt's data field value for a specific cell within a control that contains multiple rows and columns.

Syntax: PromptID.List.Cell(x,y).Value = vValue

Alternate: vValue = PromptID.List.Cell(x,y).Value  
vValue (Variant) is the value within the cell.

Examples:

```
lstCars.List.Cell(2, 2).Value = "1"  
RFPrompt("lstCars").List.Cell(2, 2).Value = "1"  
vValue = RFPrompt(2).List.Cell(2, 2).Value
```

## List.Clear

This method is used to remove all items from a Listbox, Combo box or MenuList. There is an optional Boolean parameters to clear any columns manually specified.

Syntax: PromptID.List.Clear([bClearColumns])

bClearColumns (Boolean) Optional – set to True to purge any manual “SetColumn” assignments and “false” to keep them. The default is True.

Examples:

```
lstCars.List.Clear(True)  
RFPrompt("lstCars").List.Clear  
RFPrompt(2).List.Clear
```

## List.Column

This method is used to read or change properties of a specific column within a control that contains columns.

Syntax: PromptID.List.Column(Col).<method or property>  
Col (Long) specifies the column number in the grid

Examples:

```
lstCars.List.Column(2).Width = 10  
RFPrompt(2).List.Column(2).Width = 10
```

## List.Column(x).Align

This property aligns the text of the column within a control that contains columns. The options are either center, left, or right.

Syntax: PromptID.List.Column(x).Align = enValue

enValue (enColAlign) an enumeration that contains TextCenter, TextLeft, and TextRight.

Examples:

```
lstCars.List.Column(2).Align = TextCenter  
RFPrompt("lstCars").Align = TextCenter  
RFPrompt(2).Align = TextCenter
```

## List.Column(x).Autosize

This property will size the column based on the widest value in that column.

Syntax: PromptID.List.Column(x).Autosize = bValue

bValue (Boolean) set to True or False to autosize the width of the column.

Examples:

```
lstCars.List.Column(2).Autosize = True  
RFPrompt("lstCars").Autosize = True  
RFPrompt(2).Autosize = True
```

## List.Column(x).BackColor1

This method is used to read or change the primary background color of the whole column within a control that contains multiple rows and columns.

Syntax: PromptID.List.Column(x).BackColor1 = vValue

Alternate: IValue = PromptID.List.Column(x).BackColor1

IValue (Long) is the color.

vValue (Variant) sets the color.

Examples:

```
lstCars.List.Column(2).BackColor1 = RGB(255,255,0)
'yellow
lstCars.List.Column(2).BackColor1 = &HFF0000
'blue
lstCars.List.Column(2).BackColor1 = QBColor(5)
'magenta
RFPrompt("lstCars").List.Column(2).BackColor1 =
vbWhite
RFPrompt(1).List.Column(2).BackColor1 = vbWhite
```

## List.Column(x).BackColor2

This method is used to read or change the secondary background color of the whole column within a control that contains multiple rows and columns. It is used to produce gradients from one color to another.

Syntax: PromptID.List.Column(x).BackColor2 = vValue

Alternate: IValue = PromptID.List.Column(x).BackColor2

IValue (Long) is the color.

vValue (Variant) sets the color.

Examples:

```
lstCars.List.Column(2).BackGradient =
GradientVertical
lstCars.List.Column(2).BackColor2 = RGB(255,255,0)
'yellow
lstCars.List.Column(2).BackColor2 = &HFF0000
'blue
lstCars.List.Column(2).BackColor2 = QBColor(5)
'magenta
RFPrompt("lstCars").List.Column(2).BackColor2 =
vbWhite
RFPrompt(1).List.Column(2).BackColor2 = vbWhite
```

## List.Column(x).BackGradient

This property uses either just the BackColor(1) for solid backgrounds or both BackColor properties to create gradients in one of several directions for the whole column within a control that contains multiple columns.

Syntax: PromptID.List.Column(x).BackGradient = enValue

enValue (enGradients) enumeration that contains GradientDefault,  
GradientDiagonalLeft, GradientDiagonalRight,

GradientHorizontal, GradientNone, GradientVertical and GradientVerticalSplit

Examples:

```
lstCars.List.Column(2).BackColor1 = RGB(0,0,255)
lstCars.List.Column(2).BackColor2 = vbWhite
lstCars.List.Column(2).BackGradient =
GradientVertical
RFPrompt("lstCars").List.Column(2).BackGradient =
GradientVertical
RFPrompt(1).List.Column(2).BackGradient =
GradientVertical
```

### List.Column(x).Bold

This property accesses the column's bold option for the whole column within a control that contains columns.

Syntax: PromptID.List.Column(x).Bold = bValue

Alternate: bValue = PromptID.List.Column(x).Bold

bValue (Boolean) is True or False for bold or not bold.

Examples:

```
lstCars.List.Column(2).Bold = True
RFPrompt("lstCars").List.Column(2).Bold = True
RFPrompt(2).List.Column(2).Bold = True
```

### List.Column(x).Caption

This property accesses the caption associated with the specified column.

Syntax: PromptID.List.Column(x).Caption = vValue

Alternate: sValue = PromptID.List.Column(x).Caption

sValue (String) is the title of the column.

vValue (Variant) sets the title of the column.

Examples:

```
lstCars.List.Column(2).Caption = "Model"
RFPrompt("lstCars").List.Column(2).Caption = "Model"
RFPrompt(1).List.Column(2).Caption = "Model"
```

### List.Column(x).DisplayOnly

This property accesses the display only property associated with the specified column. Setting it to True makes all the cells in that column unchangeable.

Syntax: PromptID.List.Column(x).DisplayOnly = bValue  
Alternate: bValue = PromptID.List.Column(x).DisplayOnly  
bValue (Boolean) is the display only state for the prompt.

Examples:

```
lstCars.List.Column(2).DisplayOnly = True
RFPrompt("txtPart").List.Column(2).DisplayOnly =
False
RFPrompt(2).List.Column(2).DisplayOnly = False
```

## List.Column(x).ForeColor

This property accesses the column's fore color property associated with the specified column.

Syntax: PromptID.List.Column(x).ForeColor = IValue  
Alternate: vValue = PromptID.List.Column(x).ForeColor  
vValue (Variant) is the color.  
IValue (Long) sets the color.

Examples:

```
lstCars.List.Column(2).ForeColor = RGB(255,255,0)
'yellow
lstCars.List.Column(2).ForeColor = &HFF0000
'blue
lstCars.List.Column(2).ForeColor = QBColor(5)
'magenta
RFPrompt("lstCars").List.Column(2).ForeColor =
vbWhite
RFPrompt(2).List.Column(2).ForeColor = vbWhite
```

## List.Column(x).Format

This property affects the format of the whole specified column. It is only an extension of the Format VBA command.

Syntax: PromptID.List.Column(x).Format = sValue  
Alternate: sValue = PromptID.List.Column(x).Format  
sValue (String) is the format mask to use when displaying data for the prompt.

**Examples:**

```
lstCars.List.Column(2).Format = "hh:mm"
RFPrompt("lstCars").List.Column(2).Format = "hh:mm"

c - General Date
dddddd - Long Date
ddddd - Short Date
ttttt - Long Time
hh:mm AMPM - Medium Time
hh:mm - Short Time
$,##0.00 or ($#,##0.00) - Currency 0.00 - Fixed
#,##0.00 - Standard 0.00% - Percent 0.00E+00 -
Scientific
Yes/No - Return "No" if zero, else return "Yes"
True/False - Return "True" if zero, else return
"False"
On/Off - Return "On" if zero, else return "Off"
```

*For further examples get help on the VB FORMAT command.*

### **List.Column(x).ImageMode**

This property formats images placed in the specified column to one mode option. The options are: BottomCenter, BottomLeft, BottomRight, CenterCenter, CenterLeft, CenterRight, ToprCenter, TopLeft, TopRight, Default, Stretched, Disabled and Tile.

Syntax: PromptID.List.Column(x).ImageMode(enVal)

Alternate: enVal = PromptID.List.Column(x).ImageMode

enVal (enImageAlign) is the alignment style for the image

**Examples:**

```
lstCars.List.Column(1).ImageMode =
ImageAlignCenterCenter
RFPrompt("lstCars").List.Column(1).ImageMode =
ImageAlignTopLeft
RFPrompt(2).List.Column(1).ImageMode =
ImageAlignBottomRight
```

### **List.Column(x).Italic**

This property accesses the column's italic option for the whole column within a control that contains columns.

Syntax: PromptID.List.Column(x).Italic = bValue

Alternate: bValue = PromptID.List.Column(x).Italic  
bValue (Boolean) is True or False for italic or not italic.

Examples:

```
lstCars.List.Column(2).Italic = True  
RFPrompt("lstCars").List.Column(2).Italic = True  
RFPrompt(2).List.Column(2).Italic = True
```

## List.Column(x).ScaleDecimals

This property formats the numeric values in the specified column if the database does not store decimals. This option will position a decimal at a position from the right side. A comma will be used for large numbers. This field is locale specific and will use the appropriate characters for each region.

Syntax: PromptID.List.Column(x).ScaleDecimals = vValue  
Alternate: vValue = PromptID.List.Column(x).ScaleDecimals  
vValue (Variant) is the decimal position.

Examples:

```
lstCars.List.Column(1).ScaleDecimals = 2  
RFPrompt("lstCars").List.Column(1).ScaleDecimals = 2  
RFPrompt(8).List.Column(1).ScaleDecimals = 2
```

## List.Column(x).TrimSpaces

This property formats the values in the specified column by deleting the leading and trailing spaces in the data.

Syntax: PromptID.List.Column(x).TrimSpaces = bValue  
Alternate: bValue = PromptID.List.Column(x).TrimSpaces  
bValue (Boolean) set to True to trim all space from the data.

Examples:

```
lstCars.List.Column(1).TrimSpaces = True  
RFPrompt("lstCars").List.Column(1).TrimSpaces = True  
RFPrompt(8).List.Column(1).TrimSpaces = True
```

## List.Column(x).Underline

This property accesses the column's underline option for the whole column within a control that contains columns.

Syntax: PromptID.List.Column(x).Underline = bValue

Alternate: bValue = PromptID.List.Column(x).Underline  
bValue (Boolean) is True or False for underline or not underline.

Examples:

```
lstCars.List.Column(2).Underline = True  
RFPrompt("lstCars").List.Column(2).Underline = True  
RFPrompt(2).List.Column(2).Underline = True
```

## List.Column(x).Visible

This property makes visible or invisible the whole column within a control that contains columns.

Syntax: PromptID.List.Column(x).Visible = bValue  
Alternate: bValue = PromptID.List.Column(x).Visible  
bValue (Boolean) is True or False for column visibility.

Examples:

```
lstCars.List.Column(2).Visible = True  
RFPrompt("lstCars").List.Column(2).Visible = True  
RFPrompt(2).List.Column(2).Visible = True
```

## List.Column(x).Width

This property sets or returns the width of the column within a control that contains columns.

Syntax: PromptID.List.Column(x).Width = vValue  
Alternate: vValue = PromptID.List.Column(x).Visible  
vValue (Variant) sets or gets the width of the specified column.

Examples:

```
lstCars.List.Column(2).Width = 25  
RFPrompt("lstCars").List.Column(2).Width = 25  
RFPrompt(2).List.Column(2).Width = 25
```

## List.Columns

This property returns the number of columns in the control.

Syntax: vValue = PromptID.List.Columns  
vValue (Variant) gets the number of columns in the control.

Examples:

```
Dim iCnt As Integer
```

```
iCnt = lstCars.List.Columns  
iCnt = RFPrompt("lstCars").List.Columns  
iCnt = RFPrompt(2).List.Columns
```

## List.Count

This function returns the number of items in the list collection.

Syntax: vValue = PromptID.List.Count

vValue (Variant) the number of items in the list

Example:

```
Dim nValue as Long  
nValue = lstCars.List.Count  
nValue = RFPrompt("lstCars").List.Count  
nValue = RFPrompt(2).List.Count
```

## List.Data

The list collection can be generated via the DB.MakeList, App.MakeList or ERP.MakeList functions and then assigned to the Listbox, Combobox or MenuList via this property.

Syntax: PromptID.List = vMyList

Alternate: sMyList = PromptID.List

vMyList (Variant) the list generated with the MakeList functions.

sMyList (String) the list contained in the prompt

Example:

The following uses the DB.MakeList function in the ListBox\_GotFocus event to populate the List property of the list box.

```
Dim sMyList As String  
Dim sSQL As String  
sSQL = "select PartNo from Inventory"  
sMyList = DB.MakeList(sSQL)  
lstParts.List.Data = sMyList  
or  
lstParts.List.Data = DB.MakeList(sSQL)
```

## List.Index

This property returns or sets the current list index property.

Syntax: PromptID.List.Index = nValue

Alternate: vValue = PromptID.List.Index

nValue (Long) sets the item index in the list

vValue (Variant) gets the item index in the list

Examples:

```
Dim nValue As Long  
nValue = Listbox1.List.Index  
nValue = RFPrompt("Listbox1").List.Index  
RFPrompt(2).List.Index = 5
```

## List.InsertItem

This method is used to manually insert items to a list control rather than simply adding new items to the list. You must specify the index in the list as the insertion point, the value to be returned should the user choose the selection and the Display Columns array of the data to be displayed, across multiple columns if desired. See the List.AddItem also.

Syntax: PromptID.List.InsertItem(nIndex, vValue, vDispCols)  
nIndex (Long) the insertion point in the list  
vValue (Variant) the returned value of the item to add to the list  
vDispCols (Variant) the data to be displayed. This is in the format:  
"display1", "display2", "display3". The  
MenuList control will determine the widest value in each  
column and format the columns so they align properly  
regardless of font.

Example:

The following uses the AddItem and InsertItem method in the OnEnter event in one prompt to populate the list in another prompt.

```
Public Sub txtBox_OnEnter(Rsp As String, Cancel As  
Boolean, ErrMsg As String)  
On Error Resume Next  
lstCars.List.AddItem("2001", "2001 ABC Motor Co.")  
lstCars.List.AddItem("2002", "2002 Widgets R Us")  
lstCars.List.InsertItem(1, "2000", "2000 Goodfellow  
Inc.")  
End Sub
```

This places the "2000 Goodfellow Inc." entry first in the list.

## List.InsertItemEx

This alternate method is used to manually insert items to a MenuList control rather than simply adding new items to the list. You must specify the index in the list as the insertion point, the value to be returned should the user choose the selection, the Image Resource for the graphic, the Options parameter if required and the Display Columns array of the data to be displayed, across multiple columns if desired. See the List.AddItemEx also.

Syntax: PromptID.List.InsertItemEx(nIndex, vValue, vImage, vOptions, vDispCols)

nIndex (Long) the insertion point in the list  
vValue (Variant) the item to add to the list  
vImage (Variant) the image associated with the list entry  
vOptions (Variant) the option for the menu item  
vDispCols (Variant) the data to be displayed. This is in the format:  
"display1", "display2", "display3". The  
MenuList control will determine the widest value in each  
column and format the columns so they align properly  
regardless of font.

Example:

The following uses the AddItemEx and InsertItemEx method in the OnEnter event in one prompt to populate the list in another prompt.

```
Public Sub txtBox_OnEnter(Rsp As String, Cancel As Boolean, ErrMsg As String)
    On Error Resume Next
    lstCars.List.AddItemEx("2", "imgCheap", "", "VW
Bug", "Chevy Colt", "Dodge Dart")
    lstCars.List.AddItemEx("3", "imgMiddle", "", "Prius", "Camry", "Tacoma")
    lstCars.List.InsertItemEx(1, "1", "imgExpensive",
    "", "Corvette", "Tesla", "Lamborghini")
End Sub
```

This places the expensive list first in the list.

### **List.PageDown**

This method scrolls the contents of a list object prompt down 1 page. A page is defined as the number of visible rows.

Syntax: PromptID.List.PageDown

Example: See *List.PageUp*

### **List.PageUp**

This method scrolls the contents of a list object prompt up 1 page. A page is defined as the number of visible rows.

Syntax: PromptID.List.PageUp

Example:

This uses a Form\_OnFkey event to activate the Listbox Page methods.

```
Public Sub Form_OnFkey(Fkey As Long)
    On Error Resume Next
    If Fkey = 5 Then
        lstParts.List.PageUp
    ElseIf Fkey = 6 Then
        lstParts.List.PageDown
    End If
End Sub
```

## List.RemoveItem

This method is used to manually remove items from a list object prompt. If the third entry is removed all entries further down are shifted up 1 place.

Syntax: PromptID.List.RemoveItem (vLocation)  
vLocation (Variant) the location to remove the item from the list

Examples:

```
lstParts.List.RemoveItem(3) ' Removes the third item
RFPrompt("lstParts").List.RemoveItem(3)
RFPrompt(2).List.RemoveItem(3)
```

## List.ScrollDown

This method scrolls the contents of a list object prompt down 1 row.

Syntax: PromptID.List.ScrollDown

Example: See List.ScrollUp

## List.ScrollUp

This method scrolls the contents of a list object prompt up 1 row.

Syntax: PromptID.List.ScrollUp

Example:

This uses a Form\_OnFkey event to activate the List box Scroll methods.

```
Public Sub Form_OnFkey(Fkey As Long)
    On Error Resume Next
    If Fkey = 5 Then
        lstParts.List.ScrollUp
    ElseIf Fkey = 6 Then
        lstParts.List.ScrollDown
```

```
End If  
End Sub
```

## List.SetColumn

This method is used in conjunction with the MenuList control to format the displayed columns. First specify which column is to be formatted, what the caption of that column is, how wide to make the column and how to justify the content. In addition to the MenuList contents being formatted, the caption of the control can also be overwritten with the column names if the ListHeading property is set to True. Then the caption will be spaced exactly the same as the columns.

Syntax: PromptID. List.SetColumn(nCols, sHeading, vSize, enAlign)  
nCols (Long) the column number to be affected by the formatting  
sHeading (String) the title that will appear across the top of the column  
vSize (Variant) the width of the column in characters  
enAlign (enColAlign) how to align the column; either Left, Right or Center; the default is Left justified

Examples:

```
oMyList.SQL = "select * from PLANTS"  
oMyList.SetColumn 1, "ID", 5, TextLeft  
oMyList.SetColumn 2, "NAME", 21, TextLeft  
sName = oMyList.ShowList
```

## List.Sorted

This property indicates whether the items of the list box are to be sorted alphabetically. Numbers are sorted alphabetically, not numerically. Therefore 1, 10, 100, 2, 3, 4 are numbers sorted alphabetically.

Syntax: PromptID. List.Sorted = bValue  
Alternate: bValue = PromptID. List.Sorted  
bValue (Boolean) set to True to sort the items in ascending order

Examples:

```
Dim bValue As Boolean  
lstParts.List.Sorted = True  
RFPrompt("lstParts").List.Sorted = True  
bValue = RFPrompt(2).List.Sorted
```

## List.Value(x)

This property returns the value in the specified row for the control. A row can have a value if it was placed there via an AddItem or InsertItem method or the ListData property of the control.

Syntax: vValue = PromptID.List.Value(Row)  
Row (Long) is the row number in the list  
vValue (Variant) is given the value assigned to the control's row

Examples:

```
Dim vValue As Variant  
vValue = lstParts.List.Value(1)  
vValue = RFPrompt("lstParts").List.Value(1)  
vValue = RFPrompt(2).List.Value(1)
```

## PageNo

This property accesses the page number property associated with a specific prompt. This property is read-only at run time.

Syntax: vPage = PromptID.PageNo

vPage (Variant) is numeric and represents the page number on which the prompt will be displayed.

Examples:

```
Dim iValue as Integer  
iValue = txtPart.PageNo  
iValue = RFPrompt("txtPart").PageNo  
iValue = RFPrompt(2).PageNo
```

## Password

This property, when set to True or False, will turn on or off asterisks (\*) in the data field of a textbox that mask the data.

Syntax: PromptID.Password = bValue

bValue (Boolean) set to True or False

Examples: These reference the RFLogin form:

```
Password.Password = True  
RFPrompt("Password").Password = True  
RFPrompt(5).Password = True
```

## Required

This property, when set to 'True', forces users to enter data into the prompt, while setting it to False allows users to skip the field. If the prompt never gets the focus, this property will not have a chance to be used.

Syntax: PromptID.Required = bValue

Alternate: bValue = PromptID.Required

bValue (Boolean) is the required state for the prompt.

**Examples:**

```
Dim bValue As Boolean  
bValue = txtPart.Required  
bValue = RFPrompt("txtPart").Required  
RFPrompt("txtPart").Required = True
```

## ScrollBars

This property can be used either on the form control or a list control to enable or disable the scrollbars. The options are: Horizontal, Vertical, Both, None, or the default value from the selected theme.

Syntax: PromptID.ScrollBars = enValue

Alternate: Form.ScrollBars = enValue

Alternate: enValue = PromptID.ScrollBars

enValue (enScrollBars) is the option for enabling or disabling the scrollbars on the control.

**Examples:**

```
Form.ScrollBars = enScrollDefault  
txtPart.ScrollBars = enScrollBoth  
RFPrompt("txtPart").ScrollBars = enScrollVert  
RFPrompt("txtPart").ScrollBars = enScrollNone
```

## SelLength

In graphical mode, this property sets or returns the number of characters highlighted by the mouse for a specified text prompt. For example, if the text box contained '123456' and using the mouse you drag and select only '456', the property will return 3, the number of characters selected.

Syntax: vValue = PromptID.SelLength

Alternate: PromptID.SelLength = iValue

vValue (Variant) returns the numeric value

iValue (Integer) sets the numeric value and selects the specified number of characters in the field starting from the SelStart value

**Examples:**

```
Dim vValue as Variant  
vValue = txtPart.SelLength  
vValue = RFPrompt("txtPart").SelLength  
RFPrompt("txtPart").SelLength = 3
```

## SelStart

In graphical mode, this property sets or returns the position where the highlighting starts for a specified text prompt. For example, if the text box contained '123456' and using the mouse you drag and select only '56', the property will return 4, the position where the selection started.

Syntax: vValue = PromptID.SelStart

Alternate: PromptID.SelStart = iValue

Nbr (Variant) is numeric, or fieldname, and refers to a specific prompt.

vValue (Variant) returns the numeric value

iValue (Integer) sets the numeric value

Examples:

```
Dim vValue As Variant
```

```
vValue = txtPart.SelStart
```

```
vValue = RFPrompt("txtPart").SelStart
```

```
RFPrompt("txtPart").SelStart = 1
```

## Text

This property accesses the data contained by a text box object associated with a specific prompt. Other prompts also use this property such as the list objects. The property will return the highlighted entry in the list.

Syntax: PromptID.Text = vValue

Alternate: Value = PromptID.Text

vValue (Variant) is the data collected in a prompt object.

Examples:

```
Dim vValue As Variant
```

```
vValue = txtPart.Text
```

```
vValue = RFPrompt("txtPart").Text
```

```
RFPrompt(2).Text = "Car Parts"
```

## Top

This property accesses the row position for a text box object associated with a specific prompt. This value can be changed at run time if there is a need to move prompts around on the screen. Depending on the client environment the default values may be in either pixels or characters. The command SYS.UsePixels can be used to switch between pixels or characters to make getting or setting this property more convenient.

Syntax: PromptID.Top = nValue

Alternate: vValue = PromptID.Top

nValue (Long) sets the row for a prompt's data field.

vValue (Variant) is the row for a prompt's data field.

Examples:

```
Dim iValue As Integer  
iValue = txtPart.Top  
iValue = RFPrompt("txtPart").Top  
RFPrompt("txtPart").Top = 5
```

## Type

This property will return a read-only enumeration describing what type of prompt is defined based on the prompt index name or number.

Syntax: enValue = PromptID.Type

enValue (enFieldType) the enumeration for the prompt type. Values are:

```
rfButton  
rfButtonList  
rfCheckBox  
rfComboBox  
rfDataGrid  
rfDesktopIcons  
rfForm  
rfFrame  
rfHostScreen  
rfImage  
rfImageList  
rfLabel  
rfListBox  
rfMenuList  
rfOptions  
rfSignature  
rfTextBox  
rfVocollectTask
```

Examples:

```
Dim enValue As enFieldType  
enValue = txtPart.Type
```

```
Dim i As Integer  
For i = 1 To App.PromptCount  
    If RFPrompt(i).Type = rfLabel Then Exit For  
Next
```

## ValField

This property accesses the validation field property associated with a specific prompt. When using ValTable and ValField properties, you must code them specifically with ValTable first and then ValField. When changing the validation field, you must make sure the validation table contains the new validation field. See ValTable.

Syntax: PromptID.ValField = vValue

Alternate: sValue = PromptID.ValField

vValue (Variant) sets the validation field to access whenever data is entered at the prompt.

sValue (String) is the validation field in the Table specified by ValTable.

Examples:

```
Dim vValue As Variant  
vValue = txtPart.ValField  
'  
RFPrompt("txtPart").ValTable = "ItemMaster"  
RFPrompt("txtPart").ValField = "PartNo"
```

## ValTable

This property accesses the validation table property associated with a specific prompt. When using ValTable and ValField properties, you must code them specifically with ValTable first and then ValField. When changing the validation table, you must make sure the validation field is contained in the new validation table. See ValField.

Syntax: PromptID.ValTable = vValue

Alternate: sValue = PromptID.ValTable

vValue (Variant) sets the validation table to access whenever data is entered at the prompt.

sValue (String) is the validation table to access whenever data is entered at the prompt.

Examples:

```
Dim vValue As Variant  
vValue = txtPart.ValTable  
'  
RFPrompt("txtPart").ValTable = "ItemMaster"  
RFPrompt("txtPart").ValField = "PartNo"
```

## Visible

This property accesses the visible property associated with a specific prompt. Setting it to True makes a prompt visible, while setting it to False makes it invisible. Even though the prompt may be invisible, the

GotFocus, OnEnter and LostFocus events will still be executed for this prompt. The screen must either be manually refreshed immediately or the event allowed to finish before the screen will be updated.

Syntax: PromptID.Visible = bValue

Alternate: bValue = PromptID.Visible

bValue (Boolean) is the visible state for the prompt.

Examples:

```
Dim bValue As Boolean  
bValue = txtPart.Visible  
bValue = RFPrompt("txtPart").Visible  
RFPrompt(2).Visible = False
```

## Width

This property accesses the Width property for a prompt object. This value can be changed at run time if there is a need to change prompt sizes on the screen. Depending on the client environment the default values may be in either pixels or characters. The command SYS.UsePixels can be used to switch between pixels or characters to make getting or setting this property more convenient.

Syntax: PromptID.Width = nValue

Alternate: vValue = PromptID.Width

nValue (Long) sets the display length for a prompt object.

vValue (Variant) is the display length for a prompt object.

Examples:

```
Dim vValue As Variant  
vValue = txtPart.Width  
vValue = RFPrompt("txtPart").Width  
RFPrompt(2).Width = 10
```

# Application-Based Extensions

Application-based language extensions are a group of commands that form the backbone of an application or transaction macro. We recommend looking in this group first for general commands that are not specific to mobile devices, screen mapping or other specific functions.

## Balloon

This command will create a popup message on the screen that can last indefinitely or for some number of seconds.

Syntax: App.Balloon(sText, nDuration)  
sText (String) the message to display.  
nDuration (Long) the number of seconds the message should display.  
Set to zero to clear the balloon or -1 for it to display indefinitely. If -1 is used, use a Balloon set to 0 duration to remove it.

Example:

```
App.Balloon("Transaction Processing...", 5)
```

## CallForm

This command will transfer the user to another application. There is an option to return from the called application to the previous one but if another CallForm command issued to go to a third or more applications first then as each application is exited the user will be brought back to each of the previous ones until finally coming back to the original application. There is no limit to the number of "sub" applications that can be called.

Syntax: App.CallForm(sName, [bReturn], [bSaveRSP], [bInitScreen])  
sName (String) is the name of the application to call.  
bReturn (Boolean) Optional – set to True to return to the current application after collecting data in the called application.  
The default is False.  
bSaveRSP (Boolean) Optional – set to True to retain the value in the current prompt's text field. The default is False.  
bInitScreen (Boolean) Optional – set to True to delete the screens memory object prior to going to the application. This is done in case the commands App.GetValue, App.SetValue, or App.ClearValues affected the data on this application prior to calling it. The default is False.

Example: This will call application "Cycle" without returning.

```
App.CallForm("Cycle", False)
```

## CallMacro

This function is used to call any transaction macro and pass in the required passing parameters. It returns True if the macro was successfully completed. These macros can be created from the Transactions Tree.

Syntax: enValue = App.CallMacro(sMacroName, bQueueOffline, [vParams])  
enValue (enMacroResult) – Values are:  
MacroFailed  
MacroNotProcessed

	MacroQueued
	MacroSucceeded
sMacroName	(String) – This is the name of the macro to be called.
bQueueOffline	(Boolean) – This determines whether the macro can be queued for later processing if the host is not currently available.
vParams	(Variant) – Optional: A series of passing parameters as required by the selected macro. Note: these parameters are the fields you defined when you wrote the macro.

Example:

```
Dim enValue As enMacroResult
enValue = App.CallMacro("SaveData", True, "Sam", "2")
```

## CallMenu

This command will transfer the user to another menu. It is typically used to log a user into the server when bypassing the Logon process. If the application making the menu call was deep within the menu structure, the server will assume that the called menu is the root menu. Going back up 1 level from a called menu will take you to the Login screen if one exists. (Note: the new menu will be called only after the Visual Basic Event has been exited.)

Syntax: App.CallMenu(sName)

sName (String) is the name of the menu to call as defined by the Menus tree.

Example:

```
App.CallMenu("Sample") ' Calls the "Sample" menu.
```

## ChangeLoginForm

This function will set the default login application for the duration of the client's session. The purpose is to allow several different login applications (possibly different languages) to exist at the same time and have the client's device specify which login application should be used. In the RFgen.bas module use the OnLocale event to get the client's location and then use App.ChangeLoginForm in that event to display the proper application for the user.

Syntax: App.ChangeLoginForm (sName)

sName (String) is the default login application name for the connected Client device.

Examples:

```
App.ChangeLoginForm("RFLoginSpanish")
App.ChangeLoginForm("RFLoginForkLift")
```

## ChangeUserPassword

This command will change the user's password to a new value if the old password is correctly provided. This is a permanent change stored in the system.

Syntax: [bValue =] App.ChangeUserPassword(sOldPwd, sNewPwd)  
bValue      (Boolean) Optional – is True or False based on the success of the command.  
sOldPwd     (String) is the user's current password  
sNewPwd    (String) is the new password

Example:

```
App.ChangeUserPassword("Mike123", "Mike456")
```

## ClearValues

This command will erase the values of all fields contained in the specified application. It is used to reset the values for applications that must be called more than once.

Syntax: App.ClearValues([vName])  
vName (Variant) Optional – is the name of the application's prompt to erase.

Example:

```
App.ClearValues("Cycle") ' Clears the values on the application titled "Cycle".
```

## ClientType

This function will return the type of client connecting in to the server.

Syntax: sType = App.ClientType  
sType (String) "TE", "GUI", "MOBILE", "XML", "TMQUEUE", or "TMEVENT"

## ConnAvailable

This function will return a Boolean (True / False) response that indicates whether or not the specified data connection is currently working.

Syntax: bValue = App.ConnAvailable(vSource)  
bValue    (Boolean) equals True if the server is currently connected to the specified source; equals False if it is not available

vSource (Variant) is the string representation or the numeric representation of the data connection

Examples:

```
Dim bCheck As Boolean  
bCheck = App.ConnAvailable(1)  
bCheck = App.ConnAvailable("RFSample")
```

## ExecuteMenuItem

This command is used to select a menu item based on the index number or name of the item in the menu. This command could be used to automate menu selections.

Syntax: App.ExecuteMenuItem(vSelection)

vSelection (Variant) either the menu index or display text of the menu prompt's item to be selected and executed.

Examples:

```
App.ExecuteMenuItem("Inventory Transfer")  
App.ExecuteMenuItem(1)
```

## ExitForm

This command exits the current application and returns to the calling menu or application. The VBA event where this is called must exit before this statement is carried out. Therefore it is always best to use and Exit Sub statement immediately after this command.

Syntax: App.ExitForm

## ExitSession

This command exits the current device session completely, the same as entering 'Q' at the original login screen. The VBA event where this is called must exit before this statement is carried out. Therefore it is always best to use and Exit Sub statement immediately after this command.

Syntax: App.ExitSession

## GetInput

This function will retrieve data from the input device that is not associated with any prompt. By specifying a set of X,Y coordinates, the cursor will blink at that position and accept data entry until the Length property is reached. This is similar in concept to a standard InputBox control. See SYS.UsePixels to reference a screen location by pixel.

Syntax: vValue = App.GetInput(iColumn, iRow, [iLength])  
vValue (Variant) is the value that is entered/scanned by the user.  
iColumn (Integer) is the column at which to place the cursor.  
iRow (Integer) is the row at which to place the cursor.  
iLength (Integer) Optional – is a maximum data entry length.

Example:

```
Dim vValue As Variant  
vValue = App.GetInput(0,12,5)
```

## GetString

This function will return text from the Translation Resources that have been saved as part of the configuration. There are two ways this command can be used. Either enter the Text ID and optionally the default text value or just enter the Text Resource value in place of the Text ID with no optional default text parameter.

Syntax: sValue = App.GetString(TextID, [vDefaultText])  
sValue (String) is the text string extracted from the Translation resource based on the locale of the client.  
TextID (Variant) is the Text ID value acting as a key to the Translation resource. Alternatively it can be the Text Resource value (an alternate key) to the Translation resource.  
vDefaultText (Variant) Optional – is the text to be used if the Text ID cannot be located in the Translations resource.

Example:

```
Dim sText As String
```

If the keys for a Translation resource were:

Text Id: 25

Text Resource: Hello

and the translation grid contained:

Locale: English

String Value: Hello There!

```
sText = App.GetString(25)  
sText = Hello There!  
sText = App.GetString(25, "Hello")  
sText = Hello There!  
sText = App.GetString("Hello")  
sText = Hello There!  
sText = App.GetString(99, "Hello")
```

```
sText = Hello - Because the ID could not be found the default text was used  
sText = App.GetString(99)  
sText = <nothing> - Because the ID could not be found and there was no default text  
sText = App.GetString("Hi")  
sText = Hi - Because the ID could not be found but it was of string type so it was considered the default text
```

## GetValue

This function will get the current value of a field in the current recordset or the value of a specific prompt.

Syntax: vValue = App.GetValue(vField, [vFormName])  
vValue (Variant) is the value extracted. (Note: this will always be a string value regardless of the actual field's data type.)  
vField (Variant) to extract the value from a prompt, or is the name of a field in the current recordset.  
vFormName (Variant) Optional – is the name of the application's recordset to extract data from. This is typically used to extract data from a calling application's recordset.

Examples:

```
App.GetValue(2) ' Gets value from prompt #2  
App.GetValue("PartNo") ' Gets the value from "PartNo"  
App.GetValue(2,"Cycle") ' Gets the value from the second prompt on the "Cycle" application
```

## IpAddress

This function will return the IP Address of the current Client device.

Syntax: sValue = App.IpAddress()  
sValue (String) is the IP Address of the connected Client device.

Example:

```
Dim sAddress As String  
sAddress = App.IpAddress
```

## Locale

This function can set or return the number associated with the current locale of the connecting device. This command accepts a number (LCID) and sets the locale within the client session. Examples would be 1033 for United States and 1041 for Japan. A web search for 'locale codes' should provide a list of LCIDs.

Syntax: nValue = App.Locale  
Alternate: App.Locale = nValue  
nValue (Long) is the number associated with the current locale

Example:

```
Dim nLocale As Long
nLocale = App.Locale
App.Locale = 1041
```

## LogError

This extension writes an entry into the General Error Log file. The purpose for the percent signs is if you use this command in a global capacity and want to pass in parameters. Each percent sign is simply replaced with the next item in the Params list.

Syntax: App.LogError(sProcedure, sErrDesc, [sParams])  
sProcedure (String) is typically the application name  
sErrDesc (String) is the ErrDesc entry  
sParams (string array) Optional – these are parameters that will be substituted into the ErrDesc where a percent sign is used

Examples:

```
App.LogError("Application1", "Invalid Item")
Time...: 3/27/2006 1:37:55 PM
Process: Application1
ErrDesc: Invalid Item
```

```
App.LogError("Application1", "Invalid Item by user
%", App.User)
Time...: 3/27/2006 1:37:55 PM
Process: Application1
ErrDesc: Invalid Item by user Sam
```

```
App.LogError("Application1", "Invalid Item % % %",
"1", "2", "3")
Time...: 3/27/2006 1:37:55 PM
Process: Application1
ErrDesc: Invalid Item 1 2 3
```

## LogErrorEx

This extension is the same as LogError but lets the user change the category value of the log entry.

Syntax: App.LogError(sProcedure, sErrDesc, Category, [sParams])  
sProcedure (String) is typically the application name

- sErrDesc** (String) is the ErrDesc entry  
**Category** (enErrorCategory) is the list of message types such as Error, Information, System, Trace and Warning  
**sParams** (string array) Optional – these are parameters that will be substituted into the ErrDesc where a percent sign is used

Example:

```
App.LogErrorEx("App1", "Invalid Item", catInfo)
Time..: 3/27/2014 1:37:55 PM
Process: App1
ErrDesc: Invalid Item
Category: Information
```

## MakeList

This function builds a scrolling list of items that may then be presented to the user for selection using App.ShowList.

Syntax: App.MakeList(sListData, vValue, vDisplay, [vHeading])  
 sListData (String) is the variable containing the list being created.  
**vValue** (Variant) is the value that is returned when the user selects a specific list item.  
**vDisplay** (Variant) is what the user will see for a specific list item. (Non-numeric values must be in quotes.)  
**vHeading** (Variant) Optional – is the heading for the list. (Note: it only needs to be assigned once).

Example:

```
Dim sRsp As String
Dim sList As String
App.MakeList sList, 2000, "2000 ABC Company", "Select
order"
App.MakeList sList, 2001, "2001 Widgets R Us"
App.MakeList sList, 2002, "2002 GoodFellows Inc."
sRsp = App.ShowList(sList)
```

## MsgBox

This function clears the screen in character mode or displays a message box when in graphical mode to the user. The 'value', 'type', 'default', and 'buttons' variables match those of the Visual Basic MsgBox function; e.g.:

Syntax: vValue = App.MsgBox(sMessage, [enType], [iDefault], [vButtons], [vCaption])  
**vValue** (Variant) will contain an integer or string indicating which selection was made. A string is the result only when custom

	buttons are defined. vbOK =OK button vbCancel =Cancel button vbAbort=Abort button vbRetry =Retry button vbIgnore = Ignore button vbYes =Yes button vbNo =No button
sMessage	(String) the message text to be displayed on the device
enType	(enMsgBoxTypes) Optional – the expected responses: vbOkOnly = OK vbOkCancel = OK, Cancel vbAbortRetryIgnore = Abort, Retry, Ignore vbYesNoCancel = Yes, No, Cancel vbYesNo = Yes, No vbRetryCancel = Retry, Cancel vbCritical – displays the Critical icon vbCustom – displays the Custom icon vbExclamation – displays the Exclamation icon vbInformation – displays the Information icon vbQuestion – displays the Question icon
iDefault	(Integer) Optional – an optional message box default: 1 = First Button 2 = Second Button 3 = Third Button
vButtons	(Variant) Optional – captions for the custom buttons; "[A] [B]" will set "A" as the caption for the first button, "B" for the second.
vCaption	(Variant) Optional – the text that appears as the title of the message box

**Examples:**

```
Dim vValue As Variant
vValue = App MsgBox("Ok?", vbCritical+vbRetryCancel)
Combining the Type parameter to create the icon and the button type desired.
vValue = App MsgBox("Continue with transaction?", vbYesNo, 1)

Dim vValue as Variant
vValue = App MsgBox("Continue with transaction?", vbYesNo, 1, "[Good] [Bad]", "Program Stopped")

If App MsgBox("OK?", vbYesNo) = vbNo Then Exit Sub
```

*The result in Value will be a string containing the label of the button pressed only if custom buttons are used. Otherwise the button number is returned. Defining unique buttons returns the name of the selected button.*

## **PromptCount**

This function will return the number of prompts that exist in an application. It is typically used when looping through the prompts to set properties like Visible.

Syntax: vValue = App.PromptCount

vValue (Variant) the number of prompts in the current application

Example:

```
Dim iVal As Integer  
iVal = App.PromptCount
```

## **PromptNo**

This property indicates the prompt number of the prompt that has the focus (read only).

Syntax: Nbr = App.PromptNo

Nbr (Integer) the number of the current prompt

## **SendChar**

This function sends an ASCII character directly to the control on the screen as if it came from the keyboard. The server does not intercept the character and process it. It is similar to the VB 'SendKeys' command.

Syntax: App.SendKey(nKeyASCII)

nKeyASCII (Long) the ASCII value of the character

Example:

```
App.SendKey(43) ' this would send the * character
```

## **SendKey**

This function sends an ASCII character to the server as if it came from the keyboard. The server will process it if it has special meaning. For example, sending an F key will get processed in the Form\_OnFkey event if it is programmed to do so. It is similar to the VB 'SendKeys' command.

Syntax: App.SendKey(enKeyCode)

enKeyCode (enKeyCodeConstants)	
vbKey0 – vbKey9	vbKeyA - vbKeyZ
vbKeyF1 - vbKeyF16	vbKeyNumpad0 – vbKeyNumpad9
vbKeyAdd	vbKeyBack
vbKeyCancel	vbKeyCapital
vbKeyClear	vbKeyControl
vbKeyDecimal	vbKeyDelete
vbKeyDivide	vbKeyDown
vbKeyEnd	vbKeyEscape
vbKeyExecute	vbKeyHelp
vbKeyHome	vbKeyInsert
vbKeyLButton	vbKeyLeft
vbKeyMButton	vbKeyMenu
vbKeyNumlock	vbKeyPageDown
vbKeyPageUp	vbKeyPause
vbKeyPrint	vbKeyRButton
vbKeyReturn	vbKeyRight
vbKeyScrollLock	vbKeySelect
vbKeySeparator	vbKeyShift
vbKeySnapshot	vbKeySpace
vbKeySubtract	vbKeyTab
vbKeyUp	

Example:

```
App.SendKey(vbKeyReturn) ' this sends the enter key
```

## SetDisplay

This function sets an alternative application display, if one has been established. Each display may have a variation of the application such as an alternate language used to make prompt captions. The prompts may have their font sizes changed to be seen more easily on large displays from a distance. The underlying code references the prompts by their Field ID or number so the many displays of the application only need one code base. Based on the user profile or the IP address of the connecting device, the proper display can be automatically shown to the user.

Syntax: App.SetDisplay (sDisplay, [iWidth], [iHeight])

sDisplay (String) the name established for the alternative application display.

iWidth, iHeight (Integer) Optional – allows the user to get a display of an application based on the screen size of the application created if the Display name was not found.

For example, if you had these displays created:

- Default 20x16
- Spanish 240x320
- Forklift 40x8
- Forklift 800x600

Then the command App.SetDisplay("Forklift",800,600) would bring up the last display. For telnet mode the character widths and heights are used. For graphical devices, pixels are used.

## **SetFocus**

This command is used to reposition the focus to a specific prompt. App.SetFocus should be the last statement for an event since subsequent statements will be ignored.

If the optional SaveRSP parameter is used the data entered for the current prompt will be saved in the current record, prior to the App.SetFocus. The SaveRSP flag does not apply to the Rsp variable in the OnEnter event but will only save the text in the textbox itself. To alter the textbox value in code use the RFPrompt(1).Text property and give it a value.

The ValidateRSP flag will check the text value against the Edits property and also execute the OnEnter event. This makes the most sense when the SetFocus method is used in an event other than the OnEnter event. If the SetFocus method is used in the OnEnter event the ValidateRSP flag will not process the edits again or run through the OnEnter event a second time. Be sure not to cause one SetFocus command to execute another SetFocus command as this may lead to unintended results.

Syntax: [bValue =] App.SetFocus (vFieldId, [bSaveRSP], [bValidateRSP])

bValue (Boolean) Optional – Returns True or False depending on the success of the command

vFieldId (Variant) is the prompt's Field Id or number receiving the focus.

bSaveRSP (Boolean) Optional – set to True to save any changes to the current prompts value before switching focus to the new prompt.

bValidateRSP (Boolean) Optional – set to True to call the edit checks and to re-run the OnEnter event.

Examples:

`App.SetFocus(5) ' Cursor will go to prompt #5`

`App.SetFocus("PartNo") ' Focus goes to PartNo prompt`

```
App.SetFocus("PartNo", True, True) ' Cursor will go  
to "PartNo", save current Text property and execute  
edits and OnEnter event
```

## **SetMenu**

This function will set the default menu for the current Client device. This is typically used to set an initial menu for an undefined user (i.e., you are bypassing the normal login process.)

Syntax: App.SetMenu(sName)

sName (String) is the default menu name for the connected Client device.

Example:

```
App.SetMenu("Sample")
```

## **SetValue**

This command will set the value of a field in the current recordset or of a specific prompt.

Syntax: App.SetValue(vField, vValue, [vName])

vField (Variant) to set the value or a prompt, or is the name of a field in the current recordset.

vValue (Variant) is the value to insert into the current recordset. (Note: all values are treated as strings until passed to the database.)

vName (Variant) Optional – is the name of the application's recordset to update. This is typically used to insert data into a calling application's recordset.

Examples:

```
App.SetValue(1, "FX1") ' Puts "FX1" in prompt #1.
```

```
App.SetValue("PartNo", "FX1", "Cycle")
```

Puts "FX1" in "PartNo" prompt on "Cycle" application.

## **ShowList**

This function causes a scrolling list to be displayed on the Client device and allows the user to make a selection. (Maximum entries: 32,767.)

Syntax: vValue = App.ShowList(sList, [bForceDisplay])

vValue (Variant) is the value that is returned when the user selects a specific list item.

sList (String) is the list to be returned for display (i.e., set RSP = Value to display the list on the Client device.)

bForceDisplay (Boolean) Optional – Use True or False to specify sending this command immediately rather than at the end of the current event. The default is False.

Example: See *App.MakeList* and *DB.MakeList*

## Sleep

This command will cause the Client device to sleep for the specified number of seconds. (Note: any activity by the user will terminate the sleep mode.) The Visual Basic Wait command may be a viable alternative.

Syntax: App.Sleep(nSeconds)

nSeconds (Long) is the number of seconds to suspend program operation. This is usually used to flash a message to the user, and then to erase it. (Note: The new timer feature is recommended for time related programming.)

Example:

`App.Sleep(1)`

## TimerEnabled

This command will enable / disable the ‘Form\_OnTimer’ event within the client session. It is a good idea to always disable the application level timer event when exiting the application. Use the Form\_Unload event or just before any App.CallForm command.

Note: see the Mobile Development Studio sample ‘FieldService’ VBA script for an example.

Syntax: App.TimerEnabled = bValue

Alternate: bValue = App.TimerEnabled

bValue (Boolean) is the state of the timer event. Set to ‘True’ to enable timer functions.

Example: See *TimerInterval*

## TimerInterval

This command sets the interval for the Form\_OnTimer event.

Syntax: App.TimerInterval = nValue

Alternate: nValue = App.TimerInterval

nValue (Long) is the number of milliseconds between timer events; i.e., for 10 seconds, set TimerInterval = 10000.

Example:

```
App.TimerEnabled = True  
App.TimerInterval = 1000
```

## User

This function can set or return the user of the current Client device. The default login screen sets this user. This id will then be used by any process that needs to know the logged in user.

Syntax: sUser = App.User()

Alternate: App.User = sUser

sUser (String) is the current user of the connected Client device.

Examples:

```
Dim sUser As String  
sUser = App.User  
App.User = "SAM"
```

## UserProperty

This function can set or return the value of the specified property on the currently logged in user's profile. This command references the App.User command to get the correct user and then looks up the parameter specified.

Syntax: vValue = App.UserProperty(vProperty)

Alternate: App.UserProperty(vProperty) = vValue

vValue (Variant) is the value stored under the property's value on the Users profile.

vProperty (Variant) is the property added to the current user's profile

Examples:

```
Dim vUserAge As Variant  
vUserAge = App.UserProperty("Age")  
App.UserProperty("Language") = "English"
```

## SQLNum

**This command is not part of the App object but falls into the same category.** This command will take a specified number, remove any decimals and replace commas with decimals. If you know a certain number format will not work in an SQL statement because the database expects English formatted numbers, use this command.

Syntax: sResult = SQLNum(vValue)

sResult (String) the converted string value suitable for use in SQL statements.

vValue      (Variant) a string or numeric value containing a non-English number representation that needs to be converted.

Example:

```
Dim sValue As String  
sValue = SQLNum("123.456,78") ' returns "123,456.78"
```

## Menu Strip Extensions

The Menu Strip is a soft key button panel that can appear at the bottom of the screen and be turned on or off using code or can be permanently left on or off as configured in the Theme resource. The F-Key configuration also has an option to assign an F-Key to toggle the display of the Menu Strip.

### AppendItem

This command will add an additional button to the menu strip. When clicked the RFgen\_OnMenu (found in RFgen.bas), Form\_OnMenu and then the prompt's OnMenu events will execute.

Syntax: MenuStrip.AppendItem(sAction, sDisplay, sImageName)

sAction      (String) is the ID of the button and also can execute one of the built-in commands. These are the internal command that can be used:

- Submit – means to enter the data appearing for the current prompt
- Refresh – refreshes the display screen
- Clear – clears the data entered for the current prompt
- Exit – exits the current process (same as F4 normally does if configured to do so).
- Search – executes the OnSearch event if one exists for the prompt

sDisplay      (String) is the text to be displayed on the button.

sImageName    (String) is the Image Resource ID to be displayed on the button.

Example:

```
MenuStrip.AppendItem("Submit", "Submit", "DownArrow")
```

This adds a button to the menu strip that performs the Submit action, displays the word 'Submit' and shows a down arrow icon on the button.

### Clear

This command will delete all the buttons on the menu strip.

Syntax: MenuStrip.Clear

Example:

```
MenuStrip.Clear
```

## Count

This command will return the number of buttons that are currently on the menu strip.

Syntax: MenuStrip.Count

Example:

```
Dim iCnt As Integer  
iCnt = MenuStrip.Count
```

## Item

This command will return the Action value of the button referenced by its index value.

Syntax: MenuStrip.Item(nIndex)

nIndex        (Long) is the index value of the requested button

Example:

```
Dim sName As String  
sName = MenuStrip.Item(4) ' the Exit button
```

If the forth button is the Exit button, "Exit" is returned.

## RemoveItem

This command will remove a button from the menu strip by referencing its index. Removing and item by its name would be more self-documenting and is preferred but this command is useful when iterating through a loop.

Syntax: MenuStrip.RemoveItem(nIndex)

nIndex        (Long) is the index value of the requested button

Example:

```
MenuStrip.RemoveItem(4) ' the Exit button
```

## RemoveItemByName

This command will remove a button from the menu strip by referencing its Action value.

Syntax: MenuStrip.RemoveItemByName(sAction)

sAction (String) is the Action name of the requested button

Example:

```
MenuStrip.RemoveItemByName ("Exit")
```

## Reset

This command will return the menu strip to the default settings configured in the Configuration / Environment Properties screen.

Syntax: MenuStrip.Reset

Example:

```
MenuStrip.Reset
```

## SetItem

This command will change an existing button to have different properties. For example, turn the Submit button into an Exit button.

Syntax: MenuStrip.SetItem(nIndex, sAction, sDisplay, sImageName)

nIndex (Long) is the index value of the requested button

sAction (String) is the ID of the button and also can execute one of the built-in commands. These are the internal command that can be used:

- Submit – means to enter the data appearing for the current prompt
- Refresh – refreshes the display screen
- Clear – clears the data entered for the current prompt
- Exit – exits the current process (same as F4 normally does if configured to do so).
- Search – executes the OnSearch event if one exists for the prompt

sDisplay (String) is the text to be displayed on the button.

sImageName (String) is the Image Resource ID to be displayed on the button.

Example:

```
MenuStrip.SetItem(4, "Submit", "Submit", "DownArrow")
```

## Show

This command will show or hide the menu strip.

Syntax: `MenuStrip.Show(bShow)`  
`bShow` (Boolean) Set to True or False to show or hide the menu strip

Examples:

```
MenuStrip.Show(False) ' hides the menu strip  
MenuStrip.Show(True) ' shows the menu strip
```

## Screen Display Extensions

Screen display commands typically interact with the user at the GUI level. These commands are used to print or remove text from the screen or gather information about the screen size.

### Bell

This command will cause the Client device terminal to beep. For telnet devices, the parameter is the number of times to sound the beep. For WinCE devices, this is the waveform to play. Windows CE specifies five default waveforms whose values are 0, 16, 32, 48 and 64. Each of these sounds must be setup on the CE device.

Syntax: `Screen.Bell([iNbr])`  
`iNbr` (Integer) Optional – is the bell sound.

Examples:

```
Screen.Bell(3) ' Telnet mode - bell rings 3 times.  
Screen.Bell(0) ' Graphical mode - OK sound  
Screen.Bell(16) ' Graphical mode - Hand sound  
Screen.Bell(32) ' Graphical mode - Question sound  
Screen.Bell(48) ' Graphical mode - Exclamation sound  
Screen.Bell(64) ' Graphical mode - Asterisk sound
```

### Clear

This command will clear the Client screen until the server needs to refresh the screen or unless a `Screen.Refresh` command is used. That will bring back any prompts that were in the area. Any text placed there with a `Screen.Print` command will be removed permanently.

Syntax: `Screen.Clear([bSendNow])`  
`bSendNow` (Boolean) Optional – clears the screen immediately if this option is set to True.

### ClearEOL

The Screen.ClearEOL command will clear the Client screen from the current position to the end-of-line. Using a Screen.Refresh will bring back any prompts that were in the area. Any text placed there with a Screen.Print command will be removed permanently.

Syntax: Screen.ClearEOL

## ClearEOP

This command will clear the text boxes of all user-entered fields on the devices terminal screen from the current position to the bottom of the screen. Using a Screen.Refresh will bring back any prompts that were in the area. Any text placed there with a Screen.Print command will be removed permanently.

Syntax: Screen.ClearEOP

## DrawLine

This command will draw a vertical line, horizontal line, or create a box on device screen. If you want to display boxes within boxes, you must draw them from the inside to the outside. This command works in both character and graphical mode. See SYS.UsePixels to reference a screen location by pixel.

Syntax: Screen.DrawLine(nStartCol, nStartRow, nEndCol, nEndRow, [bSendNow])

nStartCol (Long) is the column (x coordinate) in which to start drawing the line/box.

nStartRow (Long) is the row (y coordinate) in which to start drawing the line/box.

nEndCol (Long) is the column (x coordinate) in which to finish drawing the line/box.

nEndRow (Long) is the row (y coordinate) in which to finish drawing the line/box.

bSendNow (Boolean) Set to True if the method should paint the lines on the screen immediately.

Examples:

Screen.DrawLine(0,10,20,10) ' Draws a horizontal line

Screen.DrawLine(0,0,0,16) ' Draws a vertical line

Screen.DrawLine(0,0,20,16) ' Draws a box around the screen

Screen.DrawLine(0,0,20,16, True) ' Draws immediately

## Height

Returns the height in pixels or rows for the current application. (See Screen.Width).

Syntax: Value = Screen.Height

Value (Variant) equals the number of available rows in the current application or the height in pixels.

## Print

This command will move the cursor to a specified row and column, print text in normal or reverse video, and optionally clear to the end-of-line. In graphical mode, the permanent objects, prompts on the screen, will take precedence and the text will display behind the prompt. Making a prompt invisible and then using Screen.Print may be an alternate solution. As with all 'screen-printing' commands, you cannot use this command in the Form\_Load event because the application is not created until the Form\_Load event is finished.

Syntax: Screen.Print(nColumn, nRow, vText, [bReverse], [bClearEOL], [bSendNow])

nColumn (Long) is the column (x coordinate) in which to place the cursor.

nRow (Long) is the row (y coordinate) in which to place the cursor.

vText (Variant) is the text to print at the current row and column.

bReverse (Boolean) Optional – set to True to print text in reverse video. This only applies to a character based device, not graphical.

bClearEOL (Boolean) Optional – set to True to clear to the end-of-line after printing text.

bSendNow (Boolean) Optional – set to True to send the print packet to the Client device immediately

Example:

```
Screen.Print(0, 10, "F4 to Exit", , True, True)
```

## Refresh

This command will clear the Client screen and then repaint all standard prompts and data. (Note: any text displayed using Screen.Print statements will not be redisplayed.)

Syntax: Screen.Refresh

## ResetCursor

This command is for internal use only and has no use for the user.

## ReverseOff

This command is for internal use only and has no use for the user.

## **ReverseOn**

This command is for internal use only and has no use for the user.

## **Width**

Returns the width in pixels or columns for the current application. (See Screen.Height).

Syntax: vValue = Screen.Width

vValue (Variant) equals the number of available columns in the current application or the width in pixels.

Example:

```
If App.ClientType = "TE" Then  
    RFPrompt("Description").Length = Screen.Width
```

In this case, the text box containing the description will be the same width as the current application. Since the graphical mode would return pixels, setting the length of a textbox to a very large number would not be appropriate.

# **Soft Input Panel Extensions**

The soft input panel (SIP) is the small keyboard window that pops up when a field on the mobile (graphical capable) device allows for text input. The display and characteristics of this panel can be controlled from the script.

## **GetCurrentType**

This method returns a textual representation of the current input panel such as “Keyboard” or “Transcriber”.

Syntax: [sValue = ] SIP.GetCurrentType

sValue (String) – is the variable containing the result.

Example:

```
Dim sVal as String  
sVal = SIP.GetCurrentType
```

In this case sVal will have a value like “Keyboard”.

## **GetTypes**

This method returns a pipe ( | ) delimited list of available input types that are supported on the device. Not all devices will have the same list of available types. Here are a few examples:

Letter Recognizer	Block Recognizer	Phone Keypad
Compact QWERTY	Full QWERTY	WinCE Handwriting
WinCE Keyboard	Keyboard	Kana
Kensaku	Romaji	Tegaki

Syntax: sList = SIP.GetTypes

sList (String) – Contains a list of all available types of input that are supported on the device.

Example:

```
Dim sList as String  
sList = SIP.GetTypes
```

## Mode

This property is used to set one or modes together to create the proper input. For standard English use the Roman mode. For other variations a combination may be required.

Syntax: [bOK =] SIP.Mode(enMode)

bOK (Boolean) – Optional – Returns True if the command was successful.

enMode (enSIPMode) – are the individual properties that can combine to make up the proper input panel. They are:

- enSIP\_CHARCODE
- enSIP\_CHINESE
- enSIP\_FULLSHAPE
- enSIP\_HANGUL
- enSIP\_JAPANESE
- enSIP\_KATAKANA
- enSIP\_LANGUAGE
- enSIP\_NATIVE
- enSIP\_ROMAN

Examples:

```
Dim bOK as Boolean  
bOK = SIP.Mode(enSIP_ROMAN)  
bOK = SIP.Mode(enSIP_ROMAN or enSIP_FULLSHAPE or  
enSIP_KATAKANA or enSIP_NATIVE)
```

In the second example, several modes are ORed together to create the Katakana input panel.

## **SetType**

This method sets the current input panel. The Input Method parameter can either be text as it is returned in the GetTypes method or it can be a zero-based number referring to the same list. This list is the same as the dropdown list on the mobile device where you choose between the styles of recognition.

Syntax: [bOK = ] SIP.SetType(vIM)

bOK (Boolean) – Optional – Returns True if the command was successful.

vIM (Variant) – set to a number or a string representing one of the available recognition styles.

Examples:

```
Dim bOK as Boolean  
bOK = SIP.SetType(0)' 0 may not represent the  
keyboard  
bOK = SIP.SetType("Keyboard")
```

## **Show**

This method is used to display or hide the SIP.

Syntax: [bOK = ] SIP.Show(bShow)

bOK (Boolean) – Optional – Returns True if the command was successful.

bShow (Boolean) – set to True for the SIP to display, set to False for the SIP to be hidden.

Example:

```
Dim bOK as Boolean  
bOK = SIP.Show(True)
```

# **Server-Based Extensions**

Server-based commands are used by the mobile device when not in a Thin-client state to send and receive data to and from the server.

## **CallMacro**

This function is used to call a screen mapping or transaction macro and pass the macros required parameters. Using this function gives you the ability to “store-and-forward” transactions while the host is off-line for backup or other reasons. It returns ‘True’ if the macro was successfully completed.

Syntax: [enOK =] Server.CallMacro(sMacroName, bQueueOffline, [vParams])

enOK	(enMacroResults) Optional – Returns 1 of the following 4 values: MacroFailed MacroNotProcessed MacroQueued MacroSucceeded
sMacroName	(String) – This is the name of the macro to be called.
bQueueOffline	(Boolean) – This determines whether the macro can be queued for later processing if the host is not currently available.
vParams	(Variant) – Optional: A series of passing parameters as required by the selected macro. Note: these parameters are the fields you defined when you record the macro.

Example:

```
Dim bOK As enMacroResults
bOK = Server.CallMacro("PICK", True, "Sam", "12")
```

## CommandTimeout

This command limits the number of seconds any Server command waits for a response from the server. All Server commands will return with a response or failure message no later than the specified number of seconds. Set the parameter to zero (0) to disable this feature and go back to the default for each command.

Syntax: Server.CommandTimeout(nTimeout)

nTimeout (Long) parameter to specify a number of seconds

Example:

```
Server.CommandTimeout(10)
```

## Connect

This command connects to the remote server via TCP/IP. Default values are stored in the registry when the files are installed based on the profile.

Syntax: [bOK =] Server.Connect

bOK	(Boolean) Optional – return a True if the mobile device is able to make a connection to the host within 30 seconds or less.
-----	---

Examples:

```
Dim bOK as Boolean
```

```
bOK = Server.Connect
```

## Disconnect

This command disconnects from the remote server. You should always use this command when you know the connection is no longer needed. Using the Form\_Unload event may be a good place.

Syntax: Server.Disconnect

## ExecuteSQL

This function will execute a pass-through SQL statement on the host through the connection maintained by the server. Any results from the statement will be returned in a text-delimited object. (Note: this means that the item is a static view of the database and cannot be updated.)

Syntax: [bOK =] Server.ExecuteSQL(sSQL, [vColumns], [vRows])

bOK      (Boolean) Optional – is a return value; a value of True means the SQL statement processed normally.

sSQL      (String) is the SQL statement to be sent to the database. As this is a pass through statement, its syntax must be understood by the database, as no pre-processing will occur.

vColumns(Variant) – Optional – is a string representation of the columns returned by the SQL statement.

vRows      (Variant) – Optional – is a string representation of the static result of an SQL statement.

Example:

```
Dim bOK As Boolean
Dim sSQL As String
Dim sCols As String
Dim sRows As String
sSQL = "select * from Inventory"
bOK = Server.ExecuteSQL(sSQL, sCols, sRows)
```

In the case of an insert or an update, the sCols and sRows variables would not be necessary.

## GetTable

This command executes the SQL statement on the remote server and copies the results to an existing local table.

Syntax: [bOK =] Server.GetTable(sSQL, sLocalTable, [vKeyFields])

bOK      (Boolean) Optional – is a return value; a value of True means the SQL statement processed normally.

- sSQL (String) is the SQL statement to be sent to the database. As this is a pass through statement, its syntax must be understood by the database, as no pre-processing will occur.
- sLocalTable(String) is the name of the table already created on the mobile device.
- vKeyFields (Variant) Optional – is 1 or more key fields to represent a unique record. One key would be represented by “PartNo” and multiple keys look like “PartNo,Onhand”.

Examples:

```
Dim bOK as Boolean  
bOK = Server.GetTable("Select * from Inventory",  
"Inv2", "PartNo")
```

Note: In the event that you need to update data using the Key field you should pay close attention to how the table has been populated. To optimize update performance you should try to match the order in which the data was populated in the table initially. For instance assume table INVENTORY has a unique ID column. Also assume that you will need to update a small subset or rows on the device database using the Server.GetTable() language extension. You would want to initially populate the table using the GetTable function with a SQL statement similar to:

```
Server.GetTable("SELECT * FROM INVENTORY ORDER BY  
Id", "INVENTORY")
```

To optimize performance while updating records you would want to match the data that way it was downloaded by using the ORDER BY clause in your update like:

```
Server.GetTable("SELECT Id, Column1, Column2 FROM  
INVENTORY WHERE LastWrite > '07/22/2013' ORDER BY Id",  
"INVENTORY", "Id")
```

The ORDER BY clause will ensure that the records in the initial download and the updates are in the same order and will optimize the databases ability to seek to the next record to update.

## IsConnected

This command returns a Boolean value of True or False depending on if the Server.Connect command was previously successfully executed.

Syntax: bOK = Server.IsConnected

bOK       (Boolean) a value of True means there is already an open connection to use.

Example:

```
Dim bOK as Boolean  
bOK = Server.IsConnected
```

## MakeList

This command returns a dynamic array string containing the results of a SQL statement that was executed on the server from a mobile device. For this command to be successful the server must be in wireless range for a connection to be established.

Syntax: [bOK   =]   Server.MakeList(sSQL,   sList,   [bRtnAllCols],  
[bNormalize])

bOK       (Boolean) Optional – is a return value; a value of True means the SQL statement processed normally.

sSQL       (String) is the SQL 'SELECT' statement to be sent to the database.

sList       (String) is the list to be returned for display (i.e., set RSP = Value to display the list on the Client device.)

bRtnAllCols (Boolean) Optional – when set to True will return all the columns as the potential key, not just the first column. Default is False.

bNormalize (Boolean) Optional – when set to True will trim the spaces from the data so that it will display consistently. Default is False.

Example:

```
Dim sSQL As String  
Dim sMyList As String  
sSQL = "select PartNo from ItemMaster"  
sMyList = Server.MakeList(sSQL, True, True)  
Rsp = App.ShowList(sMyList)  
Or  
Rsp = App.ShowList(Server.MakeList(sSQL, True, True))
```

## Ping

This command returns a True / False regarding the ability to reach the server. Based on the result you may choose to continue with the Server.Connect or go to a disconnected state. Default values are stored in the registry when the files are installed.

Syntax: bOK = Server.Ping([vServer])

- bOK (Boolean) returns a True if the mobile device could connect to the server.  
vServer (Variant) Optional – to specify the name or IP address of the server. If this is not specified, the registry settings will be used.

Example:

```
Dim bOK As Boolean  
bOK = Server.Ping("192.168.123.45")
```

## QueueMacro

This function is used to queue any transaction macro and pass its required passing parameters directly to the server while in mobile mode. Unlike the TM.CallMacro with the queue property set to True, this command will not check for a valid host connection, or move to a linked screen. It returns 'True' if the macro was successfully queued. These macros can be created on the Transactions tree.

Syntax: [nSeq =] Server.QueueMacro(sMacroName, [vParams])  
nSeq (Long) Optional – Returns the sequence number of the macro when it is successfully queued.  
sMacroName (String) – This is the name of the macro to be called.  
vParams (Variant) Optional – A series of passing parameters as required by the selected macro. Note: these parameters are the fields you defined when you wrote the macro.

Example:

```
Dim nSeq As Long  
nSeq = Server.QueueMacro("ChangeUser", "Sam", "1234")
```

## ReadFile

This command will read data from a file on the server. String data or binary data can be read.

Syntax: [bOK =] Server.ReadFile(sFileName, vData)  
bOK (Boolean) Optional – the success or failure of the command.  
sFileName (String) specifies where on the server the file should be found  
vData (Variant) contains the data read from the file. String or binary data is allowed.

Example:

```
Dim bOK As Boolean  
Dim vData As Variant  
Dim sFile As String
```

```
sFile = "C:\Program Files\MyFile.txt"  
bOK = Server.ReadFile(sFile, vData)
```

## SendQueue

This function sends any locally queued transactions to the remote server queue for processing. Upon successfully transferring the local queue to the server, the local queue is cleared.

Syntax: [bOK =] Server.SendQueue([vDestQueue])

bOK           (Boolean) Optional – Returns True if the queued macros were successfully sent to the host for processing.

vDestQueue (Variant) Optional – an option to send the queue to a queue with a different name

Example:

```
Dim bOK As Boolean  
bOK = Server.SendQueue
```

## SendTable

This function executes the SQL statement locally and copies the results to an existing table on the remote server.

Syntax: [bOK =] Server.SendTable(sSQL, sRemoteTable, [vKeyFields])

bOK           (Boolean) Optional – is a return value; a value of True means the SQL statement processed normally.

sSQL           (String) is the SQL statement to be sent to the database. As this is a pass through statement, its syntax must be understood by the database, as no pre-processing will occur.

sRemoteTable (String) is the name of the table already created on the mobile device.

vKeyFields    (Variant) Optional – is one or more key fields to represent a unique record. One key would be represented by “PartNo” and multiple keys look like “PartNo,Onhand”.

Example:

```
Dim bOK as Boolean  
bOK = Server.SendTable("Select * from Inv2",  
"Inventory", "PartNo")
```

## SetCredentials

This function sets or changes the Domain, User or Password values for connecting to the server's network when using the NTLM connection security authentication option.

Syntax: [bOK =] Server.SetCredentials([vDomain], [vUserID],  
[vPassword])

bOK (Boolean) Optional – is a return value; a value of True means the method processed normally.  
vDomain (Variant) Optional – is the name of the domain where the credentials will be validated  
vUserID (Variant) Optional – is the name of the user  
vPassword (Variant) Optional – is the password of the user

Example:

```
Dim bOK as Boolean  
bOK = Server.SetCredentials ("RFGEN", "MIKE", "PASS")
```

## SetHost

This function sets the default settings for connecting to the host server. Typically Server.Connect follows this command.

Syntax: [bOK =] Server.SetHost([vServerName], [nPort])

bOK (Boolean) Optional – is a return value; a value of True means the method processed normally.  
vServerName (Variant) Optional – is the name or IP address of the server  
nPort (Long) Optional – is the port that facilitates the data portion of the communication.

Example:

```
Dim bOK as Boolean  
bOK = Server.SetHost ("192.168.1.100", 21097)
```

## SetVPN

This function lets the user change the credentials used when using the operating system's VPN settings to establish server access.

Syntax: [bOK =] Server.SetVPN(bEnabled, [vVPNName], [vUserID],  
[vPassword])

bOK (Boolean) Optional – is a return value; a value of True means the method processed normally.  
bEnabled (Boolean) enables or disables the configured VPN  
vVPNName (Variant) Optional – is the name of the VPN as configured in the operating system

vUserID	(Variant) Optional – is the user ID to be used for making that VPN connection if it is different than the already configured user ID in the VPN setup
vPassword	(Variant) Optional – is the password to be used for making that VPN connection if it is different than the already configured password in the VPN setup

Example:

```
Dim bOK as Boolean
bOK = Server.SetVPN(True, "MyISP")
```

## SetWAN

This function lets the user change the credentials used when using the operating system's GPRS (cellular) settings to establish internet access.

Syntax: [bOK =] Server.SetWAN(bEnabled, [vWANName], [vUserID], [vPassword])

bOK	(Boolean) Optional – is a return value; a value of True means the method processed normally.
bEnabled	(Boolean) enables or disables the configured WAN
vWANName	(Variant) Optional – is the name of the WAN as configured in the operating system
vUserID	(Variant) Optional – is the user ID to be used for making that WAN connection if it is different than the already configured user ID in the WAN setup
vPassword	(Variant) Optional – is the password to be used for making that WAN connection if it is different than the already configured password in the WAN setup

Example:

```
Dim bOK as Boolean
bOK = Server.SetWAN(True, "MyWAN")
```

## ShowProgress

This command enables or disables a popup progress box for all Server commands. The elapsed number of seconds and the current activity are displayed.

Syntax: Server.ShowProgress(bShow)

bShow (Boolean) True or False for turning on or off the progress bar

Example:

```
Server.ShowProgress(True)
```

## SyncApps

This command will update a mobile device configured for mobile operation with changes made to application related items such as menus, users, applications, macros etc. If an administrator changes a device profile to include or exclude items, this command will compare what is on the device with what is in the profile and request any changed or missing items. Depending on the size of the change the Server.ShowProgress command may be useful.

Syntax: [bOK = ] Server.SyncApps([vProfile])

bOK      (Boolean) Optional – returns True or False for the success of the command

vProfile (Variant) Optional – is the name of the profile used to compare what is on the device with the server list of objects.

Examples:

```
Dim bOK As Boolean  
bOK = Server.SyncApps  
Server.SyncApps ("CEProfile1")
```

## WriteFile

This command will write a file to the server hard drive and can be used from both the Thin client mode and Mobile client mode.

Syntax: [bOK = ] Server.WriteFile(sFileName, vData, bOverwrite)

bOK      (Boolean) Optional – returns True or False for the success of the command

sFileName (String) is the path and file name for the file being created or overwritten.

vData      (Variant) the contents of the file which can be either a string or byte array

bOverwrite (Boolean) set to True, the server will overwrite an existing file, False will return a failure because the file already existed

Example:

```
Dim bOK As Boolean  
Dim sData As String  
Dim sPath As String  
  
sPath = "C:\MyFile.txt"  
sData = "Sample Text"  
bOK = Server.WriteFile(sPath, sData, True)
```

# System Error Extensions

This object contains application error information that is also written to the error log. This object does not trap VB errors such as a type mismatch but application level issues such as a macro failing. For the VB errors see the Err object.

## AddError

Adds an error to the collection in the SysErr object. You must specify the error number, native error number and a description of the error.

Syntax: SysErr.AddError(nErrNo, nNativeError, sDesc)

nErrNo (Long) the VBA error number

nNativeError (Long) the native database error number

sDesc (String) the description of the error

Example:

```
SysErr.AddError(1269, -2847638354, "TCP NOT AVAIL")
```

## AppName

This property contains the name of the current application when the error occurred.

Syntax: sName = SysErr.AppName

sName (String) the name of the faulting application as shown in the error log

Example:

```
Dim sName As String  
sName = SysErr.AppName
```

## Clear

Clears all property settings of the SysErr properties. Use this command after trapping and dealing with an error so the next error can be trapped.

Syntax: SysErr.Clear

Example:

```
SysErr.Clear
```

## Count

This property returns the number of errors. Occasionally an ODBC driver may return multiple errors and the first one may not be the most descriptive or accurate for solving the problem. Concatenating all the

errors for the message box will give the most complete description of the problem.

Syntax: vValue = SysErr.Count

vValue (Variant) is the number of errors in the collection.

Example:

```
Dim vCount As Variant  
vCount = SysErr.Count
```

## Description

This property returns a string description of the ODBC error. Using the SysErr.Count command you may specify the number of the error in a loop and extract each error for display to the user.

Syntax: sDesc = SysErr.Description([vIndex])

sDesc (String) is the error description.

vIndex (Variant) Optional – is the error number to view. Note: this value is zero-based and defaults to 1.

Examples:

```
Dim sDesc As String  
sDesc = SysErr.Description(0)  
  
Dim i As Integer  
For i = 0 To SysErr.Count - 1  
    sDesc = sDesc & SysErr.Description(i) & vbCrLf  
Next
```

## DevGUID

This property contains the device GUID value used by the server to uniquely identify a device with a session. This value can be displayed from the Enterprise Dashboard if necessary.

Syntax: sValue = SysErr.DevGUID

sValue (String) is the device GUID identifier

Example:

```
Dim sValue As String  
sValue = SysErr.DevGUID
```

## IpAddress

This property contains the IP address of the device. This value can be displayed from the Enterprise Dashboard if necessary.

Syntax: sValue = SysErr.IpAddress  
sValue (String) is the IP address of the device

Example:

```
Dim sIP As String  
sIP = SysErr.IpAddress
```

## NativeError

This property returns a numeric value specifying the native database error number.

Syntax: vValue = SysErr.ErrNative([vIndex])

vValue (Variant) is the database error number.

vIndex (Long) Optional – is the error number to view. Note: this value is zero-based and defaults to 1.

Example:

```
Dim vError As Variant  
vError = SysErr.NativeError(0)
```

## Number

This property returns a numeric value specifying the VBA error number.

Syntax: vError = SysErr.Number([vIndex])

vError (Variant) is the VBA error number.

vIndex (Long) Optional – is the error number to view. Note: this value is zero-based and defaults to 1.

Example:

```
Dim vError As Variant  
vError = SysErr.Number(0)
```

## UserName

This property contains the user logged in that experienced the error.

Syntax: sName = SysErr.UserName

sName (String) is the user name of the device when the error occurred

Example:

```
Dim sUser As String  
sUser = SysErr.UserName
```

# System Level Extensions

System level commands get or set environmental properties for users, data connections or other global system settings.

## ConnectionProperty

This function will return a characteristic of a data source. Some properties only refer to certain types of data connections. For example, dcDatabase would refer to an ODBC connection where dcHostPort would refer to a screen mapping connection. This is read-only.

Syntax: vValue = SYS.ConnectionProperty(vSource, enProp)

vValue (Variant) set to the value of the specified property

vSource (Variant) is the string representation or the numeric representation of the database, screen mapping host or ERP session

enProp (enDCProps) the following list of properties is available

dcAppGroup	(SAP)	Application Group field
dcAppName	(SM)	For Windows Console mode, this is the Application Executable field
dcAutoWrap	(SM)	Wrap text at end-of-line (VT220 only)
dcBackColor	(SM)	Back color of telnet emulator, it returns a numeric value corresponding to VB colors
dcBlockSize	(DB2)	Block Size field
dcCatalog	(DB2)	Initial Catalog field
dcCCSID	(DB2)	CCSID Conversion flag, returns a 0 (False) or a 1 (True)
dcCFIT	(SAP)	Conversion Fault Character field
dcClientNo	(SAP)	Application server's Client number
dcCmdLine	(SM)	For Windows Console mode, this is the Command Line field
dcCompany	(Dynamics)	Company field
dcConfigFile	(ERP)	Axapta and Dynamics connections, this is the Configuration File field
dcConnCheck	(DB)	Validation Check field for all database connections. Returns a 0 (No) or a 1 (Yes)
dcConnectMode	(All)	Returns User Database, Enterprise Connection, Screen Mapping Connection, or Web Service
dcConnectTimeout	(Web)	Connect Timeout field
dcConnectType	(All)	Returns the type of connection. Values are: Access, Axapta, DB2,

		Dynamics, JDE Enterprise One, ODBC, OleDB, Oracle, SAP R/3, SQL Server, SQLite, TN3270, TN5250, VT220, Web Services, Windows Exe.
dcConnectionString	(DB)	Returns the provider string used to make the connection to the database. A portion of the string may be the path and file name of a file or one or more configuration fields combined together.
dcCursorType	(DB)	The cursor type or locking state used to make the connection (eg: Optimistic / Pessimistic)
dcDatabase	(DB)	For DB2 it is the Default Library field, for SQL Server it is the Database Name field.
dcDatabaseOwner		(For future use)
dcDatabaseType	(DB)	For specific database types it returns the database name. For OleDB it returns the Database field value.
dcDataStream	(Other)	For SAP it returns the Data Format field, 0 (ASCII) or 1 (Unicode). For Screen Mapping, vt220 mode it returns the Data Stream field, 0 (Standard) or 1 (UTF-8)
dcDestructiveBS	(SM)	For VT220 only, it returns the Destructive Backspace Boolean check box option
dcDeviceName	(SM)	Device Name field for TN5250 and TN3270 only
dcDownTime	(All)	Returns the Schedule Downtime configuration for all data connector types.
dcDriver		(For future use)
dcDSN	(DB)	For ODBC type only, it returns the Data Source field value.
dcEBCDIC	(SM)	Returns the Code Page field for TN5250 and TN3270 only
dcErrorCodes	(DB)	Reset on Error field values are returned for all database connector types.

dcExtendedProps	(DB)	Returns the Extended Properties list for all database connectors except ODBC.
dcFileName	(DB)	Returns the Database File name field value for Access, SQL Server and SQLite data connectors
dcFontSize	(SM)	Font Size field used to display the host screen in the telnet emulator or the HostScreen control for Windows Console mode.
dcForceKeyPacket	(SM)	For VT220 only, it returns the Send Whole Key Packets Boolean check box option
dcForeColor	(SM)	Display ForeColor used to display the host screen in the telnet emulator or the HostScreen control for Windows Console mode.
dcGWHost	(SAP)	Gateway Host field
dcGWSERVICE	(SAP)	Gateway Service field
dclgnoreCert	(Web)	Ignore Invalid Certificates Boolean check box field
dclndexDatabase	(DB)	Index Selected Databases Boolean check box field for all database connector types.
dclndexList	(DB)	Tables Specified for Indexing list of values all database connector types.
dclsJDE	(DB)	Returns the JD Edwards Database Indexing Boolean check box on the Indexing Options tab for all database connector types.
dcLanguage	(SAP)	Returns the Language field, used to log in to the ERP system
dcLibraryList	(DB2)	Returns the Catalog Library List field value
dcLocale	(SM)	Returns the numeric value of the Display Language field
dcLocalEcho	(SM)	For VT220 only, it returns the Echo Characters Locally Boolean check box option
dcLogicalName	(SQL Svr)	Returns the Logical Name field value for the SQL Server database type only

dcLoginPage	(All)	Returns Login Mode option on the Login Options tab (0 = Automatic, 1 = Manual)
dcMsgServer dcOnCCE	(SAP) (SAP)	Message Server field returns the Character Conversion option (0 = Abort, 1 = Copy, 2 = Replace)
dcPackageName	(DB2)	Returns the SQL Package Name field value
dcPadFields	(SM)	For TN5250 and TN3270 only, returns the Pad fields When Sending Data Boolean check box value
dcPassword	(All)	Returns the Password field from most of the data connectors. Some data connector configurations do not have a password field.
dcPoolData	(All)	Returns the list of users and passwords in the Connection Pooling Named Users grid
dcPooled	(All)	Returns the Pooling Status field value (0 = Disabled, 1 = Enabled)
dcPoolMax	(All)	Returns the Allocated Licenses number
dcPort	(SM)	For TN5250, TN3270, and VT220 only, this returns the Telnet Port field value
dcProvider	(DB)	Returns the Provider Name field value for database types that have one.
dcQueryGoal	(DB2)	Returns the Query Optimize Goal field value
dcQueryTimeOut	(DB)	Returns the Query Timeout value configured for any database connector
dcR3Name	(SAP)	R/3 Name field
dcReadDataRows	(ERP)	Maximum SQL Rows to be returned from the ERP connector
dcReceiveTimeout	(Web)	Returns the Receive Timeout field value
dcResetOnFailure		(For future use)
dcRouter	(SAP)	Router String field
dcSendCRLF	(SM)	For VT220 only, Returns Send Return + Line Feed Boolean field value

dcSendTimeout	(Web)	Send Timeout field
dcServer	(All)	Returns the Host Server / Name field for any connector that has one
dcSessionTimeout	(All)	Returns the Session Timeout field for all connection types
dcSourceId	(All)	The Source Id field that names the connection
dcStartMenu	(SM)	Returns Session Default Main Menu option
dcStartMenuLocked	(SM)	Returns 1 if the data connection is locked to a specific menu rather than the default of any main menu. This is found in the Login Options tab, Connection Pooling Named Users grid but only applies at runtime.
dcSystemDatabase		(For future use)
dcSystemNo	(SAP)	System Number field
dcTraceMode	(SM)	Returns the Enable Packet Trace Boolean check box field value
dcTrimFields	(SM)	For TN5250 and TN3270 only, returns the Trim Fields When Retrieving Data Boolean field value
dcTrusted		(For future use)
dcUseFile	(DB)	Returns 1 for SQLite and Access and SQL Server only if a file is selected instead of a database server.
dcUseHTTPS	(Web)	Returns the Connect Using HTTPS Boolean field value
dcUser	(All)	Returns the User ID field for any connector that uses login credentials.
dcUseSSH	(SM)	For VT220 only, returns the Connect via SSH Boolean check box option
dcViewOwner	(DB)	Returns the Include Owner Name in Indexes Boolean check box field
dcViewSource	(DB)	Returns the Include Source Name in Indexes Boolean check box field
dcWinDomain	(ERP)	For Axapta and Dynamics only, returns the Windows Domain field
dcWinProxy	(ERP)	For Dynamics only, returns the Proxy Domain field

dcWinPwd	(ERP)	For Axapta and Dynamics only, returns the Proxy Password or Domain Password field
dcWinUser	(ERP)	For Axapta and Dynamics only, returns the Proxy User or Domain User field
dcWorkDir	(SM)	For the Windows Console Type only, returns the Working Directory field value.

Examples:

```
Dim vValue As Variant
vValue = SYS.ConnectionProperty(1, dcUser)
vValue = SYS.ConnectionProperty("RFSample", dcUser)
```

## DeleteProperty

This function will delete a property (or all properties) contained within the collection specified by the key. SYS SetProperty is used to add new properties to a collection.

Syntax: SYS.DeleteProperty(vKey,v ID)

vKey (Variant) is the main key. It is the name of the collection of properties that are to be grouped together.

vID (Variant) is the property key. If an '\*' (asterisk) is used, all properties within the collection will be deleted.

Examples:

```
SYS.DeleteProperty("Counts", "LastIssue")
SYS.DeleteProperty("Counts", *)
```

## DisableTimeout

This function will allow the session to continue even when the server's Client Inactivity Timeout value has been reached. In essence, this session will never time out.

Syntax: SYS.DisableTimeout(bValue)

bValue (Boolean) set to True to enable (eliminate the Client Inactivity Timer) or False to return to normal monitoring.

Examples:

```
SYS.DisableTimeout(True)
SYS.DisableTimeout(False)
```

## EnvironmentProperty

This function will return the value of the assigned property set in the Configuration / Environment Properties / System Environment Properties grid.

Syntax: vValue = SYS.EnvironmentProperty(vProperty)

vValue (Variant) is the value stored in the System Environment Properties for the Property specified.

vProperty (Variant) is the property specified in the Environment Properties grid.

Example:

```
Dim vPlant As Variant  
vPlant = SYS.EnvironmentProperty("PlantName")
```

## GetConnection

This is a specialized method similar to the EmbeddedProc object that could be used for getting the connection object to Microsoft Dynamics. The Dynamics client must be installed on the same machine. Use the ERP.BeginTrans and ERP.CommitTrans before and after the use of this method. This exposes the business functions but the programmer must know how to use them.

Syntax: oValue = SYS.GetConnection(vSource)

oValue (Object) An object declared as Axapta3 or Object, if you choose to use late binding. It will return the ADO, RDO, OLEDB, ERP, or Web connection object

vSource (Variant) data source name (DSN) or the data source number

Example:

```
Dim oConn As Axapta3  
Dim oRecord As IAxaptaRecord  
  
Set oConn = SYS.GetConnection("Dynamics")  
  
Set oRecord =  
oConn.CreateRecord("INVENTJOURNALTABLE")  
If oRecord Is Nothing Then  
    App MsgBox "Error: Axapta Record object failed."  
    Exit Sub  
End If  
App MsgBox "Record company: " & oRecord.company
```

## GetProperty

This function will return a property value that has been set using SYS SetProperty. SYS.GetProperty and SYS SetProperty can be used

in place of making function calls to an INI file or to the System Registry. The keys and data are stored in the solution MDB file.

Syntax: vValue = SYS.GetProperty(vKey, vID)

vValue (Variant) is the property value.

vKey (Variant) is the main key. It is the name of the collection of properties that are to be grouped together.

vID (Variant) is the property key. It is the reference to the value being returned. If an "\*" (asterisk) is used as the ID, a Chr(1) delimited list of properties contained within the collection will be returned.

Examples:

```
Dim vTransfersCt As Variant  
Dim vIssuesCt As Variant  
vTransfersCt = SYS.GetProperty("Counts", "LastTfr")  
vIssuesCt = SYS.GetProperty("Counts", "LastIssue")
```

## SendMessage

This function sends a message to a specific device number currently connected to the server. Device numbers of users can be obtained by using the SYS.UserList command.

Syntax: SYS.SendMessage(iDevNo, sMessage)

iDevNo (Integer) the ID of the mobile device

sMessage (String) the message to send to the other device

Example:

```
SYS.SendMessage(4, "Please see Sam immediately.")
```

## SetProperty

This function will save a property value for future retrieval using SYS.GetProperty. Property values will persist between sessions because they are saved in the solution MDB file. SYS.GetProperty and SYS SetProperty can be used in place of making function calls to an INI file or to the System Registry. Use this command with caution. The solution database is not designed for heavy adding and deleting of properties. Microsoft Access tends to grow over time as records are added and deleted.

Syntax: SYS SetProperty(vKey, vID, vValue)

vKey (Variant) is the main key. This will be the identifier for all like properties and their values.

vID (Variant) is the sub key. This key is referenced to obtain its value

vValue (Variant) is the property value.

Examples:

```
Dim nIssuesCt as Long  
nIssuesCt = 200  
SYS SetProperty("Counts", "LastTxr", 500)  
SYS SetProperty("Counts", "LastIssue", nIssuesCt)
```

## UsePixels

This function will toggle how the server identifies locations on the client screen. For telnet mode the screen is referenced by columns and rows of text. For graphical devices the preferred method is in pixels. This command can be used to switch between character and pixel positioning.

Syntax: SYS.UsePixels(bValue)

bValue (Boolean) Set to True to identify screen locations by pixels;  
False to use character based columns and rows

Example:

```
SYS.UsePixels(True)
```

## UserList

This function will return the device number, user and application currently in use. The purpose of this command is to log or locate a user doing a transaction, possibly for sending just the one device a message over the network. (See SYS.SendMessage)

Syntax: sUserList = SYS.UserList([bOnlyLoggedIn])

sUserList (String) contains the device number, logged in user and application of all devices in use at the time the command was issued.

bOnlyLoggedIn (Boolean) Optional – when set to True only returns entries for devices that have a logged in user. The default is False and this returns all connected devices to the server regardless of a user logged in. This could be used to generate a list of all connected devices so that they may all receive a message on the screen. The default is False.

Examples:

```
Dim sUserList As String  
sUserList = SYS.UserList  
sUserList = App.ShowList(SYS.UserList(True), True)
```

## ValidateWinUser

This function will validate user credentials against the Windows domain or local system. The user account must not be inactive.

Syntax: bOK = SYS.ValidateWinUser(sDomain, sUser, sPassword)

bOK	(Boolean) returns True if the validation was a success
sDomain	(String) Enter the domain to be searched. If the local PC is used, specify a single period for the domain or enter the PC name.
sUser	(String) Enter the user name to be validated.
sPassword	(String) Enter the password to be validated. Passwords are case sensitive. Users that do not have a password will not work.

Example:

```
Dim bOK As Boolean  
SYS.ValidateWinUser(".", App.User, gsPass)
```

In this example the password was saved in a global string variable when it was used on the login screen.

## Database Related Extensions

Database related commands send and receive data to and from ODBC data connections. When operating in a mobile environment, the SQL statements are directed to the local database on the mobile device.

### BeginTrans

This function begins a new transaction. The server will track any changes made to the database until either a DB.CommitTrans or DB.RollbackTrans are executed. You cannot have a second DB.BeginTrans for the same data source before committing or rolling back the first one. On a mobile device, there is only 1 data connection and it must be committed or rolled back before another DB.BeginTrans is used.

Syntax: [bValue =] DB.BeginTrans(vSource)

bValue (Boolean) Optional – indicates success or failure. If this command should fail because the connection is not available, the remainder of the code should not be executed. There may be unreliable results.

vSource (Variant) data source name (DSN) or the data source number

Examples:

```
DB.BeginTrans ("RFSample")
```

`DB.BeginTrans(1)`

## CommitTrans

This function saves any changes to the database that were started after the DB.BeginTrans command was executed and ends the current transaction.

Syntax: [bValue =] DB.CommitTrans(vSource)

bValue (Boolean) Optional – indicates success or failure

vSource (Variant) data source name (DSN) or the data source number

Examples:

`DB.CommitTrans("RFSample")`

`DB.CommitTrans(1)`

## Count

This function will return the number of rows contained in a specified rows item returned from the DB.Execute function or any Dynamic Array.

Syntax: vNbr = DB.Count(sRows)

vNbr (Variant) is the number of rows contained in the Records item.

sRows (String) is the string representation of a static recordset generated by a SQL statement using the DB.Execute function.

Example:

```
Dim sSQL As String
Dim sCols As String
Dim sRows As String
Dim iNbr As Integer
sSQL = "select * from ItemMaster"
DB.Execute(sSQL, sCols, sRows)
iNbr = DB.Count(sRows)
```

## Execute

This function will execute a pass-through SQL statement on the host through the connection maintained by the server. Any results from the statement will be returned in a text-delimited object. (Note: this means that the item is a static view of the database and cannot be updated.)

Syntax: [vErrNo =] DB.Execute(sSQL, [vColumns], [vRows])

vErrNo (Variant) Optional – is a return value; a value of '0' (zero numeric) means the SQL statement processed normally.

sSQL (String) is the SQL statement to be sent to the database. As this is a pass through statement, its syntax must be

- understood by the database, as no pre-processing will occur.
- vColumns (Variant) Optional – is a string representation of the columns returned by the SQL statement. This is optional because Insert, Update, and other statements do not return data.
- vRows (Variant) Optional – is a string representation of the static rows of an SQL statement. This is optional because Insert, Update, and other statements do not return data.

Example:

```
Dim sSQL As String
Dim sCols As String
Dim sRows As String
sSQL = "select * from Inventory"
DB.Execute(sSQL, sCols, sRows)
```

In the case of an insert or an update, the sCols and sRows variables would not be necessary because no recordset is returned.

## Extract

This function will extract from a recordset the one value at the specified column and row. A specific column and row intersect at a single value.

Syntax: vValue = DB.Extract(sColumns, sRows, iRecNo, vFieldNo)  
 vValue (Variant) is the value extracted. (Note: this will always be a string value regardless of the actual field's data type.)  
 sColumns(String) is the string variable used to hold the list of columns in the DB.Execute command.  
 sRows (String) is the string variable used to hold the rows of data in the DB.Execute command.  
 iRecNo (Integer) is the row number within the retrieved recordset to extract data from.  
 vFieldNo (Variant) is the column number (numeric), or column name (String) within the retrieved recordset to extract data from.

Example:

```
Dim sSQL As String
Dim sCols As String
Dim sRows As String
Dim sPart As String
sSQL = "select * from ItemMaster"
DB.Execute(sSQL, sCols, sRows)
sPart = DB.Extract(sCols, sRows, 1, "PartNo")
```

## MakeList

This function executes a pass-through SQL ‘select’ statement against the database and converts the results into a scrolling list of items when used with App.ShowList.

Syntax: sMyList = DB.MakeList(sSQL, [bRtnAllCols], [bNormalize], [bScale, nMaxRows])

sMyList (String) is the list to be returned for display (i.e., set RSP = Value to display the list on the Client device.)

sSQL (String) is the SQL ‘SELECT’ statement to be sent to the database.

bRtnAllCols (Boolean) Optional – when set to True will return all the columns as the potential key, not just the first column. Default is False.

bNormalize (Boolean) Optional – when set to True will trim the spaces from the data so that it will display consistently. Default is False.

bScale (Boolean) Optional – formats the numeric values in the specified column if the database does not store decimals. This option will position a decimal at a position from the right side. A comma will be used for large numbers. This field is locale specific and will use the appropriate characters for each region. Default is True.

nMaxRows (Long) Optional – limits how many rows will be allowed in the list. Default is 500.

Examples: Inside the OnEnter event where Rsp is declared as a String:

```
Dim sSQL As String
Dim sMyList As String
sSQL = "select PartNo from Inventory"
sMyList = DB.MakeList(sSQL, True, True, True, 1000)
Rsp = App.ShowList(sMyList)
```

Or

```
Rsp = App.ShowList(DB.MakeList(sSQL, True, True))
```

## OpenResultset

This function will execute a pass-through SQL statement on the host through the connection maintained by the server. Any results from the statement will be returned in a RDO Resultset object. (Note: this item defaults to a static view of the database and cannot be updated. Also, under the configuration of the database, the ‘Connect Using...’ setting must match the declaration of the recordset; ADO versus RDO.)

Syntax: oRs = DB.OpenResultset(sSQL, [vCursorType], [vLockType])

oRs	(rdoResultset Object or adoRecordset Object) is a snapshot type resultset generated from an SQL statement.
sSQL	(String) is the SQL statement to be sent to the database. This is a pass through statement; its syntax must be understood by the database, as no pre-processing will occur.
vCursorType	(Variant) Optional – indicates the type of cursor used by the recordset object.
vLockType	(Variant) Optional – indicates the type of lock placed on records during editing.

#### ADO Cursor Types

- 0 Provides a static copy of the records (you can't see additions, changes or deletions by other users). You can only move forward through the recordset. Forward-only is the ADO default cursor type.
- 1 Existing records at time of creation are updateable. You can't see additions or deletions. All types of movement are enabled.
- 2 Dynamic requires more overhead, because updates are immediate and all types of movement are enabled. The dynamic cursor isn't currently supported by the Microsoft Jet OLE DB Provider, and therefore defaults to a keyset cursor when adOpenDynamic is applied to a Jet database.
- 3 Provides a static copy of the records (you can't see additions, changes or deletions by other users), but all types of movement are enabled.

#### ADO Lock Types

- 1 This value indicates read-only records where the data cannot be altered.
- 2 This value indicates pessimistic locking, record by record. The provider does what is necessary to ensure successful editing of the records, usually by locking records at the data source immediately after editing.  
This lock type is supported by the Microsoft® OLE DB Provider for AS/400 and VSAM and the Microsoft OLE DB Provider for DB2. However, the OLE DB Provider for AS/400 and VSAM internally maps this lock type to adLockBatchOptimistic.
- 3 This value indicates optimistic locking, record by record. The provider uses optimistic locking, locking records only when the Update method is called.

- This lock type is not supported by the OLE DB Provider for DB2.
- 4 This value indicates optimistic batch updates and is required for batch update mode.  
This option is not supported by the OLE DB Provider for DB2.

Example: Using rdoResultset, the RDO language extensions must be enabled.

```
Dim oRs As rdoResultset
Dim sSQL As String
Dim sDesc As Variant
sSQL = "select * from ItemMaster where PartNo =
'100620'"
Set oRs = DB.OpenResultset(sSQL)
sDesc = oRs!Description
```

## RedirectDataSource

This command will disable the server's auto processing of SQL statements to determine their correct source and will instead route all SQL traffic that is intended for the specified ODBC data source to always go to the other specified data source. To clear a redirection simply set the source and destination to the same data connection.

Syntax: bValue = DB.RedirectDataSource(vFromSource, vToSource)  
bValue (Boolean) indicates success or failure  
vFromSource (Variant) is the name or number of the data source to redirect.  
vToSource. (Variant) is the name or number of the data source to receive the other database's SQL statements.

Example:

```
Dim bValue as Boolean
bValue = DB.RedirectDataSource("CAPlant", "NVPlant")
```

## RollbackTrans

This function cancels any changes made during the current transaction and ends the transaction.

Syntax: [bValue =] DB.RollbackTrans(vSource)  
bValue (Boolean) Optional – indicates success or failure  
vSource (Variant) data source name (DSN) or the data source number

Examples:

```
DB.RollbackTrans ("RFSample")
```

```
DB.RollbackTrans(1)
```

## SaveBitmap

This command will save the byte array of a bitmap picture directly to a database. The record in the database must already exist.

Syntax: [bValue =] DB.SaveBitmap(sTable, sField, bData, [vWhere])

bValue      (Boolean) Optional – indicates success or failure

sTable      (String) is the name of the table in the database

sField      (String) is the field name within the table

bData      (Byte) is the byte array containing the image

vWhere      (Variant) Optional – a SQL parameter that updates or inserts the image to a specific record

Example:

```
Dim bVal as Boolean
```

```
Dim bImage() as Byte
```

```
Dim nSize As Long
```

```
Open "C:\MyPic.bmp" For Binary Access Read As #1
```

```
    nSize = LOF(1)
```

```
    ReDim bImage(nSize - 1)
```

```
    Get #1, , bImage
```

```
Close #1
```

```
'
```

```
bVal = DB.SaveBitmap("Images", "Image", bImage,  
"PartNo='100620'")
```

## UseDataSource

This command will disable the server's auto processing of SQL statements to determine their correct source and will instead route all SQL traffic to the specified data source. Specifying zero will re-enable the server's auto processing and let it determine which data connection should receive the SQL statement based on which connection was used to download the table structures.

Syntax: [bValue =] DB.UseDataSource(vSource)

bValue      (Boolean) Optional – indicates success or failure of the language extension

vSource      (Variant) is the name or number of the data source to which all future SQL traffic will be routed.

Examples:

```
DB.UseDataSource ("RFSample")
```

```
DB.UseDataSource (0)
```

# Mobile Device Extensions

Commands are supported that are used specifically on a mobile installation. These commands provide a method of synchronizing the server, sending and receiving data, queuing transactions on the host, etc. For the queuing commands, the server's Transaction Manager and the client's Transaction Manager must be enabled.

The mobile environment also supports the ability to load and execute COM objects on the Windows CE device. COM objects are loaded and run by executing the language extensions ceObject.Create, ceObject.Execute, and ceObject.Release. (ceObject is an object you must declare as a variable first and then use that variable.) The objects must be derived from the IDispatch interface and must be compatible with VBA scripting environments.

## ClickAndSkipPrompts

This method will turn on or off the user's ability to click in to prompts in a random sequence. The line "Device.ClickAndSkipPrompts(False)" will allow the user to click in the very next prompt or any prompt that already contains data. All other prompts will not receive focus if they are clicked. Using Device.ClickAndSkipPrompts(True) will set the client back to its default behavior.

Syntax: Device.ClickAndSkipPrompts(bEnabled)  
bEnabled (Boolean) True or False turns this feature on or off.

Example:

```
Device.ClickAndSkipPrompts (True)
```

## ClickCoordinates

This command will return the last set of X, Y coordinates relative to the control that was clicked. For example, after clicking on an image control, a call to this command will return X, Y coordinates with 0, 0 being the upper left corner of the image control. If there is a label or caption for the control, its area is included as possible click space. Note: this will not work on some controls that require clicks such as a signature capture box.

Syntax: Device.ClickCoordinates(vX, vY)  
vX (Variant) the horizontal axis pixel in the last click event  
vY (Variant) the vertical axis pixel in the last click event

Example:

```
Dim vX as Variant  
Dim vY as Variant  
Device.ClickCoordinates(vX, vY)
```

## EnableGPS

This property will activate or deactivate the GPS receiver in the mobile device. Use Device.GetGPSInfo to retrieve the real-time GPS data.

Syntax: Device.EnableGPS(bValue)

bValue (Boolean) set to True or False to activate or deactivate the GPS receiver

Example:

```
Device.EnableGPS (True)
```

## ForceLocal

This property will tell the Mobile Client in a Roaming state that it should only look to either the server or local database for all data retrieval and updates. For example, if Mobile Client in a Roaming state was connected to the server and the ForceLocal was set to False, when the user executes a TM.QueueMacro it would actually be queued on the server not the local device. DB.Execute would look to the server automatically to retrieve the latest data rather than the locally stored data on the device.

Syntax: Device.ForceLocal = bValue

bValue (Boolean) set to True or False to force data access to either the server or local database

Example:

```
Device.ForceLocal = True
```

## GetGPSInfo

This command will retrieve GPS data from an attached GPS receiver on the mobile device. If no receiver is available or no signal is received the output will be zeros. This command would need to be called repeatedly to update the screen if the device is in motion. Use the Form\_OnTimer event to continuously populate variables.

Note: The operating system GPS APIs must exist on the device. To configure the GPS settings on the mobile device locate the GPS configuration, set the Program COM port to 'none' and set the Hardware COM port to the proper number based on the manufacturer's documentation.

Syntax: [bOK =] Device.GetGPSInfo([vLongitude], [vLatitude],  
[vHeading], [vAltitude], [vSpeed])

OK (Boolean) Optional – the success or failure of the command.  
Longitude (Variant) Optional – returns the longitude as a float value  
Latitude (Variant) Optional – returns the latitude as a float value  
Heading (Variant) Optional – returns the heading as a value (0-360)  
Altitude (Variant) Optional – returns the altitude in meters  
Speed (Variant) Optional – returns the speed in knots (nautical miles)

**Examples:**

```
Dim bOK As Boolean
Dim vLong As Variant
Dim vLat As Variant
Dim vHead As Variant
Dim vAlt As Variant
Dim vSpeed As Variant

bOK = Device.GetGPSInfo (vLong, vLat, vHead, vAlt,
vSpeed)
bOK = Device.GetGPSInfo (, , , vAlt)
```

For converting meters to US feet use the following conversion:

```
vAlt = vAlt * 3.28083989501312
```

For converting knots to US MPH use:

```
vSpeed = vSpeed * 1.15077945
```

For converting knots to KPH use:

```
vSpeed = vSpeed * 1.852
```

For converting the heading into a direction use:

```
Select Case vHead
Case Is < 22.5: sText = "N"
Case Is < 67.5: sText = "NE"
Case Is < 112.5: sText = "E"
Case Is < 157.5: sText = "SE"
Case Is < 202.5: sText = "S"
Case Is < 247.5: sText = "SW"
Case Is < 292.5: sText = "W"
Case Is < 337.5: sText = "NW"
Case Else: sText = "N"
End Select
```

## GoOffline

This command returns a True / False regarding the attempt to close the socket connecting the mobile client to the server. This will make the device go from THIN client mode to a MOBILE disconnected state.

Syntax: [bOK =] Device.GoOffline([vUser], [vMenu], [vForm])

bOK	(Boolean) Optional – returns a True if the mobile device successfully closes the socket to the server. Since the thin client is shut down with this command, evaluating the response variable will never be reached unless there is a failure.
vUser	(Variant) Optional – If a user is specified then the device will set the current user and load the menu associated with the user bypassing the login screen.
vMenu	(Variant) Optional – Specifying a specific menu requires that the optional User parameter be filled in. The User parameter will still load the default menu for that user but then the Menu parameter will load, in addition. This means that if the user backs out of the specified menu the server will display the user's default menu.
vForm	(Variant) Optional – If the form is specified then the form will be loaded after the specified menu or the default menu. The Form depends on the UserID being filled in.

Examples:

```
Dim bOK as Boolean  
bOK = Device.GoOffline  
bOK = Device.GoOffline("Sam")  
bOK = Device.GoOffline("Sam", , "IMASTER")  
bOK = Device.GoOffline("Sam", "BasicApps", "IMASTER")
```

## GoOnline

This command returns a True / False regarding the status of opening a socket connection with the server. This will make a disconnected MOBILE client an online THIN client to the server. If the user is going online for the first time the server will present the login screen. If the user goes online with the configured client inactivity timeout value they would see the session as they left it when going mobile. This command will not work for Mobile clients with a startup mode of Disconnected.

Syntax: [bOK =] Device.GoOnline([vServer], [nPort])

bOK	(Boolean) Optional – return a True if the mobile device is successfully connected to the server. Since the mobile client is shut down with this command, evaluating the response variable will never be reached unless there is a failure.
-----	--

- vServer (Variant) Optional – parameter to specify which server name or IP address should be used. If this is not specified, the registry settings will be used.
- nPort (Long) Optional – parameter to specify which server port number should be used. If this is not specified, the registry settings will be used.

Example:

```
Dim bOK as Boolean  
bOK = Device.GoOnline("192.168.123.45", 21098)
```

## IsOffline

This command returns a True / False regarding the status of an open socket connection with the server. This command is typically used to evaluate which state the client is in and then set variables that would vary between the server and the mobile device such as file path information.

Syntax: bOK = Device.IsOffline

bOK (Boolean) return a True if the mobile device is currently not connected to the server.

Example:

```
Dim bOK as Boolean  
bOK = Device.IsOffline
```

## IsOnline

This command returns a True / False regarding the status of an open socket connection with the server. This command is typically used to evaluate which state the client is in and then set variables that would vary between the server and the mobile device such as file path information.

Syntax: bOK = Device.IsOnline

bOK (Boolean) return a True if the mobile device is connected to the server.

Example:

```
Dim bOK as Boolean  
bOK = Device.IsOnline
```

## Platform

This command returns an enumeration indicating the platform of the connected device.

Syntax: enValue = Device.Platform

enValue (Enumeration)

0 = Device_None	(Application Testing)
1 = Device_WinCE	(CE / Mobile Device)
2 = Device/Desktop	(Windows Desktop Client)
3 = Device_Android	(Android Device)
4 = Device_iOS	(Apple Device)
5 = Device_Vocollect	(Vocollect Talkman)
6 = Device_TN	(Standard Telnet Client)
7 = Device_SOA	(SOA Service Connection)

Example:

```
Dim vValue as Variant  
vValue = Device.Platform
```

## PlaySound

This command plays any mobile supported sound file by specifying the path to the file.

Syntax: Device.PlaySound(sPath)

sPath (String) the full path to the sound file to be played.

Example:

```
Device.PlaySound ("\Program Files\ERROR.WAV")
```

## PrinterOff

This command is obsolete and has been replaced with Device.SendCommPort. It will turn off transparent print mode and redirect all text received by the Client device back to the standard display. Note: Device.PrinterOff will not function in the GUI mode, Consider using Device.SendCommPort instead.

Syntax: Device.PrinterOff

Example: See *Device.Send*

## PrinterOn

This command is obsolete and has been replaced with Device.SendCommPort. It will turn on transparent print mode and redirect all text received by the client device to the attached serial printer. Note: Device.PrinterOn will not function in the GUI mode. Consider using Device.SendCommPort instead.

Syntax: Device.PrinterOn

Example: See *Device.Send*

## ReadFile

This command will read data from a file on the mobile device. String data or binary data can be read.

Syntax: [bOK =] Device.ReadFile(sFileName, vData)

bOK           (Boolean) Optional – the success or failure of the command.

sFileName     (String) specifies where on the mobile device the file should be found

vData         (Variant) contains the data read from the file. String or binary data is allowed.

Example:

```
Dim bOK As Boolean  
Dim vData As Variant  
Dim sFile As String  
  
sFile = "\Program Files\MyFile.txt"  
bOK = Device.ReadFile(sFile, vData)
```

## Send

This command is obsolete and has been replaced with *Device.SendCommPort*. It will send raw, unformatted text to the Client device; it is typically used to send text to an attached barcode printer or other auxiliary device. Used with *Device.PrinterOn* and *Device.PrinterOff* only.

Syntax: *Device.Send(vPacket)*

vPacket     (Variant) is the text stream that is to be sent to the attached barcode printer.

Example:

```
Device.PrinterOn  
Device.Send("Part: 100620, Desc: Office Chair")  
Device.PrinterOff
```

## SendCommPort

Sends data to the device serial port. This command replaces the commands *Device.PrinterOn* and *Device.PrinterOff* that are still supported but are now obsolete. The server will automatically turn transparent print mode on or off as required.

Syntax: [bOK] = *Device.SendCommPort(vPacket, iPort)*

bOK         (Boolean) Optional return value showing success or failure

vPacket (Variant) is the text stream to be sent to the serial port.  
iPort (Integer) when set to a number other than zero it will route the data to this COM port, if already configured. When set to zero, the default, it will send to the first COM port configured with the Device.SetCommPort.

Example: Sending a Tab character to a device that expects this format.

```
Device.SendCommPort ("&T", 3)
```

## **SetCameraOption**

This command changes some default settings for supported cameras. Since different cameras may use different settings, values for some IDs cannot be known beforehand.

Syntax: Device.SetCameraOption(enOption, vValue)

enOption (enCameraOptions) this ID contains either ImageBrightness,  
ImageContrast, or ImageFlash

vValue (Variant) the camera's expected value for the named ID

Example:

```
Device.SetCameraOption(ImageFlash, False)
```

## **SetCommMode**

This command changes the phonebook option located in the mobile configuration application. The option to connect using WiFi or GPRS can be set and then upon the next connection attempt the new method will be used.

Syntax: Device.SetCommMode(enMode)

enMode (enConnectMode) can be 1 of these 3 options:

ConnIsGPRS

ConnIsUndefined

ConnIsWiFi

Example:

```
Device.SetCommMode(ConnIsGPRS)
```

## **SetCommPort**

This command will enable or disable the serial port on a 'CE-based' data device.

Syntax:      Device.SetCommPort(bEnabled, iPort, nBaudRate,  
iByteSize, enStopBits, enParity, enFlowControl,

	nPollingInterval,	[bIsUnicode],	[bAddPrefix],
[nPacketSize])			
bEnabled:	(Boolean) True / False, enable or disable the serial port		
iPort	(Integer) 1,2,3,etc - the port number to activate		
nBaudRate	(Long) 9600, 19200, etc.		
iByteSize	(Integer) 7, 8, etc.		
enStopBits	(enStopBits) Select the appropriate stopbits from the drop-down list (1,1.5 or 2)		
enParity	(enParity) Select the appropriate Parity from the drop-down list (none, even, odd, etc.)		
enFlowControl	(enFlowControl) Select the appropriate FlowControl from the drop-down list (CTS, DSR, XONXOFF, etc.)		
nPollingInterval	(Long) The device will poll the serial port and look for data coming in using this timing interval in milliseconds.		
bIsUnicode	(Boolean) Optional – alerts the server that the attached COM device sends and receives in UNICODE. The default is False.		
bAddPrefix	(Boolean) Optional – when set to true, all data returned in the scan event coming from a COM port will have the COM port as a prefix		
nPacketSize	(Long) Optional – when set to a positive number, the data will only be returned in strings of this size. RFgen will cache the data until the packet size has been read.		

Example:

```
Device.SetCommPort(True, 1, 9600, 8, 1, NONE, NONE,
1000, False, True, 8)
```

## TakePicture

This command will retrieve an image from supported cameras and will fill a byte array with a BMP type image. (See also DB.SaveBitmap)

Syntax: [bOK =] Device.TakePicture(bImage)

bOK      (Boolean) Optional – the success or failure of the command.

bImage    (Byte) stores the byte array of the BMP image

Example:

```
Dim bOK As Boolean
Dim sTableName As String
Dim sField As String
Dim sWhere As String
Dim bImage() As Byte

sTableName = "Images"
sField = "Image"
```

```
sWhere = "PartNo = '100620'"  
bOK = Device.TakePicture(bImage)  
imgPic.Bitmap = bImage  
DB.SaveBitmap(sTableName, sField, bImage, sWhere)
```

## WriteFile

This command will write data to a file on the mobile device. String data or binary data can be written.

Syntax: [bOK =] Device.WriteFile(sPath, vData, bOverwrite)

bOK           (Boolean) Optional – the success or failure of the command.

sPath          (String) specifies where on the mobile device the file should be placed

vData         (Variant) contains the data to be written to the file. If it is string data it will be saved in Unicode format otherwise it will be left as is. A byte array is also allowed.

bOverwrite    (Boolean) set to True, this command will overwrite an existing file, False will return a failure because the file already existed

Example:

```
Dim bOK As Boolean  
Dim sData As String  
Dim sPath As String  
  
sPath = "\Program Files\MyFile.txt"  
sData = "Sample Text"  
bOK = Device.WriteFile(sPath, sData, True)
```

## DeviceObject

This object is declared as a variable type and used to execute COM objects from the CE device. It is not part of the Device object directly but is a subset of CE functionality specifically for calling COM objects.

The following examples would all start with:

```
Dim [WithEvents] oMyObj As DeviceObject
```

If the object supports events include the WithEvents statement in the Dim statement. If WithEvents is declared as part of the Dim statement, this event would also be available from the script window's drop-down:

```
Private Sub oMyObj_OnEvent(ByVal EventName As String,  
ByRef rsData As DataRecord)
```

```
End Sub
```

## Create

This command loads a COM object stored on the CE device. Calling Create multiple times on the same program id will increment the reference count. Create will only register for events, if specified, on the first call to Create.

The SysErr.Number object should be inspected to determine if Create was successful.

Syntax: [bOK =] oMyObj.Create

bOK (Boolean) Optional – the success or failure of the command.

Example: (*See Execute*)

## Execute

This command executes the COM object stored on the CE device. Execute will first determine if the COM object has been loaded. If the object has not been loaded then Execute will attempt to load the object without events. If the object could be loaded successfully, the method executes. The object will remain loaded after this method completes execution.

The SysErr.Number object should be inspected to determine if Execute was successful.

Syntax: [bOK =] oMyObj.Execute(sMethod, [vParams])

bOK (Boolean) Optional – the success or failure of the command.

sMethod (String) Method is a text string that represents the method that you wish to execute.

vParams (Variant) Optional – parameter array where you specify the parameters to the method. The parameters must be specified in the appropriate order that the method expects.

Example:

```
Dim bOK As Boolean
Dim oMyObj As DeviceObject
Dim vErr As Variant
Dim sParam1 As String
Dim sParam2 As String

Set oMyObj = New DeviceObject
oMyObj.Name = "DeviceObjectTest.ioInterface"
oMyObj.Create
sParam1 = "Value1"
sParam2 = "Value2"
```

```
bOK = oMyObj.Execute("Method1", sParam1, sParam2)
vErr = oMyObj.LastError
If vErr <> "" Then
    App MsgBox "COM failure: " & CStr(vErr)
Else
    App MsgBox "COM object returned: " &
oMyObj.ReturnValue
End If
oMyObj.Release
```

## LastError

This property returns the last error generated from the Execute method.

Syntax: vValue = oMyObj.LastError

vValue (Variant) is the error number returned from the COM object

Example: (See *Execute*)

## Name

This property sets the program ID of the object that you wish to load. The object must have been successfully registered on the device for the Create function to succeed.

Syntax: oMyObj.Name = sProgID

Alternate: sProgID = oMyObj.Name

sProgID (String) a string indicating the program ID of the object that you wish to execute.

Example: (See *Execute*)

## Release

This command releases the COM object. Calling Release multiple times on the same program ID will decrement the reference count. Release will free the object when the reference count reaches 0.

The SysErr.Number object should be inspected to determine if Release was successful.

Syntax: [bOK =] oMyObj.Release

bOK (Boolean) Optional – the success or failure of the command.

Example: (See *Execute*)

## ReturnValue

This property returns any value being passed back from the COM object

Syntax: [vValue =] oMyObj.ReturnValue  
vValue (Variant) Optional – the value returned by the COM object.

Example: (See *Execute*)

## Transaction Management Extensions

Commands relating to Transaction Management capabilities and queuing are called from the TM object.

### AbortTrans

This function can be used inside a Transaction macro to halt processing of the transaction if conditions warrant it. This command will notify the Transaction Manager that the current macro processing was aborted for a reason other than failure. The Transaction Manager will then keep this transaction at the top of the queue and try again on the next cycle. For example, if an SM command comes back with a failed status or the BeginTrans command fails, TM.AbortTrans can be executed to give the transaction macro's Boolean value a False value. In the application's script, you will know if the called macro was successful.

Syntax: TM.AbortTrans

Example:

`TM.AbortTrans`

### CheckStatus

This method looks up the status of a queued or processed transaction and returns what happened to that transaction. The return options are that the macro cannot be found, it failed, it succeeded, it is still queued and the macro couldn't be executed.

Syntax: enResult = TM.CheckStatus(sQueue, nSeqNo)

enResult (enMacroResults) the enumeration describing what was found. Possible values are MacroFailed, MacroNotFound, MacroNotProcessed, MacroQueued and MacroSucceeded.

sQueue (String) the name of the queue to be searched

nSeqNo (Long) the sequence number to look for

Example:

```
Dim enValue As enMacroResults
enValue = TM.CheckStatus("RFQueue", 100)
```

### GetItems

This method loads a set of macros to be evaluated from the Transaction Management tables. Selecting the category, date and user, this command will return a DataRecord containing the macros and their data. If a list of macros with a more complex selection criteria is desired use TM.GetItemsEx.

Syntax: [bOK = ] TM.GetItems(enStatus, vTranDate, sUserID, oTran)

bOK	(Boolean) Optional – True or False for the success of the command
enStatus	(enItemStatus) There are 4 options for the ItemStatus parameter: <b>tmAllItems</b> all macros in all queues, in the queue, failed, or completed <b>tmCompleted</b> all macros in all queues that were completed successfully <b>tmFailed</b> all macros in all queues that were completed unsuccessfully <b>tmInProcess</b> all macros in all queues that are not yet processed
vTranDate	(Variant) specifies a single day to be retrieved
sUserID	(String) specifies the user that performed the transaction
oTran	(DataRecord) the variable declared as a DataRecord that contains the list of macros and their data. For more information on how the DataRecord object is used see the DataRecord section of the manual.

#### Example:

```
Dim bOK As Boolean
Dim oList As DataRecord
bOK = TM.GetItems(tmFailed, Date, "SAM", oList)
```

This command will return all the failed macros on the current day that was performed by the user Sam.

### GetItemsEx

This method loads a set of macros to be evaluated from the Transaction Management tables. Specify a SQL statement that will retrieve the desired recordset and take advantage of the complexity allowed by ODBC. Knowledge of the table and field names is required.

Syntax: [bOK = ] TM.GetItemsEx(sSQL, oTran)

bOK	(Boolean) Optional – True or False for the success of the command
sSQL	(String) specifies the SQL statement for retrieving a list of macros

`oTran` (`DataRecord`) the variable declared as a `DataRecord` that contains the list of macros and their data. For more information on how the `DataRecord` object is used see the `DataRecord` section of the manual.

The table names required depend on the name of the queue configured in the Transaction Management setup. For the `RFQueue` the tables created are:

`RFQueue` – used to store the `InProcess` items

`RFQueue_Completed` – used to store successfully completed items

`RFQueue_Failed` – used to store failed transaction macros

Substitute the `RFQueue` name for alternate queues that may have been defined. The fields that can / should be referenced through SQL within the tables are shown below.

**RFQueue** table:

<b>Field Name</b>	<b>Data Type</b>	<b>Description</b>
SeqNo	Number	the sequence number
TranDate	Number	when the macro was queued (20151231)
TranTime	Number	when the macro was queued (235959)
Source	Text	linked source for the macro if it exists
Name	Text	name of the macro
Record	Text	contains the passed values to the macro
FormId	Text	form that queued the macro
UserId	Text	user that queued the macro
DeviceNo	Number	device number assigned to the client
IPAddress	Text	IP address of the client device
DevGuid	Text	assigned GUID of the client device
ExecDate	Number	date the macro executed (20151231)
ExecTime	Number	time the macro executed (235959)
Status	Number	integer status of the macro

The `RFQueue_Completed` table and `RFQueue_Failed` table have the same design as `RFQueue`.

**Example:**

```
Dim oList As DataRecord
TM.GetItemsEx("Select * from RFQueue where UserId =
'SAM'", oList)
```

## MacroName

This returns the name of the macro currently in process by the Transaction Manager. This command is only available while the

transaction macro is being executed so it must be used either in the macro itself or in any global function calls that may be used to process generic macros such as a logging feature.

Syntax: sValue = TM.MacroName

sValue      (String) the name of the macro being executed

Example:

```
Dim sName As String  
sName = TM.MacroName
```

## MoveQueue

This function will take a queue from one database and guarantee its delivery to another database. If another instance of the server is monitoring the second database, that instance will become responsible for executing the queued transactions. The Transaction Management database connection must be capable of seeing both databases.

Syntax: [bValue =] TM.MoveQueue(sFromQueue, sToQueue)

bValue      (Boolean) Optional – the success or failure to move the queue from one database to another.

sFromQueue (String) the source queue to be moved

sToQueue    (String) the destination queue to receive the transactions

Example:

```
Dim bValue As Boolean  
bValue = TM.MoveQueue("RFQueue", "HostQueue")  
bValue = TM.MoveQueue("RFQueue", "AltDB.HostQueue")
```

## QueueMacro

This function is used to queue Data Transaction macros and pass any required parameters. It returns the sequence number of the macro after it has been successfully queued. These macros can be created on the Transactions tree.

Syntax: [nSeq =] TM.QueueMacro(sMacroName, [vParams])

nSeq      (Long) Optional – the sequence number of the macro after it has been successfully queued.

sMacroName (String) is the name of the macro to be called.

vParams    (Variant) Optional – A series of passing parameters as required by the selected macro. Note: these parameters are the fields you defined when you wrote the macro.

Example:

```
Dim nSeq As Long
```

```
nSeq = TM.QueueMacro("ChangeUser", "Sam", "1234")
```

## QueueName

This function will return the name of the queue currently being processed if this command is executed from within the macro itself.

Syntax: sValue = TM.QueueName

sValue (String) is the name of the queue

Example:

```
Dim sValue As String  
sValue = TM.QueueName
```

## SeqNo

This function will return the sequence number of the transaction currently being processed if this command is executed from within the macro itself.

Syntax: nValue = TM.SeqNo

nValue (Long) is the sequence number

Example:

```
Dim nValue As Long  
nValue = TM.SeqNo
```

# Enterprise Resource Planning Extensions

Commands relating to ERP capabilities are called from the ERP object.

## BeginTrans

This command is used to retrieve an ERP connection from the managed pool and keep it for an unspecified amount of time. It would typically be used to execute a sequence of ERP commands against a pooled connection. Note: If the connection is not pooled, or will call only a single business function, this command is not needed. The first ERP data connection is the default Source value.

Syntax: [bValue =] ERP.BeginTrans([vSource])

bValue (Boolean) Optional – A True/False notification that the command processed successfully

vSource (Variant) Optional – data source name (DSN) or the data source number

Examples:

```
Dim bValue As Boolean  
bValue = ERP.BeginTrans(1)  
ERP.BeginTrans("SAP")
```

## CommitTrans

This command is used to release an ERP connection back to the managed pool once the process is finished with it. Note: You must always use this function paired with ERP.BeginTrans, otherwise you will deplete the pool of connections and prevent other users from having ERP access. The first ERP data connection is the default Source value.

For an SAP system, this command will also execute BAPI\_TRANSACTION\_COMMIT.

Syntax: [bValue =] ERP.CommitTrans([vSource])

bValue (Boolean) Optional – A True/False notification that the command processed successfully

vSource (Variant) Optional – data source name (DSN) or the data source number

Example:

```
Dim bValue As Boolean  
bValue = ERP.CommitTrans(1)  
ERP.CommitTrans("SAP")
```

## LogOff

This function is used to logoff a non-pooled ERP connection. Note: this function does not need to be called by the user. The server will call it automatically when shutting down the session. The first ERP data connection is the default Source value.

Syntax: [bValue =] ERP.LogOff (vSource)

bValue (Boolean) Optional – A True/False notification that the command processed successfully

vSource (Variant) data source name (DSN) or the data source number

Example:

```
Dim bValue As Boolean  
bValue = ERP.LogOff(1)  
ERP.LogOff("SAP")
```

## LogOn

This is used to logon the ERP connection and specify a user / password sequence for a non-pooled ERP connection (optional).

Note: the user does not always require this function as the server calls it automatically when a session starts.

Syntax: [bValue =] ERP.LogOn (vSource, [vUserId], [vUserPwd], [vOptions])

bValue (Boolean) Optional – A True/False notification that the command processed successfully  
vSource (Variant) data source name (DSN) or the data source number  
vUserId (Variant) Optional – The login name to be used to connect to the ERP system  
vUserPwd (Variant) Optional – The login password to be used to connect to the ERP system  
vOptions (Variant) Optional – SAP only additional parameters that can be added to the logon string. Separate multiple parameters with the pipe ( | ) symbol.

CLIENT	- SAP Client
LANG	- Logon language
SYNSNR	- SAP System number
ASHOST	- SAP application server
MSHOST	- SAP message server
GWHOST	- Gateway host
GWSERV	- Gateway service
R3NAME	- R/3 name
GROUP	- Group of SAP application servers
TPHOST	- Host of the external server program
TYPE	- Type of remote host (2 = R/2, 3 = R/3, E = External)
TRACE	- Enable RFC trace (1=on, 0 = off)
CODEPAGE	- Initial code page in SAP notation
LCHECK	- Enable logon check at open time (1=on, 0 = off)
QOSDISABLE	- Disable Quality of Service check (1=disable, 0 = enable)
GRT_DATA	- Additional data for GUI
USE_GUIHOST	- Redirect remote GUI to this Host
USE_GUISERV	- Redirect remote GUI to this Service
USE_GUIPROGID	- Server program id that will start the remote GUI
SNC_MODE	- Enable Secure Network Communications (1=on, 0 = off)
SNC_PARTNERNAME	- SNC partner (p:CN=R3, O=XYZ-INC, C=EN)
SNC_QOP	- SNC Level of security (1-9)
SNC_MYNAME	- SNC Name - overrides default SNC partner
SNC_LIB	- SNC service library path
DEST	- R/2 destination
SYSTEM	- Name of the system as used in the system landscape definition

LOGONMETHOD	- Authentication method used to log on to the destination (UIDPW, SAPLOGONTICKET, X509CERT)
-------------	---

### Examples:

```
Dim bValue As Boolean
bValue = ERP.LogOn("SAP", "User345", "Pass345")
bValue = ERP.LogOn(1, "User345", "Pass345",
"CLIENT=800|TYPE=3")
bValue = ERP.LogOn("JDE", "JDEUser", "JDEPass1")
```

## MakeList

This function executes a pass-through SQL 'select' statement against the ERP system's database and converts the results into a scrolling list. You may use App.ShowList to display the list to the user. The first ERP data connection is assumed as the source.

Syntax: sMyList = ERP.MakeList(sSQL, [bRtnAllCols], [bNormalize], [bScale, nMaxRows])

sMyList (String) is the list to be returned for display (i.e., set RSP = MyList to display the list on the Client device.)

sSQL (String) is the SQL 'SELECT' statement to be sent to the database.

bRtnAllCols (Boolean) Optional – when set to True will return all the columns as the potential key, not just the first column. Default is False.

bNormalize (Boolean) Optional – when set to True will trim the spaces from the data so that it will display consistently. Default is False.

bScale (Boolean) Optional – formats the numeric values in the specified column if the database does not store decimals. This option will position a decimal at a position from the right side. A comma will be used for large numbers. This field is locale specific and will use the appropriate characters for each region. Default is True.

nMaxRows (Long) Optional – limits how many rows will be allowed in the list. Default is 500.

### Example:

```
Dim sSQL As String
Dim sMyList As String
sSQL = "select PartNo from ItemMaster"
sMyList = ERP.MakeList(sSQL, True, True, True, 1000)
Rsp = App.ShowList(sMyList)
```

Or

```
Rsp = App.ShowList(ERP.MakeList(sSQL, True, True))
```

## **ReadData**

This command executes a read-only SQL statement against the ERP system. Note: In the case of SAP, the business function RFC\_READ\_TABLE is used and it is not an officially released BAPI and has significant limitations. Therefore it may not work with all versions of SAP. If this is the case, please contact support for a solution.

Syntax: [bValue =] ERP.ReadData(sSQL, sCols, sRows)

bValue (Boolean) Optional – A True/False notification that the command processed successfully

sSQL (String) the SQL statement to be executed on the ERP table

sCols (String) a variable containing the columns of the resulting data

sRows (String) a variable containing the rows of resulting data

SAP Limitations – since the RFC\_READ\_TABLE function is used, SQL statements must specify at least 1 field. Using an asterisk (\*) or a function will not be accepted. Examples are “select \*”, “select count(\*)”, select SUM(QTY) …”, etc. Only a comma-delimited list of field names is acceptable. Finally you may only specify 1 table in the SQL statement, no joins.

For example, if you would like to retrieve the Material numbers from a table, such as the Material Documents table within SAP, specify the following:

Example:

```
Dim bValue As Boolean  
bValue = ERP.ReadData("select MATNR from MSEG",  
sCols, sRows)
```

## **RollbackTrans**

This command is used to undo executed functions against an ERP connection assuming the ERP system is capable of undoing those functions. The first ERP data connection is the default Source value.

For an SAP system, this command will also execute BAPI\_TRANSACTION\_ROLLBACK.

Syntax: [bValue =] ERP.RollbackTrans ([vSource])

bValue (Boolean) Optional – A True/False notification that the command processed successfully

vSource (Variant) Optional – data source name (DSN) or the data source number

Example:

```
Dim bValue As Boolean  
bValue = ERP.RollbackTrans(1)  
ERP.RollbackTrans ("SAP")
```

## SetHardRelease

This function (exclusively for JDE connections) is to be called at the beginning of any transaction macro that will be affected by the resource leak in JD Edwards. It sets an internal flag that will release the data pointers when an ERP.CommitTrans or ERP.RollbackTrans is called. At this point the internal flag is toggled off.

Syntax: ERP.SetHardRelease

Example:

```
ERP.SetHardRelease
```

## SetSession

If there is more than one ERP connection and a transaction may need them independently but at the same time, ERP.SetSession will direct embedded business functions to the specified session, ignoring the defaults. The 'DataSource' method will overwrite this function if it is included in the VBA code.

Syntax: [bValue =] ERP.SetSession (vSource)

bValue (Boolean) Optional – A True/False notification that the command processed successfully

vSource (Variant) data source name (DSN) or the data source number. Setting this value to 0 will re-enable automatic determination of which data connection should be used based on which connection was used to download a particular business function being called.

Example:

```
Dim bValue As Boolean  
bValue = ERP.SetSession(1)  
ERP.SetSession ("SAP")
```

# Printer Extensions

Printer commands specifically interact with creating and delivering data to tethered or IP addressable printers.

## Activate

This command will redirect output to the specified Windows printer. Specifying the printer by name should be exactly the same as it appears in the Windows Control Panel under the Printers section. Make sure this is the first command used and reused after any EndDoc commands.

Syntax: Printer.Activate(vName)

vName (Variant) is the printer name in the 'Printers Window'. Click Start/ Settings/Printers. Note: You may use the printer number if desired.

Example:

```
Printer.Activate("HPDeskJet 697C")
```

## Copies

This property accesses the copy count parameter associated with a specific Windows printer. If it is not specified, the default is 1 copy.

Syntax: Printer.Copies = nValue

nValue (Long) is the number of copies to print.

Example:

```
Printer.Copies = 10
```

Note: there is a special condition with the Copies, Orientation and PrintQuality commands. Since the server manages the beginning and ending of a print job, the VBA scripts must not use one of these commands after the Printer.Print command. These 3 commands will start a new print job and all settings defined up to this point will be lost.

This is valid:

```
Printer.Copies = 10
Printer.Orientation = PrintHorizontal
Printer.Print "Sample Text"
Printer.EndDoc
```

This is not because the Orientation line will lose the 2 lines above it.

```
Printer.Copies = 10
Printer.Print "Sample Text"
Printer.Orientation = PrintHorizontal
Printer.EndDoc
```

## EndDoc

This command will close the print job on the selected Windows Printer and start the printing process.

Syntax: Printer.EndDoc

Example:

```
Printer.EndDoc
```

Note: All printer settings such as font, copies, orientation and others are reset to defaults after this command. They must be re-issued to take effect on the next print job.

## FontBold

This property accesses the font bold parameter associated with a specific Windows printer.

Syntax: Printer.FontBold = bValue

bValue (Boolean) determines whether text is printed using the bold font option.

Example:

```
Printer.FontBold = True
```

## FontItalic

This property accesses the font italic parameter associated with a specific Windows printer.

Syntax: Printer.FontItalic = bValue

bValue (Boolean) determines whether text is printed using the italic font option.

Example:

```
Printer.FontItalic = True
```

## FontName

This property accesses the font name parameter associated with a specific Windows printer.

Syntax: Printer.FontName = sValue

sValue (String) specifies the type of font to use.

Example:

```
Printer.FontName = "Arial"
```

## FontSize

This property accesses the font size parameter associated with a specific Windows printer.

Syntax: Printer.FontSize = nValue  
nValue (Long) is the font size to use.

Example:

```
Printer.FontSize = 12
```

## FontStrikeThru

This property accesses the font strike thru parameter associated with a specific Windows printer.

Syntax: Printer.FontStrikethru = bValue  
bValue (Boolean) determines whether text is printed using the strike thru font option.

Example:

```
Printer.FontStrikeThru = True
```

## FontUnderline

This property accesses the font underline parameter associated with a specific Windows printer.

Syntax: Printer.FontUnderline = bValue  
bValue (Boolean) determines whether text is printed using the underline font option.

Example:

```
Printer.FontUnderline = True
```

## GetName

This function returns the name for the specified Windows printer number. You may use this command within a loop to get a list of names for the user to pick from. See Printer.Activate for setting a printer.

Syntax: sName = Printer.GetName(nIndex)  
sName (String) is the Windows name of the requested printer.  
nIndex (Long) is the Windows printer number.

Example:

```
Dim sName As String
sName = Printer.GetName(1)
```

## NewPage

This command will send a page eject character to the selected Windows Printer.

Syntax: Printer.NewPage

Example:

```
Printer.NewPage
```

## Orientation

This function accesses the orientation parameter associated with a specific Windows printer.

Syntax: Printer.Orientation = enValue

enValue (enPrintOrientation) is the text orientation to use.

PrintHorizontal (Portrait)

PrintVertical (Landscape)

Example:

```
Printer.Orientation = PrintHorizontal
```

Note: there is a special condition with the Copies, Orientation and PrintQuality commands. Since the server manages the beginning and ending of a print job, the VBA scripts must not use one of these commands after the Printer.Print command. These 3 commands will start a new print job and all settings defined up to this point will be lost.

This is valid:

```
Printer.Copies = 10  
Printer.Orientation = PrintHorizontal  
Printer.Print "Sample Text"  
Printer.EndDoc
```

This is not because the Orientation line will lose the 2 lines above it.

```
Printer.Copies = 10  
Printer.Print "Sample Text"  
Printer.Orientation = PrintHorizontal  
Printer.EndDoc
```

## PageWidth

This function sets the printer print width to a specific value. The value is reset to 0 after an EndDoc command is issued.

Syntax: Printer.PageWidth = nValue

nValue (Long) is the printer width.

Examples:

```
Printer.PageWidth = 80  
Printer.PageWidth = 132
```

## Print

This command will print the text on the selected Windows printer. Each Print statement will be on a new line.

Syntax: Printer.Print(sText)

sText (String) is the text stream that is to be sent to the network printer.

Example:

```
Printer.Print("20lb Super Green Lawn Food")
```

## PrintQuality

This property accesses the print quality parameter associated with a specific Windows printer.

Syntax: Printer.PrintQuality = enValue

Alternate: enValue = Printer.PrintQuality

enValue (enPrintQuality) is the print quality desired.

DraftQuality

HighQuality

LowQuality

MediumQuality

Example:

```
Printer.PrintQuality = MediumQuality
```

Note: there is a special condition with the Copies, Orientation and PrintQuality commands. Since the server manages the beginning and ending of a print job, the VBA scripts must not use one of these commands after the Printer.Print command. These 3 commands will start a new print job and all settings defined up to this point will be lost.

This is valid:

```
Printer.Copies = 10  
Printer.Orientation = PrintHorizontal  
Printer.Print "Sample Text"  
Printer.EndDoc
```

This is not because the Orientation line will lose the 2 lines above it.

```
Printer.Copies = 10
Printer.Print "Sample Text"
Printer.Orientation = PrintHorizontal
Printer.EndDoc
```

## PrintRaw

This command will send a byte stream to the selected printer. A typical example would be sending escape sequences.

Syntax: Printer.PrintRaw(vData)

vData (Variant) is the byte stream or string that is to be sent to the printer. If the passed value is an actual byte array it will be sent to the printer unmodified. If it is not a byte array the server will convert it to an SBCS string and send that.

Example:

```
Dim sData As String
sData = "Print Me"
Printer.PrintRaw(sData)
```

Converts “Print Me” to Unicode and sends it

## Screen Mapping Extensions

Screen mapping commands deal specifically with placing and scraping text to and from a green screen / legacy system. The following details all of the Screen Mapping VBA Extensions.

### BeginTrans

This function is provided to allow a series of commands to be sent to a single host session when connection pooling is enabled. It basically gets a connection from the pool and holds it until SM.CommitTrans is called. If connection pooling is disabled, it has no effect. It returns True if a connection handle was available and removed from the pool.

Syntax: [bOK =] SM.BeginTrans([vScreen])

bOK (Boolean) Optional – Returns True if a connection handle is available for use.

vScreen (Variant) Optional – the name of a screen macro to be called so that upon connection the host will attempt to go to this screen. If the screen macro does not exist the pooled handle will be released and the function will return a False.

Example:

```
Dim bOK As Boolean  
bOK = SM.BeginTrans
```

## CommitTrans

This subroutine basically returns the current handle to the connection pool (if it was previously retrieved by using the SM.BeginTrans function) and makes it available to other client sessions. It returns True if the handle was successfully released back to the pool.

Syntax: [bOK =] SM.CommitTrans

bOK (Boolean) Optional – Returns True if the handle was released back to the connection pool.

Example:

```
Dim bOK As Boolean  
bOK = SM.CommitTrans
```

## Connected

This function is used to determine if the host is available for direct input or if transactions should be queued for later input. It also evaluates both the current state of the socket connection to the host and the state of any data sent but not yet received. If there is non-received data in the send buffer, this command will mark the connection as closed. It returns True if the host is currently available.

Syntax: blsConnected = SM.Connected

blsConnected (Boolean) Returns True if the host is currently available (e.g. the telnet connection is currently valid).

Example:

```
Dim bIsConnected As Boolean  
bIsConnected = SM.Connected
```

## CurScreen

This function returns the name of the current screen in the host session if it can be identified. The server identifies the screen by comparing the host screen to the Text Identifier grid and looks for an ID match. If a match is found, it returns the screen name otherwise this function returns an empty string.

Note: This function only works if the current screen has been defined as a macro that is linked to the main menu macro that is currently in use.

For example if MainMenu1 links to TransactionScreen1 and Screen2 and MainMenu1 is either configured in the screen mapping connector as the default menu or the program performed the SM.SetBase("MainMenu1") command, then only Screen1 and Screen2 can be identified. If Screen3 is linked to MainMenu2 and you are on Screen3 but MainMenu1 is your base menu, Screen3 cannot be identified. Use the SM.GetText command to verify your location.

Syntax: sScreenName = SM.CurScreen()

sScreenName (String) Returns the name of the current screen or an empty string if screen is unknown.

Example:

```
Dim sScreenName As String  
sScreenName = SM.CurScreen()
```

## FindText

This function is used to determine if a specified text string is currently displayed on the host screen. It can be used to look for the text in a specific screen location or to search the entire host screen for the text. It returns True if it is found. Optionally, it can also return the screen coordinates where the text was found.

Syntax: bFound = SM.FindText(sText, iStartCol, iStartRow, [vFoundCol], [vFoundRow])

bFound (Boolean) Returns True if the specified text string was found.

sText (String) Text string to be searched for.

iStartCol (Integer) the screen column to search for the text. Specify a column position of '-1' (minus 1) to search for the text in any screen column position.

iStartRow (Integer) the screen row to search for the text. Specify a row position of '-1' (minus 1) to search for the text in any screen row.

vFoundCol (Variant) Optional – the column position where the text was found

vFoundRow(Variant) Optional – the row position where the text was found.

Example:

```
Dim bFound As Boolean  
bFound = SM.FindText("UserID", 3, 5)
```

## GetArea

This command gets the text off the screen in any rectangular area.

Syntax: [bOK =] SM.GetArea(iStartCol, iStartRow, iEndCol, iEndRow,  
sAreaText, [bTrimData], [bAppendData])

bOK	(Boolean) Optional – Returns True / False depending on the success of the command
iStartCol	(Integer) start column position coordinate
iStartRow	(Integer) start row position coordinate
iEndCol	(Integer) end column position coordinate
iEndRow	(Integer) end row position coordinate
sAreaText	(String) variable containing the captured text
bTrimData	(Boolean) Optional – trims each line (row) of data. Default is False
bAppendData	(Boolean) Optional – appends the new block of data to the data variable rather than replaces it. Default is False

Example:

```
Dim sText As String
SM.GetArea(5, 5, 10, 10, sText, True)
SM.GetArea(5, 11, 10, 15, sText, True, True)
SM.GetArea(5, 16, 10, 20, sText, True, True)
```

## GetAttribute

This command returns an integer value describing the characteristics of an X,Y coordinate from a host screen. This command does not work with a VT host, only 5250 and 3270.

Syntax: iValue = SM.GetAttribute(iCol, iRow)

iValue (Integer) contains the value in the table below

- 1 - error was returned
- 0 - Normal
- 1 - Bold
- 2 - Reverse Video
- 4 - Underline
- 8 - Half
- 16 - Protected
- 32 - Hidden
- 64 - Graphic

iCol (Integer) is the column position in question.

iRow (Integer) is the row position in question.

Example:

```
Dim iValue As Integer
iValue = SM.GetAttribute(5,18)
```

## **GetBackColor**

This command returns an integer value representing the back color located at the specified coordinates. This command does not work with a VT host, only 5250 and 3270.

Syntax: iValue = SM.GetBackColor(iCol, iRow)

iValue (Integer) contains the integer value of the back color

- 1 - Error was returned
- 0 - Black
- 1 - Blue
- 2 - Green
- 3 - Cyan
- 4 - Red
- 5 - Purple
- 6 - Yellow
- 7 - Gray
- 8 - Light Blue
- 9 - Light Green
- 10 - Light Cyan
- 11 - Light Red
- 12 - Light Purple
- 13 - Light Yellow
- 14 - Light Gray
- 15 - White

iCol (Integer) Is the current column position in question.

iRow (Integer) Is the current row position in question.

Example:

```
Dim iValue As Integer  
iValue = SM.GetBackColor(3,22)
```

## **GetCursor**

This method is used to determine where the cursor is currently located on the host screen.

Syntax: SM.GetCursor(vCol, vRow)

vCol (Variant) Returns the current column position of the cursor.

vRow (Variant) Returns the current row position of the cursor.

Example:

```
Dim vCol As Variant  
Dim vRow As Variant  
SM.GetCursor(vCol, vRow)
```

## GetForeColor

This command returns an integer value representing the fore color located at the specified coordinates. This command does not work with a VT host, only 5250 and 3270.

Syntax: iValue = SM.GetForeColor(iCol, iRow)

iValue (Integer) contains the integer value of the fore color

- 1 - Error was returned
- 0 - Black
- 1 - Blue
- 2 - Green
- 3 - Cyan
- 4 - Red
- 5 - Purple
- 6 - Yellow
- 7 - Gray
- 8 - Light Blue
- 9 - Light Green
- 10 - Light Cyan
- 11 - Light Red
- 12 - Light Purple
- 13 - Light Yellow
- 14 - Light Gray
- 15 - White

iCol (Integer) Is the current column position in question.

iRow (Integer) Is the current row position in question.

Example:

```
Dim iValue As Integer  
iValue = SM.GetForeColor(3,22)
```

## GetText

This method is used to retrieve text from the host screen. Note: you can retrieve the entire screen buffer by specifying a starting position of 1,1 and a length of 2,000.

Syntax: SM.GetText(iCol, iRow, iLength, sScreenText, [bTrim])

iCol (Integer) The starting screen column position to get text.

iRow (Integer) The starting screen row position to get the text.

iLength (Integer) The number of screen columns to retrieve text from. Note: on DBCS systems, this may not be the number of characters returned as some characters require two screen columns.

sScreenText (String) The text being returned.

bTrim            (Boolean) Optional – if True, the string will be returned with all padding removed. Default is False.

Example:

```
Dim sText As String  
SM.GetText(3, 6, 10, sText, True)
```

## GoToScreen

This function attempts to navigate the host menu system to move to the requested application screen. It will return True if the desired screen is successfully displayed. Its method of operation (simplified) is as follows:

1. Test if the current screen is the requested screen.
2. Call the current screens “ReturnToMainMenu” method.
3. Call the requested screens “GoToScreen” method.

If the screen you are trying to get to is not part of the linked macros from the current main menu then the server will not navigate properly. The current main menu is defined in the screen mapping connector or by using the SM.SetBase command.

Syntax: [bOK =] SM.GoToScreen(sScreenName)

bOK            (Boolean) Optional – Returns True if the host session was successfully moved to the requested screen.

sScreenName    (String) The name of the screen to activate on the host session.

Example:

```
Dim bOK As Boolean  
bOK = SM.GoToScreen("UserLogin")
```

## IsScreen

This function returns a True or False by comparing the specified screen and optionally the page number against the host screen's current page.

Syntax: [bIsScreen =] SM.IsScreen(sScreenName)

bIsScreen        (Boolean) Optional – Returns True if the requested screen is the current active screen

sScreenName    (String) The name of the screen to be evaluated.

Example:

```
Dim bIsScreen As Boolean  
bIsScreen = SM.IsScreen("invTrans")
```

## LogOff

This function is used internally to logoff the session just prior to termination of the process. It works by sending the user to the base screen and executes the ‘LogOff’ macro. It returns True if the session was logged off properly. Note: this function is not required to be called by the user as the server calls it automatically when the user disconnects.

Syntax: [bOK =] SM.LogOff  
bOK      (Boolean) Optional – Returns True if the session was successfully logged off.

Example:

```
Dim bSuccess As Boolean  
bSuccess = SM.LogOff
```

## LogOn

This function is used to log in to the host system. It returns True if the session was logged in properly. The host system must already be logged off or you must use the SM.LogOff command. A user and password may be specified but the host does not use this information. This is for validation later in the programming. Note: this function is not always required to be called by the user as the server calls it automatically.

Syntax: [bOK =] SM.LogOn([vUser], [vPassword], [vMenu])  
bOK      (Boolean) Optional – Returns True if the session was successfully logged on.  
vUser      (Variant) Optional – value accessed by SM.SessionUser  
vPassword      (Variant) Optional – value accessed by SM.SessionPwd  
vMenu      (Variant) Optional – value specifying the menu to log in to

Example:

```
Dim bSuccess As Boolean  
bSuccess = SM.LogOn  
bSuccess = SM.LogOn("UserName", "Password",  
"MainMenu")
```

## PadInput

This command adds spaces to the end of a string until the total length of the string is the size specified in the command. Using a number too small will truncate the string.

Syntax: SM.PadInput(sText, iLength)  
sText      (String) the string of characters to be padded

iLength (Integer) the number of spaces to add to the end of the string

Example:

```
Dim sPartNo as String  
sPartNo = "12345"  
SM.PadInput(sPartNo, 8)
```

Result is “12345 ” with 3 spaces at the end

## PingHost

This command sends a single ping packet to host and waits for response. It works against the current connection and can be used in transaction or navigation macros, used against a local connection, or you can get a pooled connection (SM.BeginTrans) and then use it to test the connection.

Syntax: bOK = SM.PingHost

bOK (Boolean) returns True / False if the server can be reached

Example:

```
Dim bOK As Boolean  
bOK = SM.PingHost
```

## ResetConnection

This command will reset the active host session in an effort to re-establish a locked-up connection.

Syntax: SM.ResetConnection

Example:

```
SM.ResetConnection
```

## SendCTRL

This function sends the <CTRL> code with the specified character. It returns True if the key was successfully entered. Note: reasons for a negative or False value might be that keyboard entry is inhibited, or that the cursor was not in an input field.

Syntax: [bOK =] SM.SendCTRL(sKey)

bOK (Boolean) Optional – Returns True if the key was successfully sent to the host session.

sKey (String) Valid characters are: A-Z, space, [ , / , ] , ~ , ? , 0-9

Example:

```
Dim bOK As Boolean  
bOK = SM.SendCTRL("C")
```

## SendCTRLAlt

This function sends the <CTRL> code with the specified character to a specific location in host session's screen. It returns True if the key was successfully entered. Reasons for a negative or False value might be that keyboard entry is inhibited, or that the coordinates were not an input field. Note: the ability to move the cursor to a specific location is only supported for tn5250 and tn3270 sessions.

Syntax: [bOK =] SM.SendCTRLAlt(iCol, iRow, sKey)

bOK (Boolean) Optional – Returns True if the key was successfully sent to the host session.

iCol (Integer) The screen column position to input the key.

iRow (Integer) The screen row position to input the key.

sKey (String) Valid characters are: A-Z, space, [ , / , ] , ~ , ? , 0-9

Example:

```
Dim bOK As Boolean  
bOK = SM.SendCTRLAlt(8, 12, "C")
```

## SendKey

This function sends the selected key to the current cursor location in the host session's screen. It returns True if the key was successfully entered. Note: reasons for a negative or False value might be that keyboard entry is inhibited, or that the cursor was not in an input field.

Syntax: [bOK =] SM.SendKey(enKey)

bOK (Boolean) Optional – Returns True if the key was successfully sent to the host session.

enKey (enCommandKeys) Select from the drop-down list the desired key to send to the host session. Options are:

KeyAttention	KeyBackspace
KeyBackTab	KeyBreak
KeyClear	KeyCursorDown
KeyCursorLeft	KeyCursorRight
KeyCursorUp	KeyDelete
KeyDo	KeyDuplicate
KeyEnd	KeyEnter
KeyEscape	KeyF1 – KeyF24
KeyFieldAdvance	KeyFieldBack
KeyFieldErase	KeyFieldExit
KeyFieldMark	KeyFieldMinus
KeyFieldPlus	KeyFind
KeyHelp	KeyHome
KeyInsert	KeyPA1 – KeyPA3
KeyPageDown	KeyPageUp
KeyRemove	KeyReplace
KeyReset	KeySelect
KeySystemRequest	KeyTab

Example:

```
Dim bOK As Boolean
bOK = SM.SendKey(KeyEnter)
```

## SendKeyAlt

This function sends the selected key to a specific location in host session's screen. It returns True if the key was successfully entered. Reasons for a negative or False value might be that keyboard entry is inhibited, or that the coordinates were not an input field. Note: the ability to move the cursor to a specific location is only supported for tn5250 and tn3270 sessions.

Syntax: [bOK =] SM.SendKeyAlt(iCol, iRow, enKey)

bOK (Boolean) Optional – Returns True if the key was successfully sent to the host session.

iCol (Integer) The screen column position to input the key.

iRow (Integer) The screen row position to input the key.

enKey (enCommandKeys) Select from the drop-down list the desired key to send to the host session. For a list see SM.SendKey.

Example:

```
Dim bOK As Boolean
bOK = SM.SendKeyAlt(8, 12, KeyEnter)
```

## SendText

This function is used to send text to the current cursor location in the host session's screen. It returns True if the text was successfully

entered. Note: reasons for a negative or False value might be that keyboard entry is inhibited, that the cursor was not in an input field, or that the text exceeded the input field's length.

Syntax: [bOK =] SM.SendText(sText, [iPadSize], [vPadChar])

bOK       (Boolean) Optional – Returns True if the text was successfully sent to the host session.

sText      (String) The desired text to send to the host session.

iPadSize   (Integer) Optional – total length of padded string

vPadChar  (Variant) Optional – the character to use for padding

Examples:

```
Dim bOK As Boolean  
bOK = SM.SendText("SAM")  
SM.SendText("SAM", 10, " ")
```

## SendTextAlt

This function is used to send text to a specific cursor location in the host session's screen. It returns True if the text was successfully entered. Reasons for a negative or False value might be that keyboard entry is inhibited, that the coordinates were not an input field, or that the text exceeded the input field's length. Note: the ability to move the cursor to a specific location is only supported for tn5250 and tn3270 sessions.

Syntax: [bOK =] SM.SendTextAlt(iCol, iRow, sText, [iPadSize], [vPadChar])

bOK       (Boolean) Optional – Returns True if the text was successfully sent to the host session.

iCol       (Integer) The screen column position to input the text.

iRow       (Integer) The screen row position to input the text.

sText      (String) The desired text to send to the host session.

iPadSize   (Integer) Optional – total length of padded string

vPadChar  (Variant) Optional – the character to use for padding

Example:

```
Dim bOK As Boolean  
bOK = SM.SendTextAlt(10, 8, "SAM")
```

## SessionID

This function is used return the current session or pool number. If Connection Pooling is disabled, the Transaction Manager may have to establish its own connection if macros are queued. If this is the case the returned integer will be 0. If Connection Pooling is enabled, the pool number (1, 2, 3 etc.) will be returned.

Syntax: iValue = SM.SessionID

iValue (Integer) stores the pool number used by the client

Example:

```
Dim iValue as Integer  
iValue = SM.SessionID
```

## SessionPwd

This function is used to set or return the Password associated with a host session. For pooled connections the password can be defined under Connections / Connection X (Screen Mapping) / Pooling option. For non-pooled connections the SM.SessionPwd can assign the password.

Syntax: sValue = SM.SessionPwd

Alternate: SM.SessionPwd = sValue

sValue (String) variable containing the password

Example:

```
Dim sValue as String  
sValue = SM.SessionPwd
```

## SessionUser

This function is used to set or return the User ID associated with a host session. For pooled connections the user can be defined under the Connections / Connection X (Screen Mapping) / Connection Pooling option. For non-pooled connections the SM.SessionUser can assign the user ID.

Syntax: sValue = SM.SessionUser

Alternate: SM.SessionUser = sValue

sValue (String) variable containing the user ID

Example:

```
Dim sValue As String  
sValue = SM.SessionUser
```

## SetBase

This function is used to switch to an alternate Main Menu. This command must be run while on a menu screen and not an application screen.

Syntax: [bOK =] SM.SetBase(sMenu)

bOK       (Boolean) Optional – returns True / False depending on the success of the command  
sMenu      (String) the name of the new base menu

Example:

```
Dim bOK As Boolean  
bOK = SM.SetBase("MainMenu")
```

## SetCursor

This function is used to move the cursor to a specified location on the host screen. It returns True if the cursor was successfully moved to the requested location. Note: the ability to move the cursor to a specific location is only supported for tn5250 and tn3270 sessions.

Syntax: [bOK =] SM.SetCursor(iCol, iRow)

bOK       (Boolean) Optional – Returns True if the cursor was successfully moved in the host session.

iCol       (Integer) The desired new column position for the cursor.

iRow       (Integer) The desired new row position for the cursor.

Example:

```
Dim bOK As Boolean  
bOK = SM.SetCursor(2, 9)
```

## SetDelay

This function is typically used for debugging purposes against a vt220 host session. It allows users to set a delay, in milliseconds, for all screen mapping VBA extensions that change what is on the host screen. This allows the user to watch in slow motion, all screen updates as the result of a macro. Note: the user could also accomplish this by single stepping through the macro's VBA code in the test environments VBA debug window.

Syntax: SM.SetDelay(nMilliseconds)

nMilliseconds   (Long) The delay to set in milliseconds.

Example:

```
SM.SetDelay(1000)
```

## SetSession

This function is used when you are connecting to multiple hosts via Screen Mapping. It allows you to specify which screen mapping session is active. Note: the first screen mapping connection is set as the active session by default.

Syntax: [bOK =] SM.SetSession(vSource)

bOK      (Boolean) Optional – Returns True if the requested session is defined as a valid Screen Mapping Connection.

vSource    (Variant) The connection number or the name of the host session (as displayed in the status bar).

Example:

```
Dim bOK As Boolean  
bOK = SM.SetSession(2)
```

## SetTimeout

This function is used with all other SM commands, providing a maximum length of time before the other SM commands fail after no response from the host system. The internal default is 5 seconds.

Syntax: SM.setTimeout(nSeconds)

nSeconds    (Long) The number of seconds

Example:

```
SM.setTimeout (10)
```

## WaitForCursor

This function is used to time your commands to the host session. With it you can delay sending text or keys to the host session, or retrieving data from the host session until the cursor is in a specific location. It basically allows you to wait for the cursor to arrive at a specific position in the host screen for a certain amount of time. Once the cursor reaches the desired location this function will immediately return with a value of True, otherwise it will timeout and return False.

Syntax: [bOK =] SM.WaitForCursor(iCol, iRow, iSeconds)

bOK      (Boolean) Optional – Immediately returns True if the cursor stopped at the desired location.

iCol      (Integer) The column position to wait for cursor.

iRow      (Integer) The row position to wait for cursor.

iSeconds    (Integer) The maximum number of seconds to wait for the cursor to reach the requested position.

Example:

```
Dim bOK As Boolean  
bOK = SM.WaitForCursor(6, 14, 5)
```

## WaitForCursorMove

This function is also used to time your commands to the host session. With this command, you specify only an amount of time in seconds. If the cursor has changed positions within that time, a True result is returned. Otherwise it will timeout and return False.

Syntax: [bOK =] SM.WaitForCursorMove(iSeconds)

bOK      (Boolean) Optional – Immediately returns True if the cursor has changed locations.

iSeconds    (Integer) – The maximum number of seconds to wait for the cursor to change locations.

Example:

```
Dim bOK As Boolean  
bOK = SM.WaitForCursorMove(5)
```

## WaitForHost

This function is used to time your commands to a vt220 host session. With it you can delay sending text or keys to the host session, or retrieving data from the host session until the host has responded to the last command sent. Whenever input data is sent to a host session, the server sets a switch that indicates whether the host has responded. This function basically allows you to wait for a certain amount of time until the host processes and responds to the last input command. Once the host response has been received and processed by the server this function will immediately return with a value of True, otherwise it will timeout and return False.

Note: There is a difference with the VT environment where this command will return a 'True' after the last command finishes even if the host is not ready for the keyboard input. This is because in the VT environment, the keyboard is never locked and in the 5250 and 3270 environments, this commands waits for the keyboard to be unlocked before returning any values.

Syntax: [bReplyReceived =] SM.WaitForHost(iSeconds)

bReplyReceived    (Boolean) Optional – Immediately returns True once the host responds to your last input command.

iSeconds          (Integer) The maximum number of seconds to wait for the host to respond to your last input command.

Example:

```
Dim bReplyReceived As Boolean  
bReplyReceived = SM.WaitForHost(5)
```

## WaitForScreen

This function is used to confirm that we have reached the desired host application screen. With it you can positively confirm that the 'GoToScreen' or 'ReturnToMainMenu' functions have succeeded. It allows you to wait for a certain amount of time for the desired screen to be positively identified. Once the ID match has occurred, this function will immediately return with a value of True, otherwise it will timeout and return False.

Syntax: [bOK =] SM.WaitForScreen(sScreenName, iSeconds)

bOK	(Boolean) Optional – Immediately returns True if the host session is now in the desired screen.
sScreenName	(String) The host screen that we wish to confirm is active.
iSeconds	(Integer) The maximum number of seconds to wait for the screen to be identified.

Example:

```
Dim bOK As Boolean  
bOK = SM.WaitForScreen("InventoryMovements", 5)
```

## WaitForText

This function is used to time your commands to the host session. With it you can delay sending text or keys to the host session, or retrieving data from the host session until a specific text string has appeared on the screen. It allows you to wait for a text string to appear anywhere on the screen, or only at a specific position. Once the text appears, this function will immediately return with a value of True, otherwise it will timeout and return False.

Syntax: [bFound =] SM.WaitForText(sText, iSeconds, [iCol], [iRow])

bFound	(Boolean) Optional – Immediately returns True once the specified text appears on the screen.
sText	(String) The text to find.
iSeconds	(Integer) The maximum number of seconds to wait for the screen to be identified.
iCol	(Integer) Optional – the column position to look for the text. Use '-1' for any Column
iRow	(Integer) Optional – the row position to look for the text. Use '-1' for any row.

Example:

```
Dim bFound As Boolean  
bFound = SM.WaitForText("Enter Next Task Code:", 5, -  
1, -1)
```

## **WaitForWrite**

This function waits for a specified number of seconds for data to be entered at a specific location and returns a True or False. If data was written within the wait time, True is returned. If the number of seconds expires first, False is returned.

Syntax: [bOK =] SM.WaitForWrite(iCol, iRow, iSeconds)

bOK           (Boolean) Optional – Returns True if data was written at the specified coordinates within the specified time frame

iCol           (Integer) The column position to watch.

iRow           (Integer) The row position to watch.

iSeconds      (Integer) The maximum number of seconds to wait for data to be written.

Example:

```
Dim bOK As Boolean  
bOK = SM.WaitForWrite(24,13,5)
```

## **Chart Object**

A chart can be displayed on the screen if a graphical representation of data is required, however only Thin client mode is supported. Both 2D and 3D charts are supported depending on the type of chart chosen. This is the list of available chart types.

<u>Chart Type</u>	<u>Can be 3D</u>
Area	Yes
Bar	Yes
Bubble	No
Candle	No
FilledRadar	No
HiLo	No
HiLoOpenClose	No
Pie	Yes
Plot	Yes
Polar	No
Radar	No
StackingBar	Yes
Surface	Yes

## **Chart**

This object is an advanced method of creating a graphical chart that may then be presented to the user for selection.

```
Dim oChart as New Chart
```

This object has the following methods and properties:

## AddDataPoint

This method adds all the data to the different series that make up the chart. Depending on the chart there may be one or multiple series that make up the chart. For example a bar chart requires as many series as there are bars in the chart.

Syntax: oChart.AddDataPoint(nSeries, nPoint, vX, vY, [v3Dvalue])

nSeries (Long) a number specifying the series number distinguishing the different lines, bars or pie wedges of the chart  
nPoint (Long) The sequence number of points that make up the complete range of the specified series  
vX (Variant) The X coordinate of the specified point for the specified series  
vY (Variant) The Y coordinate of the specified point for the specified series  
v3DValue (Variant) for future use

Examples:

```
oChart.SeriesLabel(1) = "Prius"
oChart.AddDataPoint 1, 1, 10, 50
oChart.AddDataPoint 1, 2, 20, 60

oChart.SeriesLabel(2) = "Corvette"
oChart.AddDataPoint 2, 1, 10, 30
oChart.AddDataPoint 2, 2, 20, 40
```

## AxesFont

This method sets the font and size for the specified axis labels. These labels are usually the range of numbers that determine the scope of the axis.

Syntax: oChart.AxesFont(sAxes, [vTypeface], [vSize])

sAxes (String) the name of the axis such as "X" or "Y"  
vTypeface (Variant) The font to be used for the specified axis label  
vSize (Variant) The size of the font for the specified axis label

Examples:

```
oChart.AxesFont "X", "Verdana", 12
```

```
oChart.AxesFont "Y", "Verdana", 12
```

## AxesLabel

This property sets the title of the specified axis. This describes the meaning of the numerical scale on that axis.

Syntax: oChart.AxesLabel(sAxes) = sValue

sAxes (String) the name of the axis such as "X" or "Y"

sValue (String) the text to be displayed for the axis title

Examples:

```
oChart.AxesLabel("X") = "Years of Ownership"
```

```
oChart.AxesLabel("Y") = "Tickets"
```

## AxesTitleFont

This method sets the font and size for the specified axis title.

Syntax: oChart.AxesFont(sAxes, [vTypeface], [vSize])

sAxes (String) the name of the axes such as "X" or "Y"

vTypeface (Variant) The font to be used for the axes label

vSize (Variant) The size of the font for the axes label

Examples:

```
oChart.AxesFont "X", "Verdana", 12
```

```
oChart.AxesFont "Y", "Verdana", 12
```

## BackColor

This property sets the color of the background of the chart.

Syntax: oChart.BackColor = nValue

nValue (Long) a number or VB constant representing a color

Examples:

```
oChart.BackColor = vbRed           'red
```

```
oChart.BackColor = RGB(255,255,0) 'yellow
```

```
oChart.BackColor = &HFF0000      'blue
```

```
oChart.BackColor = QBColor(5)     'magenta
```

## ForeColor

This property sets the color of the foreground of the chart. This includes the title, axes scales, axes titles and legend text.

Syntax: oChart.ForeColor = nValue

nValue (Long) a number or VB constant representing a color

Examples:

```
oChart.ForeColor = vbRed           'red  
oChart.ForeColor = RGB(255,255,0) 'yellow  
oChart.ForeColor = &HFF0000      'blue  
oChart.ForeColor = QBColor(5)     'magenta
```

## Header

This property sets the text for the header of the chart.

Syntax: oChart.Header = sValue

sValue (String) the text of the header

Example:

```
oChart.Header = "Speeding Tickets"
```

## HeaderFont

This method sets the font and size for the chart title.

Syntax: oChart.HeaderFont([vTypeface], [vSize])

vTypeface (Variant) The font to be used for the title label

vSize (Variant) The size of the font for the title label

Examples:

```
oChart.HeaderFont "Verdana", 12  
oChart.HeaderFont "Verdana", 12
```

## Height

This property is the height in pixels for the chart size. The larger the outside rectangle the larger the chart graphic will expand to fill that rectangle. Note: this property is not required. Each chart has a default size that will be used that may be sufficient.

Syntax: oChart.Height = nValue

nValue (Long) The number in pixels of the chart rectangle height.

Example:

```
oChart.Height = 300
```

## **Image**

This property contains the byte array of the chart image as formulated by the chart object. This property is the value assigned to the Image control's Bitmap property so that image control can display the chart.

Syntax: `Image1.Bitmap = oChart.Image`

Example:

```
Image1.Bitmap = oChart.Image  
Screen.Refresh
```

The Screen.Refresh command may not be necessary if the procedure is about to end. In this case the server will repaint the screen automatically.

## **LegendFont**

This method sets the font and size for the legend of the chart.

Syntax: `oChart.LegendFont([vTypeface], [vSize])`

vTypeface (Variant) The font to be used for the legend  
vSize (Variant) The size of the font for the legend

Example:

```
oChart.LegendFont "Verdana", 12
```

## **SeriesColor**

This property sets the color of the specified series. For example, it sets the color for the specific bar, plot line or pie wedge in the chart.

Syntax: `oChart.SeriesColor(nSeries) = nValue`

nSeries (Long) a number representing the series  
nValue (Long) a number or VB constant representing a color

Examples:

```
oChart.SeriesColor(1) = vbRed           'red  
oChart.SeriesColor(2) = RGB(255,255,0) 'yellow  
oChart.SeriesColor(3) = &HFF0000       'blue  
oChart.SeriesColor(4) = QBColor(5)      'magenta
```

## **SeriesLabel**

This property sets the title of the series in the legend. This describes the meaning of the color of this series.

Syntax: oChart.SeriesLabel(nSeries) = sValue

nSeries (Long) a number representing the series

sValue (String) the name of the series displayed in the legend

Examples:

`oChart.SeriesLabel(1) = "Prius"`

`oChart.SeriesLabel(2) = "Corvette"`

## ThreeD

This property sets the chart to a 3D mode or a 2D mode. Not all charts support the 3D option. See the chart at the beginning of this section for a complete list.

Syntax: oChart.ThreeD = bValue

bValue (Boolean) Set to True for drawing the chart in 3D

Example:

`oChart.ThreeD = True`

## Type

This property specifies which chart type should be used.

Syntax: oChart.Type = enChartType

enChartType (Enumeration) Set to the desired chart type. The next section of Chart Examples shows pictures of each chart.  
The list of available charts are:

`chtTypeArea`

`chtTypeCandle`

`chtTypeHiLoOpenClose`

`chtTypePolar`

`chtTypeSurface`

`chtTypeBar`

`chtTypeFilledRadar`

`chtTypePie`

`chtTypeRadar`

`chtTypeBubble`

`chtTypeHiLo`

`chtTypePlot`

`chtTypeStackingBar`

Example:

`oArea.Type = chtTypeArea`

## Width

This property is the width in pixels for the chart size. The larger the outside rectangle the larger the chart graphic will expand to fill that rectangle. Note: this property is not required. Each chart has a default size that will be used that may be sufficient.

Syntax: oChart.Width = nValue

nValue (Long) The number in pixels of the chart rectangle width.

Example:

```
oChart.Width = 300
```

# Chart Examples

## Area



```
Public Sub DrawArea
    Dim oArea As New Chart
    Const sFontName As String = "Verdana"
    Const dFontSize As Double = 6.5

    oArea.Type = chtTypeArea
    oArea.Width = 300
    oArea.Height = 200
    oArea.Header = "Speeding Tickets"
    oArea.LegendFont sFontName, dFontSize
    oArea.HeaderFont sFontName, dFontSize

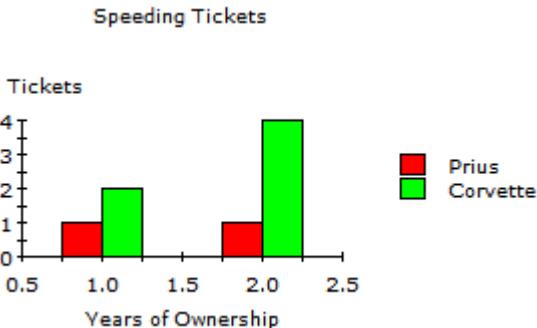
    oArea.SeriesColor(1) = vbRed
    oArea.SeriesLabel(1) = "Prius"
    oArea.AddDataPoint 1, 1, 1, 5
    oArea.AddDataPoint 1, 2, 2, 6

    oArea.SeriesColor(2) = vbBlue
    oArea.SeriesLabel(2) = "Corvette"
    oArea.AddDataPoint 2, 1, 1, 3
    oArea.AddDataPoint 2, 2, 2, 4

    oArea.SeriesColor(3) = vbGreen
    oArea.SeriesLabel(3) = "Pinto"
    oArea.AddDataPoint 3, 1, 1, 1
    oArea.AddDataPoint 3, 2, 2, 2

    img1.Bitmap = oArea.Image
End Sub
```

## Bar



```
Public Sub DrawBar
    On Error Resume Next
    '
    Dim oBar As New Chart
    Const sFontName As String = "Verdana"
    Const dFontSize As Double = 6.5

    oBar.Width = 300
    oBar.Height = 200
    oBar.Header = "Speeding Tickets"
    oBar.Type = chtTypeBar
    oBar.AxesLabel("X") = "Years of Ownership"
    oBar.AxesLabel("Y") = "Tickets"
    oBar.AxesTitleFont "X", sFontName, dFontSize
    oBar.AxesTitleFont "Y", sFontName, dFontSize
    oBar.AxesFont "X", sFontName, dFontSize
    oBar.AxesFont "Y", sFontName, dFontSize
    oBar.LegendFont sFontName, dFontSize
    oBar.HeaderFont sFontName, dFontSize

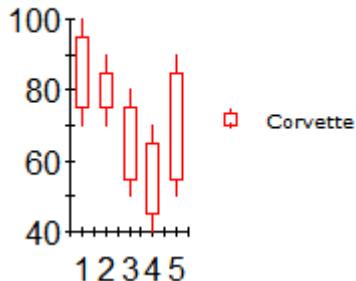
    oBar.SeriesLabel(1) = "Prius"
    oBar.SeriesColor(1) = vbRed
    oBar.AddDataPoint 1, 1, 1, 1
    oBar.AddDataPoint 1, 2, 2, 1

    oBar.SeriesLabel(2) = "Corvette"
    oBar.SeriesColor(2) = vbGreen
    oBar.AddDataPoint 2, 1, 1, 2
    oBar.AddDataPoint 2, 2, 2, 4

    img1.Bitmap = oBar.Image
End Sub
```

## Candle

Speeding Tickets



```
Public Sub DrawCandle
On Error Resume Next
'
Dim oCandle As New Chart
Const sFontName As String = "Verdana"
Const dFontSize As Double = 6.5

oCandle.Type = chtTypeCandle
oCandle.Width = 200
oCandle.Height = 200
oCandle.Header = "Speeding Tickets"
oCandle.LegendFont sFontName, dFontSize
oCandle.HeaderFont sFontName, dFontSize

oCandle.SeriesColor(1) = vbRed
oCandle.SeriesLabel(1) = "Corvette"
' High points - Corvette
oCandle.AddDataPoint 1, 1, 1, 100
oCandle.AddDataPoint 1, 2, 2, 90
oCandle.AddDataPoint 1, 3, 3, 80
oCandle.AddDataPoint 1, 4, 4, 70
oCandle.AddDataPoint 1, 5, 5, 90

' Low points - Corvette
oCandle.AddDataPoint 2, 1, 1, 70
oCandle.AddDataPoint 2, 2, 2, 70
oCandle.AddDataPoint 2, 3, 3, 50
oCandle.AddDataPoint 2, 4, 4, 40
oCandle.AddDataPoint 2, 5, 5, 50

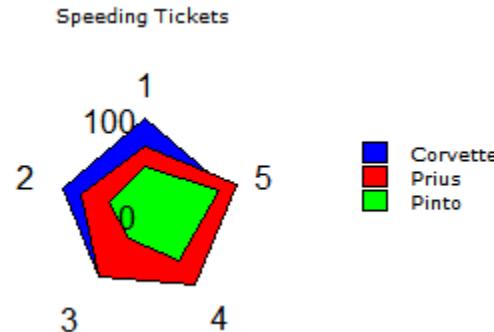
' Low Open points
oCandle.AddDataPoint 3, 1, 1, 75
```

```
oCandle.AddDataPoint 3, 2, 2, 75
oCandle.AddDataPoint 3, 3, 3, 55
oCandle.AddDataPoint 3, 4, 4, 45
oCandle.AddDataPoint 3, 5, 5, 55

' High Close points
oCandle.AddDataPoint 4, 1, 1, 95
oCandle.AddDataPoint 4, 2, 2, 85
oCandle.AddDataPoint 4, 3, 3, 75
oCandle.AddDataPoint 4, 4, 4, 65
oCandle.AddDataPoint 4, 5, 5, 85

img1.Bitmap = oCandle.Image
End Sub
```

## FilledRadar



```
Public Sub DrawFilledRadar
On Error Resume Next
'
Dim oFRadar As New Chart
Const sFontName As String = "Verdana"
Const dFontSize As Double = 6.5

oFRadar.Type = chtTypeFilledRadar
oFRadar.Width = 300
oFRadar.Height = 200
oFRadar.Header = "Speeding Tickets"
oFRadar.LegendFont sFontName, dFontSize
oFRadar.HeaderFont sFontName, dFontSize

oFRadar.SeriesColor(1) = vbBlue
oFRadar.SeriesLabel(1) = "Corvette"
oFRadar.AddDataPoint 1, 1, 1, 100
oFRadar.AddDataPoint 1, 2, 2, 90
oFRadar.AddDataPoint 1, 3, 3, 80
oFRadar.AddDataPoint 1, 4, 4, 70
oFRadar.AddDataPoint 1, 5, 5, 90

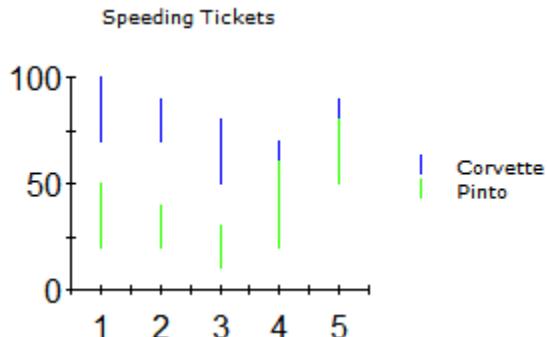
oFRadar.SeriesColor(2) = vbRed
oFRadar.SeriesLabel(2) = "Prius"
oFRadar.AddDataPoint 2, 1, 1, 70
oFRadar.AddDataPoint 2, 2, 2, 70
oFRadar.AddDataPoint 2, 3, 3, 80
oFRadar.AddDataPoint 2, 4, 4, 90
oFRadar.AddDataPoint 2, 5, 5, 100

oFRadar.SeriesColor(3) = vbGreen
oFRadar.SeriesLabel(3) = "Pinto"
```

```
oFRadar.AddDataPoint 3, 1, 1, 50
oFRadar.AddDataPoint 3, 2, 2, 40
oFRadar.AddDataPoint 3, 3, 3, 30
oFRadar.AddDataPoint 3, 4, 4, 60
oFRadar.AddDataPoint 3, 5, 5, 80

img1.Bitmap = oFRadar.Image
End Sub
```

## HiLo



```
Public Sub DrawHiLo
    On Error Resume Next
    '
    Dim oHiLo As New Chart
    Const sFontName As String = "Verdana"
    Const dFontSize As Double = 6.5

    oHiLo.Type = chtTypeHiLo
    oHiLo.Width = 300
    oHiLo.Height = 200
    oHiLo.Header = "Speeding Tickets"
    oHiLo.LegendFont sFontName, dFontSize
    oHiLo.HeaderFont sFontName, dFontSize

    oHiLo.SeriesColor(1) = vbBlue
    oHiLo.SeriesLabel(1) = "Corvette"
    oHiLo.SeriesColor(2) = vbGreen
    oHiLo.SeriesLabel(2) = "Pinto"

    ' High points - Corvette
    oHiLo.AddDataPoint 1, 1, 1, 100
    oHiLo.AddDataPoint 1, 2, 2, 90
    oHiLo.AddDataPoint 1, 3, 3, 80
    oHiLo.AddDataPoint 1, 4, 4, 70
    oHiLo.AddDataPoint 1, 5, 5, 90

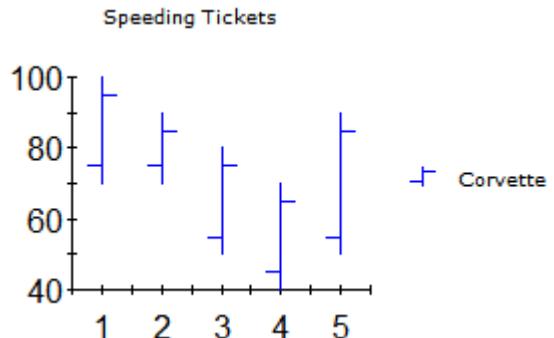
    ' Low points - Corvette
    oHiLo.AddDataPoint 2, 1, 1, 70
    oHiLo.AddDataPoint 2, 2, 2, 70
    oHiLo.AddDataPoint 2, 3, 3, 50
    oHiLo.AddDataPoint 2, 4, 4, 40
    oHiLo.AddDataPoint 2, 5, 5, 50
```

```
' High points - Pinto
oHiLo.AddDataPoint 3, 1, 1, 50
oHiLo.AddDataPoint 3, 2, 2, 40
oHiLo.AddDataPoint 3, 3, 3, 30
oHiLo.AddDataPoint 3, 4, 4, 60
oHiLo.AddDataPoint 3, 5, 5, 80

' Low points - Pinto
oHiLo.AddDataPoint 4, 1, 1, 20
oHiLo.AddDataPoint 4, 2, 2, 20
oHiLo.AddDataPoint 4, 3, 3, 10
oHiLo.AddDataPoint 4, 4, 4, 20
oHiLo.AddDataPoint 4, 5, 5, 50

    img1.Bitmap = oHiLo.Image
End Sub
```

## HiLo OpenClose



```
Public Sub DrawHiLoOpenClose
On Error Resume Next
'
Dim oHiLoOC As New Chart
Const sFontName As String = "Verdana"
Const dFontSize As Double = 6.5

img1.Caption = "HiLo Open Close Chart"

oHiLoOC.Type = chtTypeHiLoOpenClose
oHiLoOC.Width = 300
oHiLoOC.Height = 200
oHiLoOC.Header = "Speeding Tickets"
oHiLoOC.LegendFont sFontName, dFontSize
oHiLoOC.HeaderFont sFontName, dFontSize

oHiLoOC.SeriesColor(1) = vbBlue
oHiLoOC.SeriesLabel(1) = "Corvette"

' High points - Corvette
oHiLoOC.AddDataPoint 1, 1, 1, 100
oHiLoOC.AddDataPoint 1, 2, 2, 90
oHiLoOC.AddDataPoint 1, 3, 3, 80
oHiLoOC.AddDataPoint 1, 4, 4, 70
oHiLoOC.AddDataPoint 1, 5, 5, 90

' Low points - Corvette
oHiLoOC.AddDataPoint 2, 1, 1, 70
oHiLoOC.AddDataPoint 2, 2, 2, 70
oHiLoOC.AddDataPoint 2, 3, 3, 50
oHiLoOC.AddDataPoint 2, 4, 4, 40
oHiLoOC.AddDataPoint 2, 5, 5, 50
```

```
' Low Open points
oHiLoOC.AddDataPoint 3, 1, 1, 75
oHiLoOC.AddDataPoint 3, 2, 2, 75
oHiLoOC.AddDataPoint 3, 3, 3, 55
oHiLoOC.AddDataPoint 3, 4, 4, 45
oHiLoOC.AddDataPoint 3, 5, 5, 55

' High Close points
oHiLoOC.AddDataPoint 4, 1, 1, 95
oHiLoOC.AddDataPoint 4, 2, 2, 85
oHiLoOC.AddDataPoint 4, 3, 3, 75
oHiLoOC.AddDataPoint 4, 4, 4, 65
oHiLoOC.AddDataPoint 4, 5, 5, 85

    img1.Bitmap = oHiLoOC.Image
End Sub
```

## Pie

Speeding Tickets



```
Public Sub DrawPie
    On Error Resume Next
    '
    Dim oPie As New Chart
    Const sFontName As String = "Verdana"
    Const dFontSize As Double = 6.5

    oPie.Type = chtTypePie
    oPie.Width = 300
    oPie.Height = 300
    oPie.Header = "Speeding Tickets"
    oPie.LegendFont sFontName, dFontSize
    oPie.HeaderFont sFontName, dFontSize

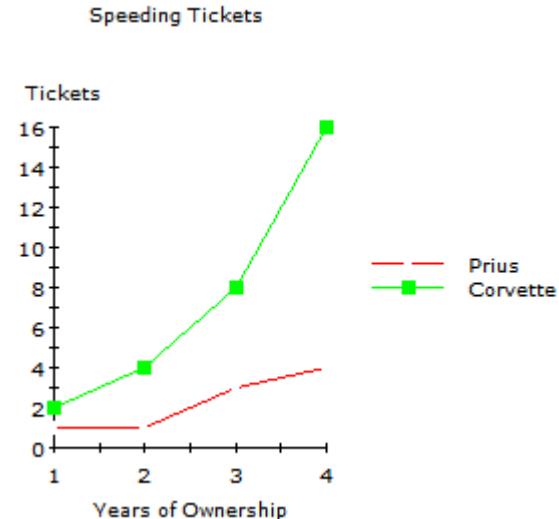
    oPie.SeriesColor(1) = vbRed
    oPie.SeriesLabel(1) = "Prius"
    oPie.AddDataPoint 1, 1, 1, 3 'Prius

    oPie.SeriesColor(2) = vbBlue
    oPie.SeriesLabel(2) = "Corvette"
    oPie.AddDataPoint 2, 1, 1, 6 'Corvette

    oPie.SeriesColor(3) = vbGreen
    oPie.SeriesLabel(3) = "Pinto"
    oPie.AddDataPoint 3, 1, 1, 1 'Pinto

    img1.Bitmap = oPie.Image
End Sub
```

## Plot



```
Public Sub DrawPlot
    On Error Resume Next
    '
    Dim oPlot As New Chart
    Const sFontName As String = "Verdana"
    Const dFontSize As Double = 6.5

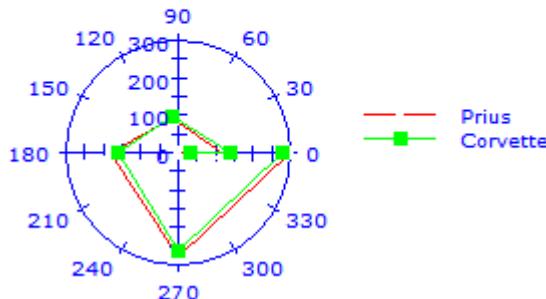
    oPlot.Width = 300
    oPlot.Height = 300
    oPlot.Header = "Speeding Tickets"
    oPlot.Type = chtTypePlot
    oPlot.AxesLabel("X") = "Years of Ownership"
    oPlot.AxesLabel("Y") = "Tickets"
    oPlot.AxesTitleFont "X", sFontName, dFontSize
    oPlot.AxesTitleFont "Y", sFontName, dFontSize
    oPlot.AxesFont "X", sFontName, dFontSize
    oPlot.AxesFont "Y", sFontName, dFontSize
    oPlot.LegendFont sFontName, dFontSize
    oPlot.HeaderFont sFontName, dFontSize

    'Prius
    oPlot.SeriesLabel(1) = "Prius"
    oPlot.SeriesColor(1) = vbRed
    oPlot.AddDataPoint 1, 1, 1, 1
    oPlot.AddDataPoint 1, 2, 2, 1
    oPlot.AddDataPoint 1, 3, 3, 3
```

```
oPlot.AddDataPoint 1, 4, 4, 4  
  
'Corvette  
oPlot.SeriesColor(2) = vbGreen  
oPlot.SeriesLabel(2) = "Corvette"  
oPlot.AddDataPoint 2, 1, 1, 2  
oPlot.AddDataPoint 2, 2, 2, 4  
oPlot.AddDataPoint 2, 3, 3, 8  
oPlot.AddDataPoint 2, 4, 4, 16  
  
img1.Bitmap = oPlot.Image  
End Sub
```

## Polar

Speeding Tickets



```
Public Sub DrawPolar
On Error Resume Next
'
Dim oPolar As New Chart
Const sFontName As String = "Verdana"
Const dFontSize As Double = 6.5

oPolar.Width = 300
oPolar.Height = 300

oPolar.Header = "Speeding Tickets"
oPolar.Type = chtTypePolar
oPolar.AxesLabel("X") = "Years of Ownership"
oPolar.AxesLabel("Y") = "Tickets"
oPolar.AxesTitleFont "X", sFontName, dFontSize
oPolar.AxesTitleFont "Y", sFontName, dFontSize
oPolar.AxesFont "X", sFontName, dFontSize
oPolar.AxesFont "Y", sFontName, dFontSize
oPolar.LegendFont sFontName, dFontSize
oPolar.HeaderFont sFontName, dFontSize
oPolar.ForeColor = vbBlue

'Prius
oPolar.SeriesLabel(1) = "Prius"
oPolar.SeriesColor(1) = vbRed
oPolar.AddDataPoint 1, 1, 0, 0
oPolar.AddDataPoint 1, 2, 0, 120
oPolar.AddDataPoint 1, 3, 100, 90
```

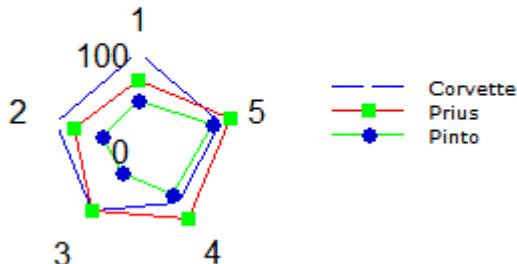
```
oPolar.AddDataPoint 1, 4, 0, -180
oPolar.AddDataPoint 1, 5, 270, 280
oPolar.AddDataPoint 1, 6, 0, 300

'Corvette
oPolar.SeriesColor(2) = vbGreen
oPolar.SeriesLabel(2) = "Corvette"
oPolar.AddDataPoint 2, 1, 30, 30
oPolar.AddDataPoint 2, 2, 30, 140
oPolar.AddDataPoint 2, 3, 140, 100
oPolar.AddDataPoint 2, 4, 10, -160
oPolar.AddDataPoint 2, 5, 250, 260
oPolar.AddDataPoint 2, 6, 10, 280

img1.Bitmap = oPolar.Image
End Sub
```

## Radar

Speeding Tickets



```
Public Sub DrawRadar
On Error Resume Next
'
Dim oRadar As New Chart
Const sFontName As String = "Verdana"
Const dFontSize As Double = 6.5

oRadar.Type = chtTypeRadar
oRadar.Width = 300
oRadar.Height = 200
oRadar.Header = "Speeding Tickets"
oRadar.LegendFont sFontName, dFontSize
oRadar.HeaderFont sFontName, dFontSize

oRadar.SeriesColor(1) = vbBlue
oRadar.SeriesLabel(1) = "Corvette"
oRadar.AddDataPoint 1, 1, 1, 100
oRadar.AddDataPoint 1, 2, 2, 90
oRadar.AddDataPoint 1, 3, 3, 80
oRadar.AddDataPoint 1, 4, 4, 70
oRadar.AddDataPoint 1, 5, 5, 90

oRadar.SeriesColor(2) = vbRed
oRadar.SeriesLabel(2) = "Prius"
oRadar.AddDataPoint 2, 1, 1, 70
oRadar.AddDataPoint 2, 2, 2, 70
oRadar.AddDataPoint 2, 3, 3, 80
oRadar.AddDataPoint 2, 4, 4, 90
oRadar.AddDataPoint 2, 5, 5, 100

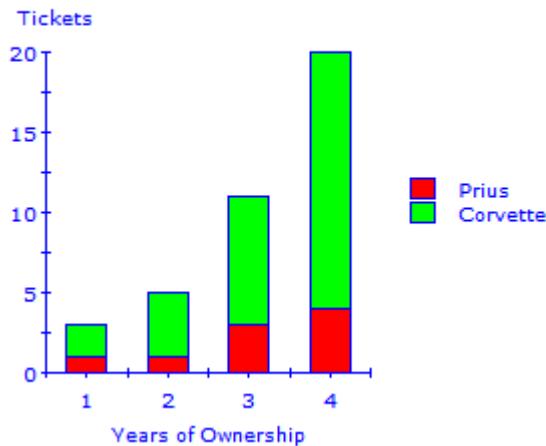
oRadar.SeriesColor(3) = vbGreen
oRadar.SeriesLabel(3) = "Pinto"
```

```
oRadar.AddDataPoint 3, 1, 1, 50
oRadar.AddDataPoint 3, 2, 2, 40
oRadar.AddDataPoint 3, 3, 3, 30
oRadar.AddDataPoint 3, 4, 4, 60
oRadar.AddDataPoint 3, 5, 5, 80

img1.Bitmap = oRadar.Image
End Sub
```

## StackingBar

Speeding Tickets



```
Public Sub DrawStackingBar
    On Error Resume Next
    '
    Dim oSBar As New Chart
    Const sFontName As String = "Verdana"
    Const dFontSize As Double = 6.5

    oSBar.Width = 300
    oSBar.Height = 300
    oSBar.Header = "Speeding Tickets"
    oSBar.Type = chtTypeStackingBar
    oSBar.AxesLabel("X") = "Years of Ownership"
    oSBar.AxesLabel("Y") = "Tickets"
    oSBar.AxesTitleFont "X", sFontName, dFontSize
    oSBar.AxesTitleFont "Y", sFontName, dFontSize
    oSBar.AxesFont "X", sFontName, dFontSize
    oSBar.AxesFont "Y", sFontName, dFontSize
    oSBar.LegendFont sFontName, dFontSize
    oSBar.HeaderFont sFontName, dFontSize
    oSBar.ForeColor = vbBlue

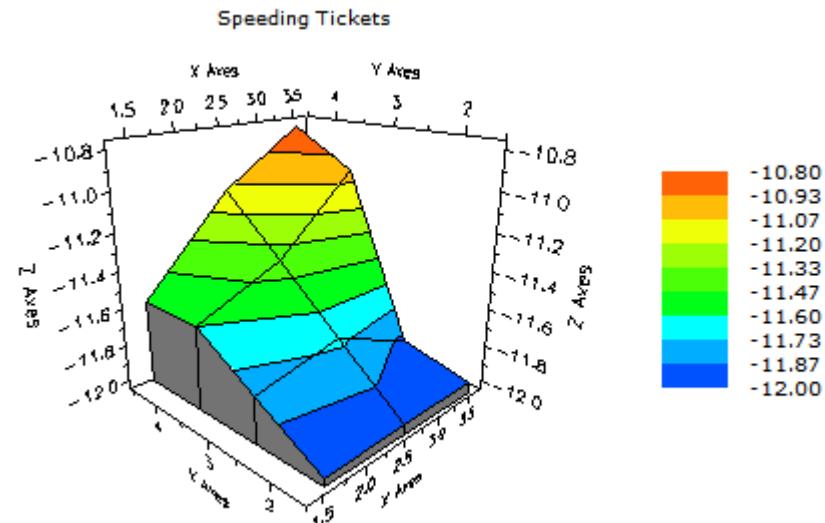
    'Prius
    oSBar.SeriesLabel(1) = "Prius"
    oSBar.SeriesColor(1) = vbRed
    oSBar.AddDataPoint 1, 1, 1, 1
    oSBar.AddDataPoint 1, 2, 2, 1
```

```
oSBar.AddDataPoint 1, 3, 3, 3
oSBar.AddDataPoint 1, 4, 4, 4

'Corvette
oSBar.SeriesColor(2) = vbGreen
oSBar.SeriesLabel(2) = "Corvette"
oSBar.AddDataPoint 2, 1, 1, 2
oSBar.AddDataPoint 2, 2, 2, 4
oSBar.AddDataPoint 2, 3, 3, 8
oSBar.AddDataPoint 2, 4, 4, 16

img1.Bitmap = oSBar.Image
End Sub
```

## Surface



```
Public Sub DrawSurface
    On Error Resume Next
    '
    Dim oSurface As New Chart
    Const sFontName As String = "Verdana"
    Const dFontSize As Double = 6.5

    oSurface.Width = 450
    oSurface.Height= 300
    oSurface.Header = "Speeding Tickets"
    oSurface.Type = chtTypeSurface
    oSurface.LegendFont sFontName, dFontSize
    oSurface.HeaderFont sFontName, dFontSize
    oSurface.AxesLabel("X") = "X Axes"
    oSurface.AxesLabel("Y") = "Y Axes"
    oSurface.AxesLabel("Z") = "Z Axes"

    oSurface.AddDataPoint 1, 1, 1, 1, -12
    oSurface.AddDataPoint 2, 1, 2, 1, -12
    oSurface.AddDataPoint 3, 1, 3, 1, -12

    oSurface.AddDataPoint 1, 2, 1, 2, -11.8
    oSurface.AddDataPoint 2, 2, 2, 2, -11.7
    oSurface.AddDataPoint 3, 2, 3, 2, -11.9

    oSurface.AddDataPoint 1, 3, 1, 3, -11.6
```

```
oSurface.AddDataPoint 2, 3, 2, 3, -11.4
oSurface.AddDataPoint 3, 3, 3, 3, -11

oSurface.AddDataPoint 1, 4, 1, 4, -11.6
oSurface.AddDataPoint 2, 4, 2, 4, -11.1
oSurface.AddDataPoint 3, 4, 3, 4, -10.8

img1.Bitmap = oSurface.Image
End Sub
```

# SearchList Object

A list is a full screen display of selected items; i.e., the data entry device screen is cleared and the contents of the list are displayed. A list box is different in that it displays selected items in the original prompt space allocated for the data in the application.

## SearchList

This object is an advanced method of creating a scrolling list of items that may then be presented to the user for selection.

```
Dim oMyList as New SearchList
```

This object has the following methods:

### AddItem

This method adds an item to the list.

Syntax: oMyList.AddItem(vSelValue, vDispCols)

vSelValue (Variant) The value to be returned when this item is chosen.  
If the user adds the pipe symbol to the selection value variable, it will cause additional columns to be created in the display.

vDispCols (param array of Variants) a comma separated list of values to be displayed in each column

Example:

```
Dim oMyList as New SearchList  
oMyList.AddListEntry(123, "1st Col Description", "2nd  
Col Description")
```

### Cell

This method is used to read or change values or properties of a specific cell within the SearchList object.

Syntax: oMyList.Cell(Row, Col).<method or property>

Row (Long) specifies the row number in the grid

Col (Long) specifies the column number in the grid

Example:

```
oMyList.Cell(2, 2).Value = "1969"
```

### Cell(x,y).BackColor1

This method is used to read or change the primary background color of a specific cell within the SearchList object.

Syntax: oMyList.Cell(x,y).BackColor1 = vValue

Alternate: lValue = oMyList.Cell(x,y).BackColor1

lValue (Long) is the color.

vValue (Variant) sets the color.

Examples:

```
oMyList.Cell(2, 2).BackColor1 = RGB(255,255,0)
```

```
'yellow
```

```
oMyList.Cell(2, 2).BackColor1 = &HFF0000 'blue
```

```
oMyList.Cell(2, 2).BackColor1 = QBColor(5) 'magenta
```

```
oMyList.Cell(2, 2).BackColor1 = vbWhite
```

## Cell(x,y).BackColor2

This method is used to read or change the secondary background color of a specific cell within the SearchList object. It is used to produce gradients from one color to another.

Syntax: oMyList.Cell(x,y).BackColor2 = vValue

Alternate: lValue = oMyList.Cell(x,y).BackColor2

lValue (Long) is the color.

vValue (Variant) sets the color.

Examples:

```
oMyList.Cell(2, 2).BackGradient = GradientVertical
```

```
oMyList.Cell(2, 2).BackColor2 = RGB(255,255,0)
```

```
'yellow
```

```
oMyList.Cell(2, 2).BackColor2 = &HFF0000 'blue
```

```
oMyList.Cell(2, 2).BackColor2 = QBColor(5) 'magenta
```

```
oMyList.Cell(2, 2).BackColor2 = vbWhite
```

## Cell(x,y).BackGradient

This property uses either just the BackColor(1) for solid backgrounds or both BackColor properties to create gradients in one of several directions for a specific cell within the SearchList object.

Syntax: oMyList.Cell(x,y).BackGradient = enValue

enValue (enGradients) enumeration that contains GradientDefault, GradientDiagonalLeft, GradientDiagonalRight, GradientHorizontal, GradientNone, GradientVertical and GradientVerticalSplit

Examples:

```
oMyList.Cell(2, 2).BackColor1 = RGB(0,0,255)  
oMyList.Cell(2, 2).BackColor2 = vbWhite  
oMyList.Cell(2, 2).BackGradient = GradientVertical
```

## Cell(x,y).Bold

This property accesses the prompt's data field bold option for a specific cell within the SearchList object.

Syntax: oMyList.Cell(x,y).Bold = bValue

Alternate: bValue = oMyList.Cell(x,y).Bold

bValue (Boolean) is True or False for bold or not bold.

Examples:

```
oMyList.Cell(2, 2).Bold = True
```

## Cell(x,y).ForeColor

This method is used to read or change the fore color of a specific cell within the SearchList object.

Syntax: oMyList.Cell(x,y).ForeColor = vValue

Alternate: lValue = oMyList.Cell(x,y).ForeColor

lValue (Long) is the color.

vValue (Variant) sets the color.

Examples:

```
oMyList.Cell(2, 2).ForeColor = RGB(255,255,0) 'yellow
```

```
oMyList.Cell(2, 2).ForeColor = &HFF0000 'blue
```

```
oMyList.Cell(2, 2).ForeColor = QBColor(5) 'magenta
```

```
oMyList.Cell(2, 2).ForeColor = vbWhite
```

## Cell(x,y).Italic

This property accesses the prompt's data field italic option for a specific cell within the SearchList object.

Syntax: oMyList.Cell(x,y).Italic = bValue

Alternate: bValue = oMyList.Cell(x,y).Italic

bValue (Boolean) is True or False for italic or not italic.

Examples:

```
oMyList.Cell(2, 2).Italic = True
```

## **Cell(x,y).Underline**

This property accesses the prompt's data field underline option for a specific cell within the SearchList object.

Syntax: oMyList.Cell(x,y).Underline = bValue

Alternate: bValue = oMyList.Cell(x,y).Underline

bValue (Boolean) is True or False for underline or not underline.

Examples:

```
oMyList.Cell(2, 2).Underline = True
```

## **Cell(x,y).Value**

This property accesses the prompt's data field value for a specific cell within the SearchList object.

Syntax: oMyList.Cell(x,y).Value = vValue

Alternate: vValue = oMyList.Cell(x,y).Value

vValue (Variant) is the value within the cell.

Examples:

```
oMyList.Cell(2, 2).Value = "1"  
vValue = oMyList.Cell(2, 2).Value
```

## **Clear**

This method clears the list's contents and optionally clears the column formatting.

Syntax: oMyList.Clear([bClear])

bClear (Boolean) Optional – Set to True to also clear the format of the columns initially set using the SetColumn method. The default is False

Examples:

```
oMyList.Clear  
oMyList.Clear(True)
```

## **Column**

This method is used to read or change properties of a specific column within the Searchlist object.

Syntax: oMyList.Column(Col).<method or property>

Col            (Long) specifies the column number in the Searchlist object

Examples:

```
oMyList.Column(2).Width = 10
```

## Column(x).Align

This property aligns the text of the column within the Searchlist object. The options are either center, left, or right.

Syntax: oMyList.Column(x).Align = enValue

enValue     (enColAlign) an enumeration that contains TextCenter, TextLeft, and TextRight.

Examples:

```
oMyList.Column(2).Align = TextCenter
```

## Column(x).Autosize

This property will size the column based on the widest value in that column.

Syntax: oMyList.Column(x).Autosize = bValue

bValue     (Boolean) set to True or False to autosize the width of the column.

Examples:

```
oMyList.Column(2).Autosize = True
```

## Column(x).BackColor1

This method is used to read or change the primary background color of the whole column within the Searchlist object.

Syntax: oMyList.Column(x).BackColor1 = vValue

Alternate: IValue = oMyList.Column(x).BackColor1

IValue     (Long) is the color.

vValue     (Variant) sets the color.

Examples:

```
oMyList.Column(2).BackColor1 = RGB(255,255,0) 'yellow
```

```
oMyList.Column(2).BackColor1 = &HFF0000                'blue
```

```
oMyList.Column(2).BackColor1 = QBColor(5)              'magenta
```

```
oMyList.Column(2).BackColor1 = vbWhite
```

## **Column(x).BackColor2**

This method is used to read or change the secondary background color of the whole column within the Searchlist object. It is used to produce gradients from one color to another.

Syntax: oMyList.Column(x).BackColor2 = vValue

Alternate: lValue = oMyList.Column(x).BackColor2

lValue (Long) is the color.

vValue (Variant) sets the color.

Examples:

```
oMyList.Column(2).BackGradient = GradientVertical  
oMyList.Column(2).BackColor2 = RGB(255,255,0) 'yellow  
oMyList.Column(2).BackColor2 = &HFF0000 'blue  
oMyList.Column(2).BackColor2 = QBColor(5) 'magenta  
oMyList.Column(2).BackColor2 = vbWhite
```

## **Column(x).BackGradient**

This property uses either just the BackColor(1) for solid backgrounds or both BackColor properties to create gradients in one of several directions for the whole column within the Searchlist object.

Syntax: oMyList.Column(x).BackGradient = enValue

enValue (enGradients) enumeration that contains GradientDefault, GradientDiagonalLeft, GradientDiagonalRight, GradientHorizontal, GradientNone, GradientVertical and GradientVerticalSplit

Examples:

```
oMyList.Column(2).BackColor1 = RGB(0,0,255)
```

```
oMyList.Column(2).BackColor2 = vbWhite
```

```
oMyList.Column(2).BackGradient = GradientVertical
```

## **Column(x).Bold**

This property accesses the column's bold option for the whole column within the Searchlist object.

Syntax: oMyList.Column(x).Bold = bValue

Alternate: bValue = oMyList.Column(x).Bold

bValue (Boolean) is True or False for bold or not bold.

Examples:

```
oMyList.Column(2).Bold = True
```

## **Column(x).Caption**

This property accesses the caption associated with the specified column.

Syntax: oMyList.Column(x).Caption = vValue

Alternate: sValue = oMyList.Column(x).Caption

sValue (String) is the title of the column.

vValue (Variant) sets the title of the column.

Examples:

```
oMyList.Column(2).Caption = "Model"
```

## **Column(x).DisplayOnly**

This property accesses the display only property associated with the specified column. Setting it to True makes all the cells in that column unchangeable.

Syntax: oMyList.Column(x).DisplayOnly = bValue

Alternate: bValue = oMyList.Column(x).DisplayOnly

bValue (Boolean) is the display only state for the prompt.

Examples:

```
oMyList.Column(2).DisplayOnly = True
```

## **Column(x).ForeColor**

This property accesses the column's fore color property associated with the specified column.

Syntax: oMyList.Column(x).ForeColor = lValue

Alternate: vValue = oMyList.Column(x).ForeColor

vValue (Variant) is the color.

lValue (Long) sets the color.

Examples:

```
oMyList.Column(2).ForeColor = RGB(255,255,0) 'yellow
```

```
oMyList.Column(2).ForeColor = &HFF0000 'blue
```

```
oMyList.Column(2).ForeColor = QBColor(5) 'magenta
```

```
oMyList.Column(2).ForeColor = vbWhite
```

## **Column(x).Format**

This property affects the format of the whole specified column. It is only an extension of the Format VBA command.

Syntax: oMyList.Column(x).Format = sValue

Alternate: sValue = oMyList.Column(x).Format

sValue (String) is the format mask to use when displaying data for the prompt.

Examples:

`oMyList.Column(2).Format = "hh:mm"`

c - General Date

dddddd - Long Date

ddddd - Short Date

tttt - Long Time

hh:mm AMPM - Medium Time

hh:mm - Short Time

\$#,##0.00 or (\$#,##0.00) - Currency 0.00 - Fixed

#,##0.00 - Standard 0.00% - Percent 0.00E+00 -

Scientific

Yes/No - Return "No" if zero, else return "Yes"

True/False - Return "True" if zero, else return "False"

On/Off - Return "On" if zero, else return "Off"

*For further examples get help on the VB FORMAT command.*

## **Column(x).ImageMode**

This property formats images placed in the specified column to one mode option. The options are: BottomCenter, BottomLeft, BottomRight, CenterCenter, CenterLeft, CenterRight, TopCenter, TopLeft, TopRight, Default, Stretched, Disabled and Tile.

Syntax: oMyList.Column(x).ImageMode(enVal)

Alternate: enVal = oMyList.Column(x).ImageMode

enVal (enImageAlign) is the alignment style for the image

Example:

`oMyList.Column(1).ImageMode = ImageAlignTopLeft`

## **Column(x).Italic**

This property accesses the column's italic option for the whole column within the Searchlist object.

Syntax: oMyList.Column(x).Italic = bValue

Alternate: bValue = oMyList.Column(x).Italic

bValue (Boolean) is True or False for italic or not italic.

Examples:

```
oMyList.Column(2).Italic = True
```

## **Column(x).ScaleDecimals**

This property formats the numeric values in the specified column if the database does not store decimals. This option will position a decimal at a position from the right side. A comma will be used for large numbers. This field is locale specific and will use the appropriate characters for each region.

Syntax: oMyList.Column(x).ScaleDecimals = vValue

Alternate: vValue = oMyList.Column(x).ScaleDecimals

vValue (Variant) is the decimal position.

Examples:

```
oMyList.Column(1).ScaleDecimals = 2
```

## **Column(x).TrimSpaces**

This property formats the values in the specified column by deleting the leading and trailing spaces in the data.

Syntax: oMyList.Column(x).TrimSpaces = bValue

Alternate: bValue = oMyList.Column(x).TrimSpaces

bValue (Boolean) set to True to trim all space from the data.

Examples:

```
oMyList.Column(1).TrimSpaces = True
```

## **Column(x).Underline**

This property accesses the column's underline option for the whole column within the Searchlist object.

Syntax: oMyList.Column(x).Underline = bValue

Alternate: bValue = oMyList.Column(x).Underline

bValue (Boolean) is True or False for underline or not underline.

Examples:

```
oMyList.Column(2).Underline = True
```

## **Column(x).Visible**

This property makes visible or invisible the whole column within the Searchlist object.

Syntax: oMyList.Column(x).Visible = bValue

Alternate: bValue = oMyList.Column(x).Visible

bValue (Boolean) is True or False for column visibility.

Examples:

```
oMyList.Column(2).Visible = True
```

## **Column(x).Width**

This property sets or returns the width of the column within the Searchlist object.

Syntax: oMyList.Column(x).Width = vValue

Alternate: vValue = oMyList.Column(x).Visible

vValue (Variant) sets or gets the width of the specified column.

Examples:

```
oMyList.Column(2).Width = 25
```

## **Columns**

This property returns the number of columns in the Searchlist object.

Syntax: vValue = oMyList.Columns

vValue (Variant) gets the number of columns in the Searchlist object.

Examples:

```
Dim iCnt As Integer  
iCnt = oMyList.Columns
```

## **Count**

This function returns the number of items in the Searchlist object.

Syntax: vValue = oMyList.Count

vValue (Variant) the number of items in the list

Example:

```
Dim nValue As Long  
nValue = oMyList.Count
```

## Index

This property returns or sets the current list index property.

Syntax: oMyList.Index = lValue

Alternate: vValue = oMyList.Index

lValue (Long) sets the item index in the list

vValue (Variant) gets the item index in the list

Examples:

```
Dim nValue As Long  
nValue = oMyList.Index  
oMyList.Index = 5
```

## List

The SearchList object can be used to generate the formatted list and then output it to a prompt's list property (like a Listbox or Combobox) or be used in methods like App.ShowList.

Syntax: sList = oMyList.List

sList (String) contains the formatted array of values for use in other methods

Examples:

```
oMyList.MaxRows = 100  
oMyList.ReturnAllRows = False  
oMyList.ShowEmptyList = False  
oMyList.SQL = "select * from PLANTS"  
oMyList.SetColumn 1, "ID", 5, TextLeft, True  
oMyList.SetColumn 2, "NAME", 21, TextLeft, True  
  
cboPlants.List.Data = oMyList.List  
or  
Rsp = App.ShowList(oMyList.List)
```

## MaxRows

This property limits how many rows will be allowed in the list. If the database will return several thousand records, you may want to limit this list to 500. An alternative is to further restrict the search criteria if using SQL.

Syntax: oMyList.MaxRows = nNum

Alternate: nNum = oMyList.MaxRows

nNum (Long) is a numeric value to limit the rows in the list

Example:

```
oMyList.MaxRows = 100
```

## Normalize

This property, when set to True, will trim the preceding and trailing spaces from the column data in the list. This should only be necessary if the database stores space-padded data values.

Syntax: oMyList.Normalize = bVal

Alternate: bVal = oMyList.Normalize

bVal (Boolean) set to True to trim all data in the list

Examples:

```
Dim bValue As Boolean  
oMyList.Normalize = True  
bValue = oMyList.Normalize
```

## ReturnAllRows

This property when set to True instructs the list to return all the values for all the columns instead of the first value of the first column when a row is selected.

Syntax: oMyList.ReturnAllRows = bValue

Alternate: bValue = oMyList.ReturnAllRows

bValue (Boolean) a True or False value (*default = False*)

Examples:

```
Dim bValue As Boolean  
bValue = oMyList.ReturnAllRows  
oMyList.ReturnAllRows = True
```

## SetColumn

This method formats the returned data to fit the device's screen and desired layout. This statement should be used for each column to be displayed.

Syntax: oMyList.SetColumn(nColumn, sHeading, vDefWidth, [enAlign],  
[bTrimSpaces], [nDecimals])

nColumn (Long) the column number to be affected by the  
formatting

sHeading (String) the title that will appear across the top of the  
column

vDefWidth	(Variant) the width of the column in pixels or characters based on the Environment Properties
enAlign	(enColAlign) Optional – how to align the column; either Left, Right or Center; the default is Left justified
bTrimSpaces	(Boolean) Optional – True means that leading and trailing spaces will be removed from the display of the data in this column
nDecimals	(Long) Optional – if the value is numeric and the database does not store decimals, use this to position a decimal at a position from the right side. A comma will be used for large numbers. This field is locale specific and will use the appropriate characters for each region.

Example:

```
oMyList.oList.MaxRows = 100
oMyList.ReturnAllRows = False
oMyList.ShowEmptyList = False
oMyList.Normalize = True
oMyList.SQL = "select * from PLANTS"
oMyList.SetColumn 1, "ID", 5, TextLeft, True
oMyList.SetColumn 2, "NAME", 21, TextLeft, True
sName = oMyList.ShowList
```

## ShowEmptyList

This property displays the blank list to the user even when there are no entries. The whole screen will be temporarily cleared and the user must press enter to acknowledge there were no entries.

Syntax: oMyList.ShowEmptyList = bValue

Alternate: bValue = oMyList.ShowEmptyList

bValue      (Boolean) is either True or False.

Example:

```
oMyList.ShowEmptyList = True
```

## ShowList

This function displays the list to the user. The whole screen will be temporarily cleared and the list will be displayed until a choice is made.

Syntax: sValue = oMyList.ShowList

sValue      (String) is a Chr(1) delimited list of the values in the columns based on the row selected.

Example:

```
Dim sValue as String
```

```
sValue = oMyList.ShowList
```

## SQL

This property will contain the SQL statement to be used in creating the list.

Syntax: oMyList.SQL = sSQL

Alternate: sSQL = oMyList.SQL

sSQL (String) the SQL statement to be executed

Example:

```
Dim sSQL As String  
sSQL = "Select * from ItemMaster"  
oMyList.SQL = sSQL
```

## Value(x)

This property returns the value in the specified row for the Searchlist.

Syntax: vValue = oMyList.Value(Row)

Row (Long) is the row number in the list

vValue (Variant) is given the value assigned to the row

Examples:

```
Dim vValue As Variant  
vValue = oMyList.Value(1)
```

# Embedded Procedure Object

The following section describes:

- Embedded Procedures – ERP and other stored procedures, macros, etc. that allow functions to be executed without using a front-end interface
- Properties and Methods – The properties of the function are the values sent and received from the backend system and the methods are commands used to manipulate the function such as 'execute' or 'clear' that operate on the object itself.

## Embedded Procedures

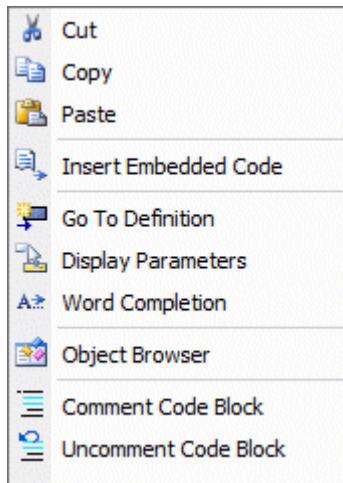
An Embedded Procedure refers to embedding VBA code that performs a previously created task. The user may generate code for:

- Business Functions – typically used with ERP systems
- Business Objects – an object variable used by Microsoft Axapta
- Database Table – used to generate Insert SQL statements
- Stored Procedures – found in ODBC compliant databases

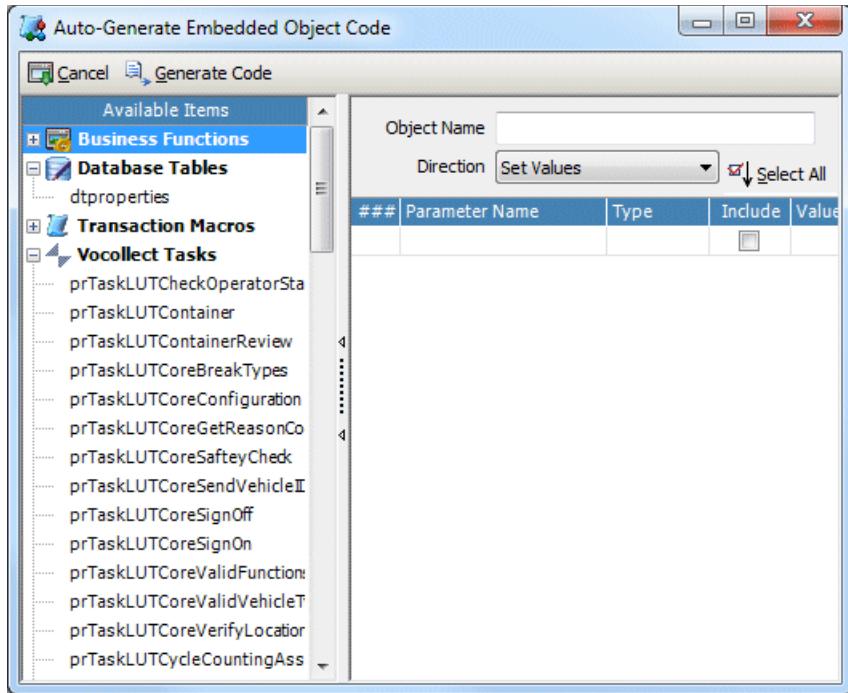
- Transaction Macros – to be queued or executed
- Vocollect Recordsets – defined Vocollect resources
- Web Service Objects – based on the Web Service Connector

Embedded Procedures are intended to be an automated approach to writing VBA code that specifies parameters and then executes the procedure. Although you can write the code yourself, this product has a code generator that makes this process much easier.

In the VBA environment right-click and select “Insert Embedded Code”.

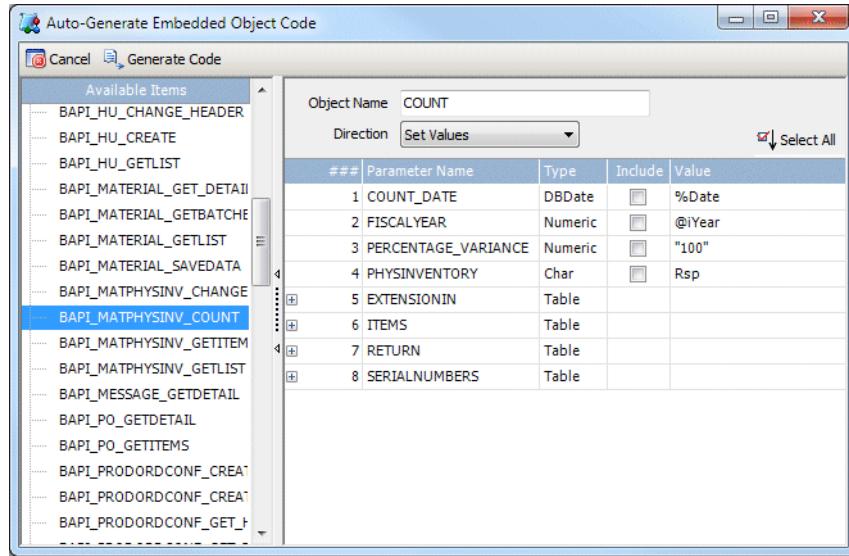


Next, select the type of object to embed from the list on the left.



A list is shown of all previously downloaded object of that type. If the item does not appear in the list, it must first be downloaded from the backend data source.

To select the desired business function double-click on the listed function. Alter any default values that may be required. Any default values that were assigned after downloading the function (under the menu item Data Link – Edit Tables / Business Functions) will appear blue. These default values will be assumed when the function executes. You can at this time override the default value, which will then turn red if it indicates that a default was changed. Any entries that do not have a value will remain black and are only used for the immediate pasting of the Embedded Procedure code. The value column is the only editable field.



When the code is generated, only the lines with non-default values are displayed. The server knows to send all parameters to the function, but unless there is a non-default value, the code window is not needlessly filled with every available parameter. A basic embedded procedure will look similar to the following:

```

Dim iYear As Integer
Dim Rsp As Variant
Dim emCOUNT As New EmbeddedProc
,
emCOUNT.Name = "BAPI_MATPHYSINV_COUNT"
emCOUNT.DataSource = "SAP"
,
emCOUNT.Param("COUNT_DATE") = App.GetValue("Date")
emCOUNT.Param("FISCALYEAR") = iYear
emCOUNT.Param("PERCENTAGE_VARIANCE") = "100"
emCOUNT.Param("PHYSINVENTORY") = Rsp
,
If Not emCOUNT.Execute Then
End If

```

As you can see, the parameters are filled in with the default values. If you choose, you could replace "2011" with RFPrompt("Year").Text.

There are several options when filling in the Value column. When only an '@' symbol is used, the name of the parameter itself will be declared as a variable to be used or replaced later. If a value is wrapped with

double quotes it will be placed in the code as a literal like the “2011” value in the above example. If any string begins with a percent sign the value will be used inside an App.GetValue or App.SetValue command. The Hungarian notation for Integer ( i ), Long ( n ), Boolean ( b ), Variant ( v ) and String ( s ) placed in front of the value will ensure the variable declaration as that type. Variant is the default type if the system cannot determine the type or a standard variable name is used. Any standard string by itself will be made into a variable.

## Embedded Procedure Properties and Methods

The following list of properties and methods are used to setup a procedure and / or be evaluated after the procedure executes.

### ClassObject

When using Microsoft ERP systems, there are functions that can be called that return class objects. This property can then be set to that returned object and methods on that object can then be called.

Syntax: emProc.ClassObject = oClass

oClass (object) the name of the class object required by the method.

Example:

```
emProc.ClassObject = oMyClassObject
```

### Clear

After the programmer or the Execute method has altered parameters, this command clears all parameters so it may be used again. This only effects the emProc.Param( ) values.

Syntax: emProc.Clear

### ColumnCount

For a given table, this function will contain the number of parameters or columns as they appear when downloaded.

Syntax: vValue = emProc.ColumnCount(vParamId)

vValue (Variant) contains the number of columns or parameters in the specified table.

vParamId (Variant) the name of the table usually referred to by name and in double quotes.

Example:

```
Dim nValue As Long
```

```
nValue = emProc.ColumnCount ("RETURN")
```

## ColumnName

For a given table or function, this function will display the name of a parameter when referenced by its index value. If there are no tables use the property ParamName.

Syntax: sValue = emProc.ColumnName(vTable, [nIndex])

sValue (String) contains the name of the parameter

vTable (Variant) the table or function name that contains parameters

nIndex (Long) Optional – the number of the parameter for which the name is retrieved. Left off and the complete list will be returned.

Example:

*If the embedded procedure had these parameters declared:*

```
emProc.Param("PLANT", "SIGN")      = "I"
```

```
emProc.Param("PLANT", "OPTION")     = "EQ"
```

```
emProc.Param("PLANT", "PLANT_LOW")  = "3000"
```

then emProc.ColumnName("PLANT", 3) would return "PLANT\_LOW"

## DataSource

This property indicates the name of the data source to connect to. This is generally setup in the HOSTS file assigning an IP address to a name.

Syntax: emProc.DataSource = sValue

sValue (String) name of the data source

Example:

```
emProc.DataSource = "RFSample"
```

## DebugLog

This property contains the path to the log file. The log file will contain all function calls by appending to this file.

Syntax: emProc.DebugLog = sValue

Alternate: sValue = emProc.DebugLog

sValue (String) the path to a text file

Example:

```
emProc.DebugLog = "\Program Files\Status.txt"
```

## DisableParam

Only used for SAP connectivity, this method allows the user to request that SAP not send data back in tables that will not be used. If a BAPI returns 5 tables of data but only 1 will be used, the other 4 can be turned off to increase the speed SAP returns with results.

Syntax: emProc.DisableParam(vParamID, bDisabled)

vParamID (Variant) the name of the table that is not necessary

bDisabled (Boolean) set to True if the table should be ignored.

## Execute

After all values are correctly set, this command sends the values to the ERP system for evaluation. The function being executed is contained in the emProc.Name property. The properties of the procedure are then filled in with the results.

Syntax: [bValue =] emProc.Execute

bValue (Boolean) Optional – contains a True / False based upon its success.

## ExecuteMethod

For Microsoft ERP systems, the user may specify a specific method to be executed. After all values are correctly set, this command sends the values to the ERP system for evaluation. The function being executed is contained in the emProc.Name property. The properties of the procedure are then filled in with the results.

Syntax: [bValue =] emProc.ExecuteMethod(sName)

bValue (Boolean) Optional – contains a True / False based upon its success.

sName (String) the name of the method to execute

## LogMode

This property can be set to log nothing, everything or just errors as it relates to executing business functions, stored procedures or macros. All messages are written to the error log.

Syntax: emProc.LogMode = enValue

Alternate: enValue = emProc.LogMode

enValue (enLogMode) an enumeration containing 3 possible values:

LogNever – nothing is written to the log

LogAlways – all data, successes and failures are logged

LogFailure – only embedded procedure failures are logged

Example:

`emProc.LogMode = LogAlways`

## Name

This property contains the function name to be executed.

Syntax: emProc.Name = sValue

Alternate: sValue = emProc.Name

sValue (String) the name of the business function

Example:

```
emProc.Name = "BAPI_GOODSMVT_CREATE"
```

## Param

This property controls all the business function's parameter values.

Syntax: emProc.Param(vParamId, [vColName], [nRowNo]) = vValue

Alternate: vValue = emProc.Param(vParamId, [vColName], [nRowNo])

vValue (Variant) the result of the parameter's value or the value being placed in the parameter

vParamId (Variant) the name of the table usually referred to by name and in double quotes.

vColName (Variant) Optional – the name of the column within a table parameter

nRowNo (Long) Optional – the row number within a table parameter

Examples:

```
emProc.Param("YEAR") = "2013"
```

- or -

```
Dim vValue As Variant  
vValue = emProc.Param("RESULTS", "ID", 1)
```

## ParamCount

For a given business function, this function will contain the number of parameters as they appear when downloaded. Each table will count as 1 parameter.

Syntax: vValue = emProc.ParamCount

vValue (Variant) contains the number of parameters for the specified business function.

Example:

```
Dim nValue as Long  
nValue = emProc.ParamCount
```

## ParamEx

For SAP systems only, this property controls all the business function's parameter values and allows for nested parameters, like a table parameter that contains another table.

Syntax: emProc.ParamEx(vParamId, vColName, [nRowNo]) = enValue

Alternate: enValue = emProc.ParamEx(vParamId, vColName, [nRowNo])

enValue (EmbeddedParam) the result of the parameter's value or the value being placed in the parameter

vParamId (Variant) the name of the table usually referred to by name and in double quotes.

vColName (Variant) the name of the column within a table parameter

nRowNo (Long) Optional – the row number within a table parameter

Examples:

```
Dim sError As Variant  
sError = emProc.ParamEx("RETURN", "MESSAGE", 1)
```

In the case of nested parameters, specify the parameter ID and the column that contains the nested table and then use “dot” notation to extend the statement.

```
Dim vValue As Variant  
vValue = emProc.ParamEx("ParamID",  
"Col1").Param("SubParam", "SubCol")
```

This notation can be used indefinitely to set or obtain data from structures within other structures. The properties available after the ParamEx property are:

```
emParam.ParamEx("ParamID", "Col").ColumnCount  
emParam.ParamEx("ParamID", "Col").ColumnName  
emParam.ParamEx("ParamID", "Col").Param  
emParam.ParamEx("ParamID", "Col").ParamEx  
emParam.ParamEx("ParamID", "Col").RowCount
```

## ParamName

For a given function, this function will display the name of a parameter when referenced by its index value. If there are tables that contain parameters, use the property ColumnName.

Syntax: sValue = emProc.ParamName([nIndex])

sValue (String) contains the name of the parameter

nIndex (Long) Optional – the number of the parameter for which the name is retrieved. Left off a full list is returned.

## **Queue**

This function returns a Boolean value depending on if the transaction could be queued. This is in place of executing the business function. The Transaction Manager will add this business function to the queue and execute it when the host is available and when it gets to the top of the queue. (Also see QueueSeqNo)

Syntax: bValue = emProc.Queue

bValue (Boolean) contains a True if the transaction was successfully queued.

Example:

```
Dim bValue as Boolean  
bValue = emProc.Queue
```

## **QueueName**

Multiple queues are allowed in the transaction manager process (as configured in Configure \ System Options \ Service Options.) This property will return the name of the queue currently in use. It can also be used to set which queue processes the transaction.

Syntax: sValue = emProc.QueueName

Alternate: emProc.QueueName = sValue

sValue (String) contains the name of the queue

Examples:

```
Dim sValue as String  
sValue = emProc.QueueName  
- or -  
emProc.QueueName = "AltQueue"
```

## **QueueOffline**

This property sets or returns whether the embedded procedure is set to queue if the host is offline.

Syntax: bValue = emProc.QueueOffline

Alternate: emProc.QueueOffline = bValue

bValue (Boolean) contains a True or False depending on if the transaction can be or should be queued.

Examples:

```
Dim bValue as Boolean  
bValue = emProc.QueueOffline  
- or -
```

```
emProc.QueueOffline = True
```

## QueueSeqNo

This property returns the sequence number given to the queued function using the emProc.Queue method.

Syntax: nValue = emProc.QueueSeqNo

nValue (Long) contains the sequence number generated by the Transaction Manager when the function was queued.

Example:

```
Dim nValue As Long  
emProc.Queue  
nValue = emProc.QueueSeqNo
```

## RowCount

For a given table, this function will contain the number of rows returned from the function given the passed parameters.

Syntax: vValue = emProc.RowCount(vParamId)

vValue (Variant) contains the number of rows in the specified table.

vParamId (Variant) the name of the table, usually referred to by name and in double quotes.

Example:

```
Dim nValue as Long  
nValue = emProc.RowCount ("RETURN")
```

## DataRecord Object

This object gives the user information about a stored recordset populated by other language extensions.

To use this object, start with a declaration similar to:

```
Dim oData as New DataRecord
```

### AddNew

This method creates an additional entry in the DataRecord. As an example, when the TM.GetItems method populates a DataRecord from a list of items in a queue, this method is used internally to grow the DataRecord until the list is complete. If this object is being used for other purposes the AddNew method may be used to grow the DataRecord as needed.

Syntax: oData.AddNew

## **Clear**

This method clears the object of any previously loaded information

Syntax: oData.Clear

## **IsEOF**

This method (End-Of-File) returns a True if the current pointer in the recordset is beyond the last entry or if there are no entries contained in the object.

Syntax: [bOK =] oData.IsEOF

bOK (Boolean) Optional – returns the True or False

## **MoveFirst**

This selects (moves the pointer to) the first entry in the object's list of values.

Syntax: [bOK =] oData.MoveFirst

bOK (Boolean) Optional – returns True or False for the success of the command

## **MoveLast**

This selects (moves the pointer to) the last entry in the object's list of values.

Syntax: [bOK =] oData.MoveLast

bOK (Boolean) Optional – returns True or False for the success of the command

## **MoveNext**

This selects (moves the pointer to) the next entry in the object's list of values.

Syntax: [bOK =] oData.MoveNext

bOK (Boolean) Optional – returns True or False for the success of the command

## **MovePrevious**

This selects (moves the pointer to) the previous entry in the object's list of values.

Syntax: [bOK =] oData.MovePrevious

bOK (Boolean) Optional – returns True or False for the success of the command

## MoveTo

This selects (moves the pointer to) a specific entry in the recordset.

Syntax: [bOK =] oData.MoveTo(nRow)

bOK (Boolean) Optional – returns True or False for the command's success

nRow (Long) the row number to move the object's pointer

## Param

This property returns the values stored in the named columns of the DataRecord.

Syntax: Param(vParamID, [vRowNo]) = vValue

Alternate: vValue = Param(vParamID, [vRowNo])

vValue (Variant) the stored value given a parameter ID

vParamID (Variant) then name of a column in the DataRecord

vRowNo (Variant) Optional – if the DataRecord is a table containing multiple rows of data, this parameter will move the data pointer to the specified row before retrieving the data.

Examples:

```
Dim vData As Variant  
oData.Param("ErrMsg") = "Wrong value."  
vData = oData.Param("DeviceNo")  
vData = oData.Param("FormId")  
vData = oData.Param("IPAddress")  
vData = oData.Param("ExecDate")
```

## ParamCount

For a given row that is a table, this function will contain the number of columns returned.

Syntax: vValue = oData.ParamCount

vValue (Variant) contains the count of the columns in the DataRecord

Example:

For example, a table named Inventory has 2 fields, Part and Quantity, and has 200 records.

```
Dim iValue As Integer
```

```
iValue = oData.ParamCount      ' will return 2.
```

## ParamName

This property is a collection of parameter IDs. In the case of the TM.GetItems use of the DataRecord object the parameter IDs are:

(1) QueueName	(5) Source	(9) UserId	(13) ExecTime
(2) SeqNo	(6) Name	(10) DeviceNo	(14) Status
(3) TranDate	(7) Record	(11) IPAddress	(15) ErrMsg
(4) TranTime	(8) FormId	(12) ExecDate	

If the DataRecord object is used for another purpose the parameter names would be different. See the ParamCount property to get a count of the parameter IDs.

Syntax: sValue = oData.ParamName([nIndex])

sValue (String) the name of the parameter at the specified index

nIndex (Long) Optional – the index of the parameter to be evaluated

Examples:

When using the TM.GetItems method, to retrieve the name of the macro for the first row in the DataRecord:

```
Dim oData As New DataRecord
Dim sMacroName As String
TM.GetItems(tmCompleted, Date, App.User, oData)
oData.MoveFirst
sMacroName = oData.Param(6)
sMacroName = oData.Param("Name")
```

## RowCount

For a given record set, this function will get the number of rows returned in the object.

Syntax: vValue = oData.RowCount

vValue (Variant) the number of rows in the DataRecord

Example:

When using the TM.GetItems method, the number of queue entries returned would be returned.

```
Dim oData As New DataRecord
Dim iValue As Integer
TM.GetItems(tmCompleted, Date, App.User, oData)
iValue = oData.RowCount
```

## **Schemaid**

This property returns the Task ID of a Vocollect task that is currently loaded in the DataRecord.

Syntax: sValue = oData.Schemaid  
sValue (String) the task Id of a Vocollect task

```
Dim sValue As String  
sValue = oData.SchemaID
```

## **Dynamic Array Extensions**

This product supports a special kind of string variable called a 'dynamic array'. The word 'dynamic' means that this type of variable is designed to allow stored data to grow or shrink in size. In addition, the stored data is easily accessible and changeable. Dynamic arrays allow volumes of data to be organized, stored, referenced, counted, and manipulated by use of just 1 single string variable. Access to the data is instantaneous via the Dynamic Array VBA extensions that follow.

### **Dynamic Array Structure**

Dynamic Arrays are special string variables that use low end ANSI characters (1, 2, and 3) as 'delimiters' to separate data into Fields using Chr(1), Subfields using Chr(2), and Sub-subfields using Chr(3). These delimiters were chosen because data being entered or scanned are never expected to contain them. To use Dynamic Arrays in your programming, you do not need to reference these characters, the server simply uses them internally to manage stored data.

### **Why Use Dynamic Arrays?**

1. There are no limits to the amount of data that may be stored
2. The alternative is to declare and use subscripted variables, the number of which may be insufficient
3. The server has been optimized to access and manipulate this type of data instantaneously.

For illustrative purposes, the 'Field' and 'Subfield' delimiters are represented in the examples below as follows:

Dynamic Array Field delimiter (Chr(1)) as a '&'  
Dynamic Array Subfield delimiter (Chr(2)) as a '#'  
Dynamic Array Sub-Subfield delimiter (Chr(3)) as a '@'

## **DCount**

The DCount function (Delimiter Count) may be used to determine the number of occurrences of a specified delimiter within a string, or a string variable. The command will also add 1 to the count to actually represent the number of values or fields stored within the string variable. This makes the DCount command really return the number of available fields in the record so that extra logic is not necessary when determining the length of a loop structure used to manipulate the contents of the string.

Syntax: vNbr = DCount(vVAR, vDELIM)

vNbr      (Variant) is a number of delimiters found in the variable plus 1.  
vVAR      (Variant) is the string value to be evaluated  
vDELIM     (Variant) is a specified delimiter character, or multi-character string to count; e.g., to count dynamic array delimiters use Chr(1), Chr(2), Chr(3).

Examples:

```
Dim nNbr As Long
Dim sNames As String
sNames = "John&Mike&Albert"
nNbr = DCount(sNames,Chr(1)) ' Nbr = 3

sNames = "&Mike&Albert"
nNbr = DCount(sNames,Chr(1)) ' Nbr = 3

sNames = "&Mike&"
nNbr = DCount(sNames,Chr(1)) ' Nbr = 3
```

Note that +1 has been added to the actual count of Chr(1) (represented here with a character '&') count of 2. Also note that just because the fields are empty, they are valid slots where data may be inserted.

## Del

The Delete function deletes a field, subfield, or sub-subfield from a dynamic array.

Syntax: vREC = Del(vREC, vFLD, [vSUB], [vSSUB])  
vREC    (Variant) is the dynamic array variable name  
vFLD    (Variant) is the array field number to search  
vSUB    (Variant) Optional – is the array subfield number  
vSSUB   (Variant) Optional – is the array sub-subfield number

Examples:

```
Dim sNames As String
sNames = "John&Mike&Albert" ' Three records of names
```

```

sNames = Del(sNames, 3)           ' sNames becomes
"John&Mike"

sNames = "John&Mike&Albert"    ' Three records of names
sNames = Del(sNames, 2)           ' sNames becomes
"John&Albert"

```

This next example shows each record with 3 values; Name, Age and Language.

```

sNames =
"John#31#English&Mike#34#Spanish&Albert#40#English"
sNames = Del(sNames, 1)
'sNames becomes
"Mike#34#Spanish&Albert#40#English"

sNames =
"John#31#English&Mike#34#Spanish&Albert#40#English"
sNames = Del(sNames, 2)
'sNames becomes
"John#31#English&Albert#40#English"

```

You can also delete portions of a record.

```

sNames =
"John#31#English&Mike#34#Spanish&Albert#40#English"
sNames = Del(sNames, 2, 1)
'sNames becomes
"John#31#English&34#Spanish&Albert#40#English"

sNames =
"John#31#English&Mike#34#Spanish&Albert#40#English"
sNames = Del(sNames, 2, 2)
'sNames is now
"John#31#English&Mike#Spanish&Albert#40#English"

sNames =
"John#31#English&Mike#34#Spanish&Albert#40#English"
sNames = Del(sNames, 2, 3)
'sNames becomes
"John#31#English&Mike#34&Albert#40#English"

```

Using the Sub-subfield concept, maybe the language has sub categories like Canadian English and United Kingdom English.

```

sNames = "John#31#Canadian English@UnitedKingdom
English&Mike...
sNames = Del(sNames, 1, 3, 1)
    'sNames becomes "John#31# UnitedKingdom
English&Mike...

```

Here everything between the Chr(1) delimiters is considered the first field. The 2 versions of English are a part of the third sub-field. So reading the DEL statement: "Using the first field, look at the third sub-field (all the languages) and delete the first sub-subfield.

A Dynamic Array is like a database stored as a single string. Everything between the Chr(1) delimiters are fields referenced by the first number. If there are any Chr(2) delimiters, each piece makes up the complete record. If there are any Chr(3) delimiters, they make up the complete subfield.

## **Ext**

The Extract function returns a field, subfield, or sub-subfield value from a dynamic array.

Syntax: vVAL = Ext(vREC, vFLD, [vSUB], [vSSUB])  
vVAL (Variant) is the string value extracted from the array.  
vREC (Variant) is the dynamic array variable name  
vFLD (Variant) is the array field number to search  
vSUB (Variant) Optional – is the array subfield number  
vSSUB (Variant) Optional – is the array sub-subfield number

Examples:

```

Dim sNames As String
Dim sValue As String
sNames = "John&Mike&Albert" ' Three records of names
sValue = Ext(sNames, 2)      'sValue becomes "Mike"

sNames = "John&Mike&Albert" ' Three records of names
sValue = Ext(sNames, 3)      'sValue becomes "Albert"

```

This next example shows each record with 3 values; Name, Age and Language.

```

sNames =
"John#31#English&Mike#34#Spanish&Albert#40#English"
sValue = Ext(sNames, 2, 3) 'sValue becomes "Spanish"

```

Here Mike#34#Spanish is the second record and the third subfield value was retrieved.

Using the Sub-subfield concept, maybe the language has sub categories like Canadian English and United Kingdom English.

```
sNames = "John#31#Canadian English@UnitedKingdom  
English&Mike...  
sValue = Ext(sNames, 1, 3, 1)    'sValue becomes  
"Canadian English"
```

There are times when the Extract function can be very helpful. For example, if you create a comma-delimited list of items and must parse through it, writing code to manipulate the string and keep track of the pointer can be extensive. Instead use a statement like:

```
sList = Replace(sList, ", ", Chr(1))
```

This replaces the comma with the Chr(1) character. Then in the loop you only need to EXT(sList, i) to get each entry out. Use the DCount command to get the length of the list.

## FixLeft

This function pads a raw string to the left with a specified character until the specified length is reached. If the optional 3<sup>rd</sup> parameter is not specified, spaces are used. (This is not specifically a Dynamic Array command but is commonly used to manipulate them.)

Syntax: vValue = FixLeft(vStringData, vChars, [vPadChar])

vValue       (Variant) is the padded string value

vStringData   (Variant) is the string containing the raw data

vChars       (Variant) is the total size of the resulting string

vPadChar      (Variant) Optional – is the pad character to use. If none is specified, spaces are used

Example:

```
Rsp = FixLeft(Rsp, 8, "A")
```

If Rsp was “123” then Rsp becomes “123AAAAA” because it puts the character ‘A’ as many times as necessary after the Rsp value until the length of Rsp is 8, left justifying the StringData value.

## FixRight

This function pads a raw string to the right with a specified character until the specified length is reached. If the optional 3<sup>rd</sup> parameter is not

specified, spaces are used. (This is not specifically a Dynamic Array command but is commonly used to manipulate them.)

Syntax: vValue = FixRight(vStringData, vChars, [vPadChar])

vValue      (Variant) is the padded string value

vStringData    (Variant) is the string containing the raw data

vChars      (Variant) is the total size of the resulting string

vPadChar    (Variant) Optional – is the pad character to use. If none is specified, spaces are used

Example:

```
Rsp = FixRight(Rsp, 8, "A")
```

If Rsp was “123” then Rsp becomes “AAAAAA123” because it puts the character ‘A’ as many times as necessary before the Rsp value until the length of Rsp is 8, left justifying the StringData value.

## Ins

The Insert function inserts a value into a field, subfield, or sub-subfield of a dynamic array.

Syntax: vREC = Ins(vREC, vFLD, [vSUB], [vSSUB], vSTR)

vREC      (Variant) is the dynamic array string variable name

vFLD      (Variant) is the array field number to search

vSUB      (Variant) Optional – is the array subfield number

vSSUB    (Variant) Optional – is the array sub-subfield number

vSTR      (Variant) is data or variable name, which will be inserted at the dynamic array location (vFLD, vSUB, vSSUB).

Examples:

```
Dim sNames As String  
sNames = Ins(sNames, 2, 0, 0, "John")
```

Inserts string ‘John’ into dynamic array sNames. If sNames was originally null, then after the insert, field2 contains ‘John’; i.e., sNames = '&John'. Here, field1 is null.

Note: Suppressing the zeros [i.e., sNames = INS(sNames, 2, “John”)] gives the same result.

```
sNames = Ins(sNames, 2, "Mike")
```

If data already exists for field2 (John), then an additional insert would move field2 data to field3; i.e., sNames = '&Mike&John'. Here, field1 is null, field2 = ‘Mike’, and field3 = ‘John’.

Using the subfields the following string can be created one element at a time using 9 Insert statements:

```
"John#31#English&Mike#34#Spanish&Albert#40#English"

sNames = Ins(sNames, 1, "John")
sNames = Ins(sNames, 1, 2, "31") ' First record,
second field
sNames = Ins(sNames, 1, 3, "English") ' First record,
third field

sNames = Ins(sNames, 2, "Mike")
sNames = Ins(sNames, 2, 2, "34") ' Second record,
second field
sNames = Ins(sNames, 2, 3, "Spanish") 'Second record,
third field

sNames = Ins(sNames, 3, "Albert")
sNames = Ins(sNames, 3, 2, "40") ' Third record,
second field
sNames = Ins(sNames, 3, 3, "English") 'Third record,
third field
```

If you used insert statements and the sub-subfield values you can get this:

"John#31#Canadian English@UnitedKingdom English&Mike"

by

```
sNames = Ins(sNames, 1, "John")
sNames = Ins(sNames, 1, 2, "31") ' First record,
second field

sNames = Ins(sNames, 1, 3, 1, "Canadian English") ' First record, third
field, first sub-field

sNames = Ins(sNames, 1, 3, 2, "UnitedKingdom
English") ' First record, third field, second sub-
field
sNames = Ins(sNames, 2, "Mike")
```

## LField

The LField function searches a string from the left to extract a sub-string by using a specified delimiter character.

Syntax: vVAL= LField(vStringData, vDelimiter, vStartField, [vNumberFields])

vVAL (Variant) is the resulting sub-string value located in StringData  
vStringData (Variant) is a string or string variable to be searched  
vDelimiter (Variant) is a specified delimiter character  
vStartField (Variant) Optional – is the Delimiter to start from when searching the StringData  
vNumberFields(Variant) Optional – specifies the number of sub-fields to retrieve.

Examples:

```
If VAR = "111|222|333", then:  
VAL = LField(VAR, "|", 1) returns VAL = '111'  
VAL = LField(VAR, "|", 3) returns VAL = '333'
```

Note: If StartField = 1, then the LField function will return a sub-string which starts at the beginning of VAR, up to the first occurrence of Delimiter.

## Locate

The Locate function may be used to find the index of a field, subfield, or sub-subfield within a dynamic array.

Syntax: vNbr = Locate(vVAL, vStringData, [vFLD], [vSUB], [vOption])

vNbr (Variant) is an integer that indicates the location of VAL, or 0 (zero) if VAL is not located.  
vVAL (Variant) is the string value to be located  
vStringData (Variant) is the dynamic array variable that will be searched  
vFLD (Variant) Optional – is the array field number to search  
vSUB (Variant) Optional – is the array subfield number to search  
vOption (Variant) Optional – the column within the row to search

Examples:

*The '&' character is used like the Chr(1) delimiter to separate the different rows. The '#' character is used like the Chr(2) delimiter to separate different values within a record, In this case, the name and the age and the language. The '@' symbol is used like Chr(3) to delimit between a sub-field. In this case, the difference between 2 languages.*

```
Dim Nbr As Integer  
Dim sNames As String  
sNames = "John#31#Canadian English@UnitedKingdom  
English&Mike"
```

```
Nbr = Locate("Mike", sNames)    ' Nbr = 2
Nbr = Locate("John", sNames)    ' Nbr = 0 since there
is depth to record 1
Nbr = Locate("John", sNames, 1)  ' Nbr = 1
Nbr = Locate("31", sNames, 1)   ' Nbr = 2 since 1st
record was specified
Nbr = Locate("UnitedKingdom English", sNames, 1, 3)
```

*Nbr = 2 since 1<sup>st</sup> record and 3<sup>rd</sup> subfield was specified*

*If you do not know the row number where your data is then you can specify 0 in place of the Field and Sub values to search the entire array.*

```
Nbr = Locate("31", sNames, 0, 0, "V2")    ' Nbr = 1
```

*Assuming you know the layout of the array (Name, Age, Language) then you can locate the "31" anywhere in the array and specifically look at column 2 for the match. This will return which row is the first with the data.*

*If your columns are Chr(3)-delimited and the rows are still Chr(1)-delimited then use an "S" to find the row:*

```
sNames = "Names" & "John@Mike@Steve@Rob" &
"Addresses"
```

```
Nbr = Locate("Steve", sNames, 0, 0, "S3")    ' Nbr = 2
```

*You still need to know the format of the array so, in this case, Steve was found in the known third position of the unknown second row.*

## LocateAdd

The LocateAdd function will add a value to the dynamic array only if the value being added does not already exist. If the value does exist but has associated subfields the new value will still be added to the end of the list. This function does not interact with SUB and SSUB fields and therefore treats a whole record with subfields as 1 string during the compare.

Syntax: LocateAdd(vVAL, vInfo, [vDLM])

vVAL (Variant) is the value to be located or added to the Dynamic Array

vInfo (Variant) is the dynamic array variable that will be searched

vDLM (Variant) Optional – is the delimiter to use when searching

Examples:

```
Dim sNames As String
```

```

sNames = "John#31#Canadian English@UnitedKingdom
English&Mike"

LocateAdd("Mike", sNames) ' Since Mike already exists
nothing is added
LocateAdd("John", sNames) ' Since John is associated
with subfields in the first record, it will again be
added to the end of the string:
"John#31#Canadian English@UnitedKingdom
English&Mike&John"
LocateAdd("Albert", sNames, "X")

```

This adds “Albert” to the end of the list and uses the letter ‘X’ to separate the last entry and Albert.

## **LocateDel**

The LocateDel function will remove a value from the dynamic array and report the location in the list where it was. If the value does not exist, 0 is returned. This function does not interact with SUB and SSUB fields and therefore treats a whole record with subfields as 1 string during the compare.

Syntax: vNbr = LocateDel(vVAL, vInfo, [vDLM])  
 vNbr (Variant) the position in the dynamic array where the VAL was found and deleted  
 vVAL (Variant) the value to be located and deleted from the Dynamic Array  
 vInfo (Variant) is the dynamic array variable that will be searched  
 vDLM (Variant) Optional – is the delimiter to use when searching

**Examples:**

```

Dim Nbr As Integer
Dim sNames As String
sNames = "John#31#Canadian English@UnitedKingdom
English&Mike"

Nbr = LocateDel("Mike", sNames) ' Nbr = 2, the second
record
Nbr = LocateDel("31", sNames, Chr(2))

```

Nbr = 2, since the age is a subfield specifying the Chr(2) delimiter finds “31” in the second position of the first record.

## **Rep**

The Replace function replaces a field, subfield, or sub-subfield in a dynamic array.

Syntax: StringData = vRep(vStringData, vFLD, [vSUB], [vSSUB], vSTR)

vStringData (Variant) is the dynamic array string variable name

vFLD (Variant) is the array field number to search

vSUB (Variant) Optional – is the array subfield number

vSSUB (Variant) Optional – is the array sub-subfield number

vSTR (Variant) is data or variable name, which will be inserted at the dynamic array location (vFLD, vSUB, vSSUB).

Examples:

```
Given : "John#31#Canadian English@UnitedKingdom  
English&Mike"  
StringData = Rep(StringData, 2, "Albert")  ' removes  
Mike and returns:  
"John#31#Canadian English@UnitedKingdom  
English&Albert"  
StringData = Rep(StringData, 1, "Fred")  ' removes  
whole first record and returns: "Fred&Albert"
```

Using subfields, given:

```
"John#31#Canadian English@UnitedKingdom  
English&Mike"  
StringData = Rep(StringData, 1, 2, "80")  ' using  
the first record and the second subfield, 31 is  
replaced with 80 giving:  
"John#80#Canadian English@UnitedKingdom English&Mike"  
  
StringData = Rep(StringData, 1, 3, 2, "US English")  
' using the first record, the third subfield and the  
second sub subfield, United Kingdom English is  
replaces with US English giving:  
"John#80#Canadian English@US English&Mike"
```

## RFIELD

The RFIELD function searches a string from the right to extract a sub-string by using a specified delimiter character.

Syntax: vVAL= RFIELD(vStringData, vDelimiter, vStartField, [vNumberFields])

vVAL (Variant) is the resulting sub-string value located in StringData

vStringData (Variant) is a string or string variable to be searched

vDelimiter (Variant) is a specified delimiter character

vStartField (Variant) is the Delimiter to start from when searching the StringData  
vNumberFields(Variant) Optional – specifies the number of sub-fields to retrieve.

Examples:

```
If VAR = "111|222|333", then:  
VAL = RField(VAR, "|", 1) returns VAL = '333'  
VAL = RField(VAR, "|", 3) returns VAL = '111'
```

Note: If StartField = 1, then the RField function will return a sub-string which starts at the end of VAR, up to the first occurrence of Delimiter.

## Socket Object

The Socket object is provided to the programmer for creating and managing their own WinSock connection from within the solution. This object is only a pass-through to the standard Windows WinSock control so additional documentation about how this works can be easily found on the Internet. There are three sections for this object, properties, methods and events.

This object is declared in a similar manner to:

```
Dim WithEvents oSocket As Socket
```

The properties for the Socket object are as follows:

### BytesReceived

This function returns the number of bytes currently in the receive buffer. This is a read-only property and is unavailable at design time. The value returned is a long integer.

Syntax: nValue = oSocket.BytesReceived  
nValue (Long) the number of bytes

Example:

```
Dim WithEvents oSocket As Socket ' module level Dim  
Dim nBytes As Long  
Set oSocket = New Socket  
nBytes = oSocket.BytesReceived
```

### LocalHostName

The LocalHostName function returns the name of the local host system. This is read-only property and is unavailable at the design time. The value returned is a string.

Syntax: sValue = oSocket.LocalHostName  
sValue (String) the name of the local host

Example:

```
Dim WithEvents oSocket As Socket ' module level Dim
Dim sName As String
Set oSocket = New Socket
sName = oSocket.LocalHostName
```

## LocalIP

The LocalIP function returns the local host system IP address in the form of a string, such as 11.0.0.127. This property is read-only and is unavailable at design time.

Syntax: sValue = oSocket.LocalIP  
sValue (String) the IP address of the local host

Example:

```
Dim WithEvents oSocket As Socket ' module level Dim
Dim sIP As String
Set oSocket = New Socket
sIP = oSocket.LocalIP
```

## Protocol

This property can get or set the protocol used to either TCP or UDP.

Syntax: nValue = oSocket.Protocol

Alternate: oSocket.Protocol = enValue

enValue (enWinsockProtocols) the numeric value representing TCP or UDP. TCP = 0, UDP = 1. There are built in constants to represent these value as well. They are sckTCPProtocol and sckUDPProtocol.

nValue (Long) the numeric value representing TCP or UDP. TCP = 0, UDP = 1.

Examples:

```
Dim WithEvents oSocket As Socket ' module level Dim
Dim wProtocols As enWinsockProtocols
Set oSocket = New Socket
oSocket.Protocol = sckTCPProtocol
wProtocols = oSocket.Protocol
```

## **RemoteHost**

The RemoteHost property returns or sets the remote host. This can be both read from and written to and is available both in design time and runtime. The value returned is a string and can be specified either as an IP address or as a DNS name.

Syntax: sValue = oSocket.RemoteHost

Alternate: oSocket.RemoteHost = Value

sValue     (String) the name of the host system the client will be connecting to.

**Example:**

```
Dim WithEvents oSocket As Socket ' module level Dim  
Dim sName As String  
Set oSocket = New Socket  
sName = oSocket.RemoteHost
```

## **RemoteHostIP**

This property returns or sets the remote host IP address.

Syntax: sValue = oSocket.RemoteHostIP

Alternate: oSocket.RemoteHostIP = Value

sValue     (String) the IP address of the host system the client will be connecting to.

**Example:**

```
Dim WithEvents oSocket As Socket ' module level Dim  
Dim sIP As String  
Set oSocket = New Socket  
sIP = oSocket.RemoteHostIP
```

## **RemotePort**

This property returns or sets the remote port number.

Syntax: nValue = oSocket.RemotePort

Alternate: oSocket.RemotePort = Value

nValue     (Long) the port number used to access the host system the client will be connecting to.

**Examples:**

```
Dim WithEvents oSocket As Socket ' module level Dim  
Dim nPort As Long  
Set oSocket = New Socket  
oSocket.RemotePort = 11908
```

```
nPort = oSocket.RemotePort
```

## State

This property returns the state of the control as expressed by an enumerated list. This is read-only property and is unavailable at design time.

Syntax: nValue = oSocket.State

nValue (Long) the number assigned to the different socket states  
0 = sckClosed  
1 = sckOpen  
2 = sckListening  
3 = sckConnectionPending  
4 = sckResolvingHost  
5 = sckHostResolved  
6 = sckConnecting  
7 = sckConnected  
8 = sckClosing  
9 = sckError

Example:

```
Dim WithEvents oSocket As Socket ' module level Dim
Private Sub oSocket_OnConnect()
    On Error Resume Next
    '
    Set oSocket = New Socket
    If oSocket.State <> sckConnected Then
        App MsgBox("Connect failed.")
    End Sub
```

The methods for the Socket object are as follows:

## sktClose

This method terminates a TCP connection from either the client or server applications. If the object is declared using the WithEvents option, then an OnClose event is available in the script environment but will not fire with the use of this method but only if the connection is closed by the server.

Syntax: oSocket.sktClose

Example:

```
Dim WithEvents oSocket As Socket ' module level Dim
Set oSocket = New Socket
If oSocket.State = sckConnected Then oSocket.sktClose
```

## sktConnect

This method requests a connection to a remote computer. If the object is declared using the WithEvents option, then an OnConnect event is available in the script environment.

Syntax: oSocket.sktConnect([vRemoteHost], [vRemotePort])

vRemoteHost    (Variant) Optional – allows a connection to a host that is not specified in the RemoteHost property  
vRemotePort    (Variant) Optional – allows a connection to a host that is not specified in the RemotePort property

### Example:

```
Dim WithEvents oSocket As Socket ' module level Dim
Set oSocket = New Socket
oSocket.Protocol = sckTCPProtocol
oSocket.RemoteHost = "127.0.0.1"
oSocket.RemotePort = 21097
oSocket.sktConnect

Private Sub oSocket_OnConnect()
    On Error Resume Next
    '
    Set oSocket = New Socket
    Do
        Loop Until oSocket.State <> sckConnecting
        If oSocket.State <> sckConnected Then App MsgBox
        "Connect failed."
    End Sub
```

## sktGetData

This method retrieves the current block of data from the buffer and then stores it in a variable of the variant type. If the object is declared using the WithEvents option, then an OnDataArrival event is available in the script environment.

Syntax:    oSocket.sktGetData(vData, [vType], [vMaxLen])

vData        (Variant) Where retrieved data will be stored after the method returns successfully. If there is not enough data available for requested type, vData will be set to Empty.

vType        (Variant) Optional – specifies the type of data being retrieved

Byte        = vbByte

Integer     = vbInteger

Long        = vbLong

Single      = vbSingle

	Double	= vbDouble
	Currency	= vbCurrency
	Date	= vbDate
	Boolean	= vbBoolean
	SCODE	= vbError
	String	= vbString
	Byte Array	= vbArray + vbByte
vMaxLen	(Variant)	Optional – Specifies the desired size when receiving a byte array or a string. If this parameter is missing for byte array or string, all available data will be retrieved. If provided for data types other than byte array and string, this parameter is ignored.

Example:

```
Dim WithEvents oSocket As Socket ' module level Dim

Private Sub oSocket_OnDataArrival(ByVal bytesTotal As Long)
    On Error Resume Next
    '
    Dim sData As String
    Set oSocket = New Socket
    oSocket.sktGetData(sData, vbString, bytesTotal)
End Sub
```

## sktPeekData

This method operates in a fashion similar to the sktGetData method. However, it does not remove data from the input queue. It is used to see what data is coming before using the sktGetData method to retrieve and delete the data from the receive buffer. This method works only for TCP connections.

Syntax:	oSocket.sktPeekData(vData, [vType], [vMaxLen])
vData	(Variant) Where retrieved data will be stored after the method returns successfully. If there is not enough data available for requested type, vData will be set to Empty.
vType	(Variant) Optional – specifies the type of data being retrieved
	Byte = vbByte
	Integer = vbInteger
	Long = vbLong
	Single = vbSingle
	Double = vbDouble
	Currency = vbCurrency
	Date = vbDate

	Boolean	= vbBoolean
	SCODE	= vbError
	String	= vbString
	Byte Array	= vbArray + vbByte
vMaxLen	(Variant)	Optional – Specifies the desired size when receiving a byte array or a string. If this parameter is missing for byte array or string, all available data will be retrieved. If provided for data types other than byte array and string, this parameter is ignored.

Example:

```
Dim WithEvents oSocket As Socket ' module level Dim

Private Sub oSocket_OnDataArrival(ByVal bytesTotal As
Long)
    On Error Resume Next
    '
    Dim sData As String
    Set oSocket = New Socket
    oSocket.sktPeekData(sData, vbString, bytesTotal)
End Sub
```

## sktSendData

This method dispatches data to the remote computer. When a UNICODE string is passed in, it is converted to an ANSI string before being sent out on the network.

Syntax: oSocket.sktSendData(vData)

vData (Variant) Data to be sent. For binary data, byte array should be used.

Example:

```
Dim WithEvents oSocket As Socket ' module level Dim
Set oSocket = New Socket
oSocket.sktSendData("Hello world.")
```

The events for the Socket object are as follows:

## OnClose

Occurs when the connection has been closed by the remote host. This does not occur when the sktClose method has been called.

Example:

```
Dim WithEvents oSocket As Socket ' module level Dim
Private Sub oSocket_OnClose()
```

```
On Error Resume Next  
End Sub
```

## OnConnect

Occurs when a connection is successfully established

Example:

```
Dim WithEvents oSocket As Socket ' module level Dim  
Private Sub oSocket_OnConnect()  
    On Error Resume Next  
End Sub
```

## OnDataArrival

Occurs when data arrives and is used to process incoming data.

Example:

```
Dim WithEvents oSocket As Socket ' module level Dim  
Private Sub oSocket_OnDataArrival(ByVal bytesTotal As  
Long)  
    On Error Resume Next  
End Sub
```

## OnError

Occurs when there is an error, such as a PC not existing.

Example:

```
Dim WithEvents oSocket As Socket ' module level Dim  
Private Sub oSocket_OnError(ByVal Number As Long,  
ByVal Description As String)  
    On Error Resume Next  
End Sub
```

## OnSendComplete

Occurs when the data has been sent

Example:

```
Dim WithEvents oSocket As Socket ' module level Dim  
Private Sub oSocket_OnSendComplete()  
    On Error Resume Next  
End Sub
```

## OnSendProgress

Occurs when the data is being sent

**Example:**

```
Dim WithEvents oSocket As Socket ' module level Dim  
Private Sub oSocket_OnSendProgress(ByVal bytesSent As  
Long, ByVal bytesRemaining As Long)  
    On Error Resume Next  
End Sub
```

## Web Object

The Web object is provided to the programmer for managing the Web data connector configured as one of the data connections.

This object is declared in a similar manner to:

```
Dim oWeb As Web
```

The properties for the Web object are as follows:

### ConnectTimeout

The ConnectTimeout property returns or sets the timeout to connect for the HTTP request. When the language extension is created it will initialize this value to what is configured in the data connection configuration.

Syntax: nValue = oWeb.ConnectTimeout

Alternate: oWeb.ConnectTimeout = Value

nValue      (Long) the number of milliseconds the system will wait for a connection

Examples:

```
Dim oWeb As Web
```

```
Dim nTimeout As Long
```

```
nTimeout = oWeb.ConnectTimeout
```

```
oWeb.ConnectTimeout = 10000
```

### Data

The Data property returns or sets the data portion of the HTTP request. This is likely to be the collected data requiring validation or the return values that come from the web server.

Syntax: oWeb.Data = sValue

Alternate: sValue = oWeb.Data

sValue      (String) the data to be sent as part of the request or returned from the server

**Example:**

```
Dim oWeb As Web
Dim sPart As String
Dim sQty As String
sPart = "100620"
sQty = "100"
oWeb.Data = sPart & " | " & sQty
```

## **DataSource**

The **DataSource** property returns or sets an index of the data connection that should be used. If a **DataSource** is not specified the language extension will use the first data connection that is configured as a WEB connection.

Syntax: vValue = oWeb.DataSource

Alternate: oWeb.DataSource = vValue

vValue (Variant) the number or name of the data connection object.

**Examples:**

```
Dim oWeb As Web
Dim vConnID As Variant
oWeb.DataSource = 2
oWeb.DataSource = "MyWebServer"
vConnID = oWeb.DataSource
```

## **HeaderValue**

The **HeaderValue** property returns or sets what the HTTP or HTTPS header values are.

Syntax: oWeb.HeaderValue(sIndex) = bValue

Alternate: bValue = oWeb.HeaderValue(sIndex)

bValue (Boolean) contains a True for False for the success of the HeaderValue assignment.

sIndex (String) the property name for the HTTP header

**Example:**

```
Dim oWeb As Web
oWEB.HeaderValue("Content-Type") = "text/xml;
charset=utf-8"
oWEB.HeaderValue("SOAPAction") =
"""/xml.namespaces.xerox.com/im
/xip/services/xclimswebservice/wsdl/getDistributionRe
furbInfo"""
```

## **QueryType**

This property sets how the Web object will communicate with the web server. Get, Put and Post are the available options.

Syntax: oWeb.QueryType = enValue

Alternate: enValue = oWeb.QueryType

enValue (QueryType) contains three options

HTTP\_GET – it is for obtaining a resource, without changing anything on the server.

HTTP\_POST – sends data to a specific URI and expects the resource at that URI to handle the request. The web server at this point can determine what to do with the data in the context of the specified resource.

HTTP\_PUT – puts a file or resource at a specific URI, and exactly at that URI. If there's already a file or resource at that URI, PUT replaces that file or resource. If there is no file or resource there, PUT creates one.

The fundamental difference between the POST and PUT requests is reflected in the different meaning of the Request-URI. The URI in a POST request identifies the resource that will handle the enclosed entity. That resource might be a data-accepting process, a gateway to some other protocol, or a separate entity that accepts annotations. In contrast, the URI in a PUT request identifies the entity enclosed with the request -- the user agent knows what URI is intended and the server MUST NOT attempt to apply the request to some other resource. If the server desires that the request be applied to a different URI, it MUST send a 301 (Moved Permanently) response; the user agent MAY then make its own decision regarding whether or not to redirect the request.

Examples:

```
Dim enType As QueryType  
Dim oWeb As Web  
oWEB.QueryType = HTTP_POST  
enType = oWEB.QueryType
```

## ReceiveTimeout

The ReceiveTimeout property returns or sets the timeout to receive a reply to the HTTP request. If a reply has not been completely received before the timeout then the request will be terminated. When this language extension is created it will initialize this value to what is configured in the data connection configuration.

Syntax: nValue = oWeb.ReceiveTimeout

Alternate: oWeb.ReceiveTimeout = nValue

nValue (Long) the number of milliseconds the system will wait for a connection

Examples:

```
Dim oWeb As Web
Dim nTimeout As Long
nTimeout = oWeb.ReceiveTimeout
oWeb.ReceiveTimeout = 20000
```

## Reply

The Reply property returns the data portion of the HTTP response. This is where the data will be held when the request returns. Keeping the Data and Reply properties separate allows Execute to be called multiple times without requiring the Data property to be reset.

Syntax: sValue = oWeb.Reply

sValue (String) the data being returned to the client from the server

Example:

```
Dim oWeb As Web
Dim sValue As String
sValue = oWeb.Reply
```

## Request

The Request property returns or sets the path to the ASP page (if executed against a Windows server) that is then appended to the URL that is used for connecting.

Syntax: oWeb.Request = sValue

Alternate: sValue = oWeb.Request

sValue (String) part of the URL that is the path to the ASP page

Example:

```
Dim oWeb As Web
Dim sRequest As String
oWeb.Request =
"/data/DoWork.asp?num=100620?MySQLTable"
sRequest = oWeb.Request
```

## SendTimeout

The SendTimeout property returns or sets the timeout value to send the HTTP request. If the send is not complete before the timeout then the request will be terminated. When the language extension is created it

will initialize this value to what is configured in the data connection configuration.

Syntax: nValue = oWeb.SendTimeout

Alternate: oWeb.SendTimeout = nValue

nValue (Long) the number of milliseconds the system will wait for submitting a HTTP request

**Example:**

```
Dim oWeb As Web
Dim nTimeout As Long
nTimeout = oWeb.SendTimeout
oWeb.SendTimeout = 30000
```

The methods for the Web object are as follows:

## Execute

The Execute method will execute the configured oWeb object. vData is optional and is equivalent to the XML data in a SOAP request.

Syntax: bValue = oWeb.Execute([vData])

bValue (Boolean) is True or False depending on the success of the submission to the web server.

vData (Variant) Optional – an object that could contain the individual properties.

**Example:**

```
Dim oWeb As Web
If oWeb.Execute Then
    RFPrompt("txtReply").Text = oWeb.Reply
End If
```

## Login

The Login method is for future use.

## Logout

The Logout method is for future use.

# Stored Procedure Extensions

These *obsolete* commands relate specifically to stored procedures and have been replaced with the Embedded Procedure structure.

## CallAction (not used with Sybase)

This function will call the stored procedure as specified, and will return the output from the procedure.

Syntax: vRetVal = SP.CallAction(sName, vParams)

vRetVal (Variant) is a return value number from the stored procedure. Typically '0' (zero) specifies that no errors occurred during processing.

sName (String) is the name of the stored procedure to call.

vParams (Variant) passing parameters as required by the stored procedure. For stored procedures that return values as passing parameters, any output parameters must be declared as Variant and be set to empty.

Example:

```
Dim vRetVal As Variant  
vRetVal = SP.CallAction ("SPname", Rsp)
```

### **CallProc (must be used with Sybase)**

This function will call the stored procedure as specified, and will return the output from the procedure. An unlimited number of passing parameters Pn (e.g., P1, P2, etc.) may be specified.

Syntax: vErrNo = SP.CallProc(sName, sOptions, sColumns, sRows, vParams)

vErrNo (Variant) is a return value from the stored procedure. Typically '0' (zero) specifies that no errors occurred during processing.

sName (String) is the name of the stored procedure to call.

sOptions (String) are the options for the stored procedure to use.  
Allowable options are:

A = Action - no rows returned

S = Select - rows are returned

O = Open - an alternative required for some databases (i.e., for Sybase: 'A' and 'S' are not useful, use only 'O').

Note: each type may optionally be suffixed by the letters 'Y' or 'N' (for Yes or No) to specify additional possible database requirements, as follows:

1st letter (Y or N): a 'Return' value is specified in the stored procedure.  
1st letter defaults (if not specified) are 'Y' for type 'A', 'N' for type 'S', and 'N' for type 'O'),

2nd letter (Y or N): to declare the passing parameters as 'input' or 'output'. The 2nd letter default (if not specified) is 'Y' for all 3 type ('A', 'S', and 'O'). A tip: consider 'N' for Oracle databases).

Examples:

```
Type='A'  
Type='S'  
Type='ONN'
```

sColumns(String) is a string representation of the columns contained in the associated Records item (below).

sRows (String) is a string representation of any results returned by the stored procedure.

vParams (Variant) passing parameters as required by the stored procedure. For stored procedures that return values as passing parameters, any output parameters must be declared as Variant and be set to empty.

Example:

```
Dim sColumns As String  
Dim sRows As String  
Dim vPn As Variant  
Dim vErrNo As Variant  
vPn = Rsp ' User must provide value  
vErrNo= SP.CallProc("byroyalty", "SNY", sColumns,  
sRows, vPn)
```

The Records variable now contains a resultset from which values can be extracted, using DB.Extract.

### CallSelect (not used with Sybase)

This function will call the stored procedure as specified, and will return the output from the procedure.

Syntax: vErrNo = SP.CallSelect(sSQL, sColumns, sRows, [vParams])

vErrNo (Variant) is a return value from the stored procedure. Typically '0' (zero) specifies that no errors occurred during processing.

sSQL (String) is the name of the stored procedure to call as a string or string variable, along with any passing parameters for the procedure (i.e., "GetCust '1001'").

sColumns (String) is a string representation of the columns contained in the associated Records item (below).

sRows (String) is a string representation of any results returned by the stored procedure.

vParams (Variant) Optional – additional parameters for the CallSelect

Example:

```
Dim vErrNo As Variant  
Dim sSQL As String  
Dim sColumns As String  
Dim sRows As String  
sSQL = "SPQuery" ' Name of stored procedure  
vErrNo = SP.CallSelect(sSQL, sColumns, sRows)
```

## Database Stored Procedure Object

This object is *obsolete* and has been replaced with the Embedded Procedure structure.

The StoredProcedure Object is used to execute a stored procedure. It features the following set of properties and methods with which you can manipulate the object and its content.

### StoredProc Properties and Methods

#### **CommandText**

Sets or returns a value containing a provider command such as a stored procedure call.

Syntax: StoredProcedure.CommandText = sValue  
Alternate: sValue = StoredProcedure.CommandText  
sValue (String) stored procedure name

Example:

See Execute

#### **CommandTimeout**

Indicates how long to wait while executing a command before terminating the attempt and generating an error.

Syntax: StoredProcedure.CommandTimeout = nValue  
Alternate: nValue = StoredProcedure.CommandTimeout  
nValue (Long) time in seconds, default is 30

Example:

See Execute

#### **CommandType**

Specifies the type of command prior to execution to optimize performance.

Syntax: StoredProc.CommandType = enValue  
Alternate: nValue = StoredProc.CommandType  
nValue (Long) numeric value for the command type  
enValue (enCommandTypes) Sets one of the following values:

<u>Constant</u>	<u>Description</u>
dbCmdText	Evaluates CommandText as a textural definition of a command.
dbCmdStoredProc	Evaluates CommandText as a stored procedure that will return records.
dbCmdExecuteNoRecords	Evaluates CommandText as a stored procedure that will return no records.
dbCmdUnknown	The type of command in the CommandText property is not known.
dbExecuteNoRecords	The stored procedure does not return records.
dbManualDeclare	This will execute the stored procedure directly without checking the syntax.
dbOpenNoRecords	For connection to Sybase.

Example:  
See Execute

## CreateParameter

Creates a new Param Object with the specified properties.

Syntax: CreateParameter (nIndex, [enDatatype], [enDirection], [nSize], [vValue])  
nIndex (Long) Represents the parameter's index  
enDatatype (enDataTypes) Optional – the parameter's data type  
enDirection (enDirections) Optional – the parameter's direction  
nSize (Long) Optional – the parameter's length  
vValue (Variant) Optional – the parameter's value

Example:  
See Execute

## Data

A string representation of any results returned by the stored procedure.

Syntax: sValue = StoredProc.Data  
sValue (String) results of the stored procedure

Example:

See Execute

## **DataSource**

This property indicates which data source to connect.

Syntax: StoredProc.DataSource = sValue

Alternate: sValue = StoredProc.DataSource

sValue (String) name of the data source

Example:

See Execute

## **Dict**

A string representation of the columns returned by the SQL statement.

Syntax: sValue = StoredProc.Dict

sValue (String) representation of the columns returned

Example:

See Execute

## **Execute**

Executes the stored procedure in the CommandText Property.

Syntax: [bOK =]StoredProc.Execute

bOK (Boolean) True or False based on the success of calling the stored procedure regardless of the outcome

Example:

```
Dim vDict As Variant
Dim vData As Variant
Dim spObj As StoredProc
Set spObj = New StoredProc
spObj.DataSource = "Oracle"
spObj.CommandText = "byroyalty"
spObj.CommandType = dbCmdStoredProc
spObj.Param(1).DataType = dbInteger
spObj.Param(1).Direction = dbParamInput
spObj.Param(1).Value = 100
spObj.Execute
vDict = spObj.Dict
vData = spObj.Data
```

## **Param**

The Param object represents a parameter or argument associated with a StoredProc object based on a parameterized stored Procedure. The Param object represents in/out arguments and the return values of stored procedures.

Syntax: spValue = StoredProc.Param(nIndex).oProperty  
spValue (StoredParam) data value of the parameter  
nIndex (Long) index of the parameter  
oProperty (object) the available objects are listed below

Example:

See Execute

## Param( ).Datatype

This property indicates the data type of the Param Object.

Syntax: StoredProc.Param(nIndex).Datatype = enValue  
Alternate: nValue = StoredProc.Param(nIndex).Datatype  
nIndex (Long) index of the parameter  
nValue (Long) the numeric value of an enDataTypes value  
enValue (enDataTypes) sets one of the following values:

<u>Constant</u>	<u>Description</u>
dbBigInt	An 8-byte signed integer.
dbBinary	A binary value.
dbBoolean	A Boolean value.
dbBSTR	A null-terminated char str (Unicode).
dbChar	A String value.
dbCurrency	A currency value. Currency is a fixed-point number with 4 digits to the right of the decimal point. It is stored in an 8-byte signed integer scaled by 10,000.
dbDate	A Date value. A date is stored as a Double, the whole part of which is the number of days since December 30, 1899, and the fractional part of which is the fraction of a day.
dbDBDate	A date value (yyyymmdd).
dbDBTime	A time value (hhmmss).
dbDBTimeStamp	A date-time stamp (yyyymmddhhmmss plus a fraction in billionths).
dbDecimal	An exact numeric value with a fixed precision and scale.
dbDouble	A double-precision floating point value.
dbEmptyNo	value was specified.
dbError	A 32-bit error code.

dbGUID	A globally unique identifier (GUID).
dbIDispatch	A pointer to an IDispatch interface on an OLE object.
dbInteger	A 4-byte signed integer.
dbIUnknown	A pointer to an unknown interface on an OLE object.
dbLongVarBinary	A long binary value.
dbLongVarChar	A long String value.
dbLongVarWChar	A long null-terminated string value.
dbNumeric	An exact numeric value with a fixed precision and scale.
dbSingle	A single-precision floating point value.
dbSmallInt	A 2-byte signed integer.
dbTinyInt	A 1-byte signed integer.
dbUnsignedBigInt	An 8-byte unsigned integer (DBType_UI8).
dbUnsignedInt	A 4-byte unsigned integer.
dbUnsignedSmallInt	A 2-byte unsigned integer.
dbUnsignedTinyInt	A 1-byte unsigned integer.
dbUserDefine	A user-defined variable.
dbVarBinary	A binary value.
dbVarChar	A string value.
dbVariant	An Automation Variant.
dbVarWChar	A null-terminated Unicode chr string.
dbWChar	A null-terminated Unicode chr string.

## Param( ).Direction

This property indicates whether the Parameter represents an input parameter, an output parameter, or both, or if the parameter is the return value from a stored procedure.

Syntax: `StoredProc.Param(nIndex).Direction = enValue`

Alternate: `nValue = StoredProc.Param(nIndex).Direction`

`nIndex` (Long) index of the parameter

`nValue` (Long) the numeric value of an `enDirections` value

`enValue` (`enDirections`) sets or returns one of the following values:

<u>Constant</u>	<u>Description</u>
<code>dbParamInput</code>	Default, indicates an input parameter
<code>dbParamOutput</code>	Indicates an output parameter
<code>dbParamInputOutput</code>	Indicates both an input and output parameter
<code>dbParamReturnValue</code>	Indicates a return value.

## Param( ).NumericScale

This property indicates the scale of the numeric value by specifying the number of decimal places to which the number will be resolved. Not all

database types support this parameter. If it is not supported, the ODBC driver will ignore this setting. The default is 0.

Syntax: `StoredProc.Param(nIndex).NumericScale = nValue`

Alternate: `nValue = StoredProc.Param(nIndex).NumericScale`

`nIndex` (Long) index of the parameter

`nValue` (Long) the number of places to resolve the parameter's value

### **Param( ).Precision**

This property indicates the degree of precision for numeric values by specifying the maximum number of digits used for a parameter. Not all database types support this parameter. If it is not supported, the ODBC driver will ignore this setting. The default is 0.

Syntax: `StoredProc.Param(nIndex).Precision = nValue`

Alternate: `nValue = StoredProc.Param(nIndex).Precision`

`nIndex` (Long) index of the parameter

`nValue` (Long) the maximum number of integers that will make up the size of the value

### **Param( ).Size**

This property indicates the maximum size, in bytes or characters of the Param Object. Use the size property to determine the maximum size for values written to or read from the value property of the Param Object.

Syntax: `StoredProc.Param(nIndex).Size = nValue`

Alternate: `nValue = StoredProc.Param(nIndex).Size`

`nIndex` (Long) index of the parameter

`nValue` (Long) size in bytes or characters

### **Param( ).Value**

This property indicates the value assigned to the Param object. Use the Value property to set or return parameter values with Param Objects.

Syntax: `StoredProc.Param(nIndex).Value = vValue`

Alternate: `vValue = StoredProc.Param(nIndex)`

`vValue` (Variant) value of the dbParam

`nIndex` (Long) index of the parameter

### **ParamCount**

This function indicates the number of parameters associated with the current stored procedure object. A parameter's index may be zero-based so the count may appear one less than expected.

Syntax: vValue = StoredProc.ParamCount

vValue (Variant) the number of parameters associated with this stored procedure.

## Prepared

This property indicates whether or not to save a compiled version of a command before execution. The prepared property is used to have the provider save a prepared (or compiled) version of the query specified in the CommandText property before the object's first execution. If the property is False, the provider will execute the object directly without creating a compiled version.

Syntax: StoredProc.Prepared = bValue

Alternate: Value = StoredProc.Prepared

bValue (Boolean) set to True to save a compiled version before execution

## Results

This property represents the entire set of records from a base table or the results of an executed command.

Syntax: oValue = StoredProc.Results

oValue (rdoResultset, adoRecordset) recordset object containing any rows returned by the stored procedure.

# Initialization Files

Initialization files are used to make changes to the default behavior of RFgen such as resetting data connections based on usage, indexing databases, preventing server-side dialog boxes from stopping the RFgen service and altering connectivity parameters for mobile devices.

## RFgen.ini

The RFgen.ini file provides configuration parameters for the following features. In all cases, the headings and settings are case sensitive.

**Note: this file must be created using Notepad and stored in the RFgen directory for it to work.**

### [Options]

ForceReset=0

set to a positive number to force RFgen to reset all data connections every “nn” minutes. Setting this value to 0 disables the reset function. It only impacts pooled data connections.

The TranLimit command will let the Transaction Manager process a specified number of transactions in the queue and then destroy and restart the client process in charge of processing the queue. This allows the system to release the memory for the process and debug memory leaks in the operating system.

TranLimit=x

set x equal to the number of transactions that will be processed before the Transaction Management process as well as its data connections are shut down and recreated. The data connections that are pooled will not be reset.

This command will have RFgen remove a connection from the pool, destroy it, and re-create it upon next use. This allows the system to release the memory for the process and debug memory leaks in the operating system.

ResetUDCx=y

set x equal to the data connector. Set the y value to the number of transactions that

will be processed before the reset takes place. The reset will only impact pooled connections.

In the case of some DB2 ODBC connections, some tables may not be visible to RFgen because of settings for the DB2 database. To make these tables visible, add an entry with the following format. You may have as many schemas as needed. These entries are only examples.

[AS400]	DB2 Data source name in RFgen
Schema1=S104WL5M.RB1	specific database / library index
Schema2=S104WL5M.QSYS	specific database / library index

To check JDE connectivity before a business function is executed, use the CheckX0010 option with the value of 1 and its related parameters.

[JDE]	
CheckX0010=0	perform a GetNextNumber test before BSFN execution to assure JDE connection state, 0 = off, 1 = on
SystemCode=41	parameter for X0010 call
NextNumberingIndexNo=2	parameter for X0010 call
CompanyKey=00001	parameter for X0010 call
DocumentType=IA	parameter for X0010 call

RFgen will monitor the JDE.log file for designated strings and then perform a series of reset based on finding one of the search strings.

CheckLog=1	perform a lookup within the JDE.log file (last 500 characters) for various test strings. If found then it will stop and restart all RFgen child processes, 0 = off, 1 = on
LogText1=RESETNOW	string 1 to look for in JDE.log
LogText2=JDB9900400	string 2 to look for in JDE.log

If one of the LogText parameters is found the server will:

1. Delete the log file. JDE creates a new file every time it starts.
2. Suspend all active client connections temporarily
3. Close all transaction manager client(s).
4. Close all data connections.
5. Start all data connections.
6. Start the transaction manager client(s).
7. Resume the active client connections.

RFgen can be made to reset every process after a certain number of transactions go through the queue. This includes thin client device connections, ODBC, SM, and ERP connections.

**TranLimit=1000** maximum number of transactions to process before resetting all RFgen child processes (data connections and all connected users).

# ERPDIALOGS.INI

The ERPDialogs.ini file provides the ability to automatically respond to dialog boxes created by software (such as JD Edwards) running on the RFgen (Communications) server.

## Syntax:

Search, Title, Message, Reset, Close, Text

Search      1 = Search on Title, 2 = Search on Message

## Title      Window Title

Message    Message Text (only if 'Search' = 2)

**Reset** 0 = Don't reset connection, 1-5 = Connection number to reset

Close      1 = Close Window, 2 = Click button

Text      Text on button to click (only if 'Close' = 2)  
(include "&" to represent underscore, e.g., "E&xit" for "Exit").

Examples:

```
1, Database Password Entry, ,1,1,
1, Database Error,,1,1,
1, OneWorld Error,,1,1,
1, OneWorld,,1,1,
1, PeopleSoft Error,,1,1,
1, PeopleSoft,,1,1,
1, Remote Job,,1,1,
1, Database Password Entry, ,1,1,
1, Database Error,,1,1,
1, OneWorld Error,,1,1,
1, OneWorld,,1,1,
1, PeopleSoft Error,,1,1,
1, PeopleSoft,,1,1,
1, Remote Job,,1,1,
2, ,JDB9103 - Fetch failed, 1, 2,
2, ,to Activate the component and
problem, 1, 2, Switch To...
1, RFDB,, 1, 2, Retry
```

Even if 'Search' = 2, a window title must be included if one exists.

## GPRS.ini

This ini file is designed to make the RFgen mobile client dial a connection so that data can be sent or retrieved to the server while in a disconnected state. This file is not intended to establish a connection for a thin client connection. If a GPRS connection is desired for a thin client connection simply establish it first, then any VPN connections if required and then launch the RFgen thin client.

In a typical implementation Server.Connect and Server.Disconnect are used both at the beginning of the process, to retrieve validation data, and at the end to submit collected data. When the connect command is executed the mobile client checks the phonebook setting in the RFgenCFG.exe program and uses either WiFi or GPRS to make the connection. If GPRS is selected the ini file tells RFgen what settings to use. If RFgen needs to establish the VPN connection as well then those settings can be documented as well.

Note that the GPRS and VPN connections must first be setup in the CE environment and RFgen simply references and starts those connections.

Using any desired method, create and deploy a file called GPRS.ini to the install directory (by default: \Program Files\RFgenCE). It contains the following settings:

```
[GPRS]
Enabled=True
Name=GPRS
User=
Pwd=
```

```
[VPN]
Enabled=True
Name=My Work Network
User=
Pwd=
```

The **Enabled** parameter just tells RFgen if it should make that connection.

The **Name** parameter is the name used in the setup of the connection in the operating system's configuration.

The **User** and **Pwd** fields, if needed, are stored in the ini file as well.

## Client.ini

This ini file is designed to allow the user to change the location of the client database. The client database stores the configuration of the Desktop Client. The usual place the client.ini file is created is:

C:\Users\username\AppData\Roaming\RFgen5

The Desktop Client reads the client.ini file for the rfgendb, the database storing the configuration. If an entry exists it will use that database, if not the Desktop Client will look for the rfgendb file in the default location. Here is an example of a database path entry:

```
[Environment]
Location=3,25,323,297
GUID=5907C90C-C337-4A57-8895-BE891D80CFD5
WindowState=1
Monitors=2
rfgendb=c:\programdata\rfgen5\rfgen.xdb
```

# Index

## A

AbortTrans, 339  
Activate, 348  
Activate the Service, 195  
Active Clients View, 198  
Active Connections View, 201  
ActiveX files in VBA, 225  
AddDataPoint, 372  
AddError, 308, 309  
AddItem, 251, 399  
AddItemEx, 252  
AddNew, 422  
Administration Services, 14  
ADO Language Extensions, 23  
Align, 235, 245  
Alpha Only Validation, 116  
App Object Extensions, 274  
App..LogError, 281  
App.LogErrorEx, 281  
App.Balloon, 274  
App.CallForm, 275  
App.CallMacro, 275  
App.CallMenu, 276  
App.ChangeLoginForm, 276  
App.ChangeUserPassword, 277  
App.ClearValues, 277  
App.ClientType, 277  
App.ConnAvailable, 277  
App.ExecuteMenuSelection, 278  
App.ExitForm, 278  
App.ExitSession, 278  
App.GetInput, 278  
App.GetString, 279  
App.GetValue, 280  
App.ipAddress, 280  
App.Locale, 280  
App.MakeList, 282  
App.MsgBox, 282  
App.PromptCount, 284

App.PromptNo, 284  
App.SendChar, 284  
App.SendKey, 284  
App.SetDisplay, 285  
App.SetFocus, 286  
App.SetMenu, 287  
App.SetValue, 287  
App.ShowList, 287  
App.Sleep, 288  
App.TimerEnabled, 288  
App.TimerInterval, 288  
App.User, 289  
App.UserProperty, 289  
AppendItem, 290  
Application Based Extensions, 274  
Application Construction Area, 121  
Application Screens, 162  
Application Statistics, 70  
Application Testing, 51  
Applications, 78  
Applications List, 82  
AppName, 308, 309  
Area Chart, 378  
Authorize the Server, 193, 195  
Auto Upgrade Clients, 212  
Autosize, 235, 245  
AxesFont, 372  
AxesLabel, 373  
AxesTitleFont, 373

## B

BackColor, 373  
BackColor1, 236, 245  
BackColor2, 236, 246  
BackGradient, 237, 246  
Balloon, 274  
Bar Chart, 379  
Basic Implementation Steps, 6  
Batch Failover, 136  
Before You Begin, 4

BeginTrans, 320, 343  
Bell, 293  
Bitmap, 242  
Bold, 240, 247  
BorderStyle, 237, 246  
Branch/Goto Validation, 119  
Button Object, 86  
BytesReceived, 437

## C

Calculation Defaults, 110  
CallAction, 449  
CallForm, 275  
CallMacro, 275, 298  
CallMenu, 276  
CallProc, 450  
CallSelect, 451  
Candle Chart, 380  
Caption, 237  
Cell, 252, 399  
ceObject Object Extensions, 327  
ChangeLoginForm, 276  
ChangeUserPassword, 277  
Character String Extraction Default, 109  
Character String Validation, 115  
**Chart Object**, 371  
Chart.AddDataPoint, 372  
Chart.AxesFont, 372  
Chart.AxesLabel, 373  
Chart.AxesTitleFont, 373  
Chart.BackColor, 373  
Chart.ForeColor, 373  
Chart.Header, 374  
Chart.HeaderFont, 374  
Chart.Height, 374  
Chart.Image, 375  
Chart.LegendFont, 375  
Chart.SeriesColor, 375  
Chart.SeriesLabel, 375  
Chart.ThreeD, 376  
Chart.Type, 376  
Chart.Width, 376  
Charts Area, 378  
Charts Bar, 379

Charts Candle, 380  
Charts FilledRadar, 382  
Charts HiLo, 384  
Charts HiLo OpenClose, 386  
Charts Pie, 388  
Charts Plot, 389  
Charts Polor, 391  
Charts Radar, 393  
Charts StackingBar, 395  
Charts Surface, 397  
CheckBox Object, 87  
Checked, 238  
CheckStatus, 339  
ClassObject, 416  
Clear, 256, 290, 293, 308, 402, 416, 423  
ClearEOL, 293  
ClearEOP, 294  
ClearValues, 277  
Click, 226  
ClickAndSkipPrompts, 327  
ClickCoordinates, 327  
Client Inactivity Timeout, 19  
ClientType, 277  
Column, 256  
ColumnCount, 416  
ColumnName, 417  
Columns, 408  
ComboBox Object, 87  
CommandText, 452  
CommandTimeout, 299, 452  
 CommandType, 452  
CommitTrans, 321, 344, 355  
Concatenation Default, 109  
Configure Source Control Integration, 25  
Configure the Server, 193  
Configuring Application Services, 13  
Configuring Database Connections, 27  
Configuring Desktop Preferences, 16  
Configuring Environment Properties, 17  
Configuring ERP Connection, 38  
Configuring Mobile Application  
    Database, 8  
Configuring Mobile Device Themes, 145  
Configuring ODBC Data Sources, 35  
Configuring RFgen Software, 8

Configuring Screen Mapping  
    Connection, 38  
Configuring System Queues and Tasks, 26  
Configuring the Event Logs, 22  
Configuring the Performance Monitoring, 21  
Configuring the Scripting Environment, 23  
Configuring Transaction Management, 42  
Configuring Web Connection, 40  
ConnAvailable, 277  
Connect, 299  
Connected, 355  
**Connection Pooling**, 33  
Connection Statistics, 70  
ConnectionProperty, 311  
ConnectTimeout, 445  
Contains String Validation, 117  
Controls, 123  
Copies, 349  
Count, 291, 308, 321, 408  
Create, 337  
CreateParameter, 453  
CurScreen, 355

## D

Data, 445, 453  
Data Range Validation, 117  
Database Path, 138  
Database Related Extensions, 320  
DataRecord Object, 422  
DataRecord.AddNew, 422  
DataRecord.Clear, 423  
DataRecord.IsEOF, 423  
DataRecord.MoveFirst, 423  
DataRecord.MoveLast, 423  
DataRecord.MoveNext, 423  
DataRecord.MovePrevious, 423  
DataRecord.MoveTo, 424  
DataRecord.Param, 424  
DataRecord.ParamCount, 424  
DataRecord.ParamName, 425

    DataRecord.RowCount, 425  
    DataRecord.Schemald, 426  
DataSource, 417, 446, 454  
Date Validation, 117  
DB Object Extensions, 320  
DB.BeginTrans, 320  
DB.CommitTrans, 321  
DB.Count, 321  
DB.Execute, 321  
DB.Extract, 322  
DB.MakeList, 322  
DB.OpenResultset, 323  
DB.RedirectDataSource, 325  
DB.RollbackTrans, 325  
DB.SaveBitmap, 326  
DB.UseDataSource, 326  
dbBigInt, 455  
dbBinary, 455  
dbBoolean, 455  
dBSTR, 455  
dbChar, 455  
dbCmdExecuteNoRecords, 453  
dbCmdStoredProc, 453  
dbCmdText, 453  
dbCmdUnknown, 453  
dbCurrency, 455  
dbDate, 455  
dbDBDate, 455  
dbDBTime, 455  
dbDBTimeStamp, 455  
dbDecimal, 455  
dbDouble, 455  
dbEmptyNo, 455  
dbError, 455  
dbExecuteNoRecords, 453  
dbGUID, 456  
dbIDispatch, 456  
dbInteger, 456  
dbUnknown, 456  
dbLongVarBinary, 456  
dbLongVarChar, 456  
dbLongVarWChar, 456  
dbManualDeclare, 453  
dbNumeric, 456  
dbOpenNoRecords, 453

dbParamInput, 456  
dbParamInputOutput, 456  
dbParamOutput, 456  
dbParamReturnValue, 456  
dbSingle, 456  
dbSmallInt, 456  
dbTinyInt, 456  
dbUnsignedBigInt, 456  
dbUnsignedInt, 456  
dbUnsignedSmallInt, 456  
dbUnsignedTinyInt, 456  
dbUserDefine, 456  
dbVarBinary, 456  
dbVarChar, 456  
dbVariant, 456  
dbVarWChar, 456  
dbWChar, 456  
DCount, 426  
DebugLog, 417  
Declarations, 226  
Defaults, 238  
Del, 427  
DeleteProperty, 316  
Deploy Mobile Applications, 58  
Description, 309  
Design Mode, 16  
Device Object Extensions, 327  
Device Profiles, 133  
Device.ClickAndSkipPrompts, 327  
Device.ClickCoordinates, 327  
Device.EnableGPS, 328  
Device.ForceLocal, 328  
Device.GetGPSInfo, 328  
Device.GoOffline, 329  
Device.GoOnline, 330  
Device.Offline, 331  
Device.Online, 331  
Device.Platform, 331  
Device.PlaySound, 332  
Device.PrinterOff, 332  
Device.PrinterOn, 332  
Device.ReadFile, 333  
Device.Send, 333  
Device.SendCommPort, 333  
Device.SetCameraOption, 334  
Device.SetCommMode, 334  
Device.SetCommPort, 334  
Device.TakePicture, 335  
Device.WriteFile, 336  
DeviceObject, 336  
DeviceObject.Create, 337  
DeviceObject.Execute, 337  
DeviceObject.LastError, 338  
DeviceObject.Name, 338  
DeviceObject.Release, 338  
DeviceObject.ReturnValue, 338  
Devices Menu, 58  
Dict, 454  
DisableParam, 417  
DisableTimeout, 316  
Discard Form Data When Chaining, 17  
Disconnect, 300  
Displaying Data From Other Tables, 108  
DisplayMode, 243  
DisplayOnly, 238  
Down Arrow as Enter Key, 17  
Download Enterprise Objects, 42  
DrawLine, 294  
Dynamic Array Structure, 426

## E

Edits, 239  
embedded methods, 163  
Embedded Procedure, 416  
Embedded Procedures, 412, 413  
emProc.ClassObject, 416  
emProc.Clear, 416  
emProc.ColumnCount, 416  
emProc.ColumnName, 417  
emProc.DataSource, 417  
emProc.DebugLog, 417  
emProc.DisableParam, 417  
emProc.Execute, 418  
emProc.ExecuteMethod, 418  
emProc.LogMode, 418  
emProc.Name, 419  
emProc.Param, 419  
emProc.ParamCount, 419  
emProc.ParamEx, 419

emProc.ParamName, 420  
emProc.Queue, 421  
emProc.QueueName, 421  
emProc.QueueOffline, 421  
emProc.QueueSeqNo, 422  
emProc.RowCount, 422  
EnableGPS, 328  
Encryption Key, 9  
EndDoc, 349  
Enterprise Resource Planning  
    Extensions, 343  
Environment Properties, 21, 221  
EnvironmentProperty, 316  
ERP Business Functions Download, 45  
ERP Object Extensions, 343  
ERP.SetHardRelease, 348  
ERP.BeginTrans, 343  
ERP.CommitTrans, 344  
ERP.LogOff, 344  
ERP.LogOn, 344  
ERP.MakeList, 346  
ERP.ReadData, 347  
ERP.RollbackTrans, 347  
ERP.Session, 348  
ErrMsg, 239  
Escape Processing Delay, 19  
Event Logs, 72  
Events, 124, 226  
Execute, 321, 337, 418, 449, 454  
ExecuteMenuItem, 278  
ExecuteMethod, 418  
ExecuteSQL, 300  
ExitForm, 278  
ExitSession, 278  
Export Items, 65  
Ext, 429  
Extract, 322  
Extraction Default, 109

## F

FieldId, 239  
FilledRadar Chart, 382  
Find in Application Scripts, 67  
FindText, 356

FixLeft, 430  
FixRight, 430  
Font.Bold, 240, 247  
Font.Italic, 240, 247  
Font.Size, 240, 248  
Font.Underline, 241, 248  
FontBold, 350  
FontItalic, 350  
FontName, 350  
FontSize, 350  
FontStrikeThru, 351  
FontUnderline, 351  
ForceLocal, 328  
ForeColor, 241, 248, 373  
Format, 241  
Forms, 82  
Frame Object, 87, 88  
Full Screen, 136, 216

## G

GetArea, 356  
GetAttribute, 357  
GetBackColor, 358  
GetBitmap, 243  
GetConnection, 317  
GetCurrentType, 296  
GetCursor, 358  
GetForeColor, 359  
GetGPSInfo, 328  
GetInput, 278  
GetItems, 339  
GetItemsEx, 340  
GetName, 351  
GetProperty, 317  
GetString, 279  
GetTable, 300  
GetText, 359  
GetTypes, 296  
GetValue, 280  
Global User Defined Subroutines and  
    Functions, 224  
Global Variables/Objects, 225  
GoOffline, 329  
GoOnline, 330

**GotFocus**, 226  
Goto Validation, 119  
**GoToScreen**, 360  
Graphical Services, 13  
Greater Than Validation, 116

## H

Header, 374  
**HeaderFont**, 374  
**HeaderValue**, 446  
Height, 242, 248, 294, 374  
Help Menu, 76  
High level, 163  
**HiLo Chart**, 384  
**HiLo OpenClose Chart**, 386  
Host Screen Macro, 181  
Host Transaction Macro, 184  
Hosts, 78  
Hosts List, 132  
Hosts Tree, 173  
**HostScreen Object**, 89  
Hot-Key, 165

## I

Image, 375  
**Image Object**, 89  
Image Resources, 143  
**Image.Bitmap**, 242  
**Image.DisplayMode**, 243  
**Image.GetBitmap**, 243  
**Image.LoadResource**, 244  
**Image.Path**, 244  
Import Items, 65  
Index, 244, 409  
Index Validation, 117  
Initialization Files - Client.ini, 463  
Initialization Files - ERPDialogs.ini, 461  
Initialization Files - GPRS.ini, 462  
Initialization Files - RFgen.ini, 459  
Initialize, 227  
Ins, 431  
InsertItem, 265  
InsertItemEx, 265

Installing Additional Files, 143  
Integer Only Validation, 116  
International Currency Support, 17  
Introduction, 1  
**IpAddress**, 280  
**IsConnected**, 301  
**IsEOF**, 423  
**IsOffline**, 331  
**IsOnline**, 331  
**IsScreen**, 360  
Italic, 240, 247  
Item, 291

## K

KeyPress, 226

## L

**Label Object**, 90  
**Label.Align**, 245  
**Label.Autosize**, 245  
**Label.BackColor1**, 245  
**Label.BackColor2**, 246  
**Label.BackGradient**, 246  
**Label.BorderStyle**, 246  
**Label.Font.Bold**, 247  
**Label.Font.Italic**, 247  
**Label.Font.Size**, 248  
**Label.Font.Underline**, 248  
**Label.ForeColor**, 248  
**Label.Height**, 248  
**Label.Left**, 249  
**Label.Theme**, 249  
**Label.Top**, 250  
**Label.Width**, 250  
Language Set, 16, 136, 216  
Last/Prior Entry Default, 110  
LastError, 338  
Left, 249, 250  
**LegendFont**, 375  
Less Than Validation, 116  
**LField**, 432  
List, 251, 409  
List.AddItem, 251

List.AddItem, 252  
List.Cell, 252  
List.Cell.BackColor1, 253  
List.Cell.BackColor2, 253  
List.Cell.BackGradient, 254  
List.Cell.Bold, 254  
List.Cell.ForeColor, 255  
List.Cell.Italic, 255  
List.Cell.Underline, 255  
List.Cell.Value, 256  
List.Clear, 256  
List.Column, 256  
List.Column.Align, 257  
List.Column.AutoSize, 257  
List.Column.BackColor1, 257  
List.Column.BackColor2, 258  
List.Column.BackGradient, 258  
List.Column.Bold, 259  
List.Column.Caption, 259  
List.Column.DisplayOnly, 259  
List.Column.ForeColor, 260  
List.Column.Format, 260  
List.Column.ImageMode, 261  
List.Column.Italic, 261  
List.Column.ScaleDecimals, 262  
List.Column.TrimSpaces, 262  
List.Column.Underline, 262  
List.Column.Visible, 263  
List.Column.Width, 263  
List.Columns, 263  
List.InsertItem, 265  
List.InsertItemEx, 265  
List.ListCount, 264  
List.ListIndex, 264  
List.PageDown, 266  
List.PageUp, 266  
List.RemoveItem, 267  
List.ScrollDown, 267  
List.ScrollUp, 267  
List.SetColumn, 268  
List.Sorted, 268  
List.Value, 268  
ListBox Object, 90  
ListCount, 264  
ListIndex, 264

Lists (Listing Choices) Default, 111  
Load, 227  
Load Balancing, 194  
Load Balancing Cluster, 15  
Loading the Software, 4  
LoadResource, 244  
Locale, 280  
LocalHostName, 437  
LocalIP, 438  
Locate, 433  
LocateAdd, 434  
LocateDel, 435  
LogError, 281  
LogErrorEx, 281  
Login, 449  
LogMode, 418  
LogOff, 344, 360  
Logoff Session, 201  
LogOn, 344, 361  
Logout, 449  
LostFocus, 227  
Low level, 163

## M

MacroName, 341  
Main Menu, 161  
MakeList, 282, 302, 322, 346  
Making Screen Mapping Work, 161  
MaxRows, 409  
Medium level, 163  
Menu List, 79  
Menu Strip Extensions, 290  
MenuList Object, 91  
Menus, 78  
MenuStrip.AppendItem, 290  
MenuStrip.Clear, 290  
MenuStrip.Count, 291  
MenuStrip.Item, 291  
MenuStrip.RemoveItem, 291  
MenuStrip.RemoveItemByName, 291  
MenuStrip.Reset, 292  
MenuStrip.SetItem, 292  
MenuStrip.Show, 292  
Microsoft Data Access Components, 36

Mobile and Wireless Client, 6  
Mobile Client, 136, 210  
Mobile CnC Services, 14  
Mobile Development Studio Menu Bar, 49  
Mobile Development Studio Overview, 1  
Mobile Development Studio Tools, 77  
Mobile Device Extensions, 327  
Mobile Device Management, 211  
Mobile Devices, 210  
Mobile Enterprise Dashboard, 197  
Mobile Enterprise Services Manager, 193  
Mobile Services Dashboard Overview, 3  
Mobile Services Manager Overview, 3  
Mode, 297  
MoveFirst, 423  
MoveLast, 423  
MoveNext, 423  
MovePrevious, 423  
MoveQueue, 342  
MoveTo, 424  
MsgBox, 282

## N

Name, 338, 419  
NativeError, 310  
Network Application Testing, 54  
NewPage, 351  
Normalize, 410  
Not Condition Validation, 118  
Not Equal To Validation, 116  
Number, 310  
Numeric Only Validation, 116

## O

OnBackup, 227  
OnClose, 443  
OnConnect, 228, 444  
OnCursor, 228  
OnDataArrival, 444  
OnDisconnect, 228  
OnEnter, 229

OnError, 444  
OnEscape, 229  
OnFkey, 229  
OnInRange, 230  
OnLocale, 230  
OnMenu, 230  
OnOutOfRange, 231  
OnReadData, 231  
OnRefresh, 231  
OnReturn, 232  
OnRowColChange, 232  
OnScan, 232  
OnSearch, 232  
OnSendComplete, 444  
OnSendProgress, 444  
OnTimer, 233  
OnUpdate, 233  
OnVoCollect, 233  
OpenResultset, 323  
Options Object, 91  
Orientation, 352

## P

PadInput, 361  
PageDown, 266  
PageNo, 269  
PageUp, 266  
PageWidth, 352  
Param, 419, 424, 454  
Param().Datatype, 455  
Param().Direction, 456  
Param().NumericScale, 456  
Param().Precision, 457  
Param().Size, 457  
Param().Value, 457  
ParamCount, 419, 424, 457  
ParamEx, 419  
ParamName, 420, 425  
Password, 138, 269  
Path, 244  
Pattern Match Validation, 115  
Pattern Not Allowed Validation, 115  
Performance Log, 72  
Pie Chart, 388

Ping, 302  
PingHost, 362  
Platform, 331  
PlaySound, 332  
Plot Chart, 389  
Polar Chart, 391  
Post-Amble, 19  
Pre-Amble, 19  
Prepared, 458  
Print, 295, 353  
Printer Extensions, 348  
Printer Object Extensions, 348  
Printer.Activate, 348  
Printer.Copies, 349  
Printer.EndDoc, 349  
Printer.FontBold, 350  
Printer.FontItalic, 350  
Printer.FontName, 350  
Printer.FontSize, 350  
Printer.FontStrikeThru, 351  
Printer.FontUnderline, 351  
Printer.GetName, 351  
Printer.NewPage, 351  
Printer.Orientation, 352  
Printer.PageWidth, 352  
Printer.Print, 353  
Printer.PrintQuality, 353  
Printer.PrintRaw, 354  
PrinterOff, 332  
PrinterOn, 332  
PrintQuality, 353  
PrintRaw, 354  
Prior Entry Default, 110  
Prompt Specific Extensions, 235  
PromptCount, 284  
PromptNo, 284  
Protocol, 438  
Provider, 138

## Q

QueryType, 446  
Queue, 421  
QueueMacro, 303, 342  
QueueName, 343, 421

QueueOffline, 421  
QueueSeqNo, 422

**R**

Radar Chart, 393  
ReadData, 347  
ReadFile, 303, 333  
ReceiveTimeout, 447  
RedirectDataSource, 325  
Refresh, 295  
Release, 338  
Re-Map, 165  
Remote Application Explorer, 61  
Remote Database Explorer, 63  
Remote Log Viewer, 64  
Remote SQL Explorer, 63  
RemoteHost, 439  
RemoteHostIP, 439  
RemotePort, 439  
RemoveItem, 267, 291  
RemoveItemByName, 291  
Rep, 435  
Replace in Application Scripts, 68  
Reply, 448  
Report All Errors, 18  
Reports Menu, 70  
Request, 448  
Required, 269  
Reset, 292  
ResetConnection, 362  
ResetCursor, 295  
Resources Tab Overview, 133  
Results, 458  
ReturnAllRows, 410  
ReturnValue, 338  
ReverseOff, 295  
ReverseOn, 296  
RFgen Mobile Client, 222  
RFgen Mobile Configuration, 212  
RFgen Sessions window, 199  
RFgen.bas, 78  
RFgen.mdb, 5, 8  
RField, 436  
RollbackTrans, 325, 347

RowCount, 422, 425

## S

SaveBitmap, 326  
Schemald, 426  
Screen Display Extensions, 293  
Screen Mapping, 161  
Screen Mapping Configuration, 166  
Screen Mapping Design Considerations, 162  
Screen Mapping Extensions, 354  
Screen Mapping Keyboard/Special Key Configuration, 165  
Screen Mapping Levels, 163  
Screen Mapping Logon Security Considerations, 164  
Screen Mapping Programming Philosophy, 162  
Screen Mapping Recording Options, 190  
Screen Mapping Runtime Environment Variables, 165  
Screen Mapping Scheduled Downtime, 171  
Screen Mapping System Integrity Considerations, 164  
Screen Mapping Theory of Operation, 161  
Screen Object Extensions, 293  
Screen.ReverseOff, 295  
Screen.ReverseOn, 296  
Screen.Bell, 293  
Screen.Clear, 293  
Screen.ClearEOL, 293  
Screen.ClearEOP, 294  
Screen.DrawLine, 294  
Screen.Height, 294  
Screen.Print, 295  
Screen.Refresh, 295  
Screen.ResetCursor, 295  
Screen.Width, 296  
Script Validation, 73  
Scripts Menu Bar, 123  
ScrollBars, 270  
ScrollDown, 267  
ScrollUp, 267  
SearchList, 399  
SearchList.AddItem, 399  
SearchList.Cell, 399  
SearchList.Cell.BackColor1, 399  
SearchList.Cell.BackColor2, 400  
SearchList.Cell.BackGradient, 400  
SearchList.Cell.Bold, 401  
SearchList.Cell.ForeColor, 401  
SearchList.Cell.Italic, 401  
SearchList.Cell.Underline, 402  
SearchList.Cell.Value, 402  
SearchList.Clear, 402  
SearchList.Column, 402  
SearchList.Column.Align, 403  
SearchList.Column.AutoSize, 403  
SearchList.Column.BackColor1, 403  
SearchList.Column.BackColor2, 404  
SearchList.Column.BackGradient, 404  
SearchList.Column.Bold, 404  
SearchList.Column.Caption, 405  
SearchList.Column.DisplayOnly, 405  
SearchList.Column.ForeColor, 405  
SearchList.Column.Format, 406  
SearchList.Column.ImageMode, 406  
SearchList.Column.Italic, 406  
SearchList.Column.ScaleDecimals, 407  
SearchList.Column.TrimSpaces, 407  
SearchList.Column.Underline, 407  
SearchList.Column.Visible, 408  
SearchList.Column.Width, 408  
SearchList.Columns, 408  
SearchList.Count, 408  
SearchList.Index, 409  
SearchList.List, 409  
SearchList.MaxRows, 409  
SearchList.Normalize, 410  
SearchList.ReturnAllRows, 410  
SearchList.SetColumn, 410  
SearchList.ShowEmptyList, 411  
SearchList.ShowList, 411  
SearchList.SQL, 412  
SearchList.Value, 412  
SelLength, 270

SelStart, 270  
Send, 333  
Send Keys Selection, 176  
Send Message, 200  
SendChar, 284  
SendCommPort, 333  
SendCTRL, 362  
SendCTRLAlt, 363  
SendKey, 284, 363  
SendKeyAlt, 364  
SendMessage, 318  
SendQueue, 304  
SendTable, 304  
SetText, 364  
SendTextAlt, 365  
SendTimeout, 448  
SeqNo, 343  
SeriesColor, 375  
SeriesLabel, 375  
Server Based Extensions, 298  
Server Object Extensions, 298  
Server.CallMacro, 298  
Server.CommandTimeout, 299  
Server.Connect, 299  
Server.Disconnect, 300  
Server.ExecuteSQL, 300  
Server.GetTable, 300  
Server.IsConnected, 301  
Server.MakeList, 302  
Server.Ping, 302  
Server.QueueMacro, 303  
Server.ReadFile, 303  
Server.SendQueue, 304  
Server.SendTable, 304  
Server.SetCredentials, 304  
Server.SetHost, 305  
Server.SetVPN, 305  
Server.SetWAN, 306  
Server.ShowProgress, 306  
Server.SyncApps, 306  
Server.WriteLine, 307  
Session Shutdown Delay, 19  
SessionID, 365  
SessionPwd, 366  
SessionUser, 366  
Set Default, 112  
SetBase, 366  
SetCameraOption, 334  
SetColumn, 268, 410  
SetCommMode, 334  
SetCommPort, 334  
SetCredentials, 304  
SetCursor, 367  
SetDelay, 367  
SetDisplay, 285  
SetFocus, 286  
SetHardRelease, 348  
SetHost, 305  
SetItem, 292  
SetMenu, 287  
SetProperty, 318  
SetSession, 348, 367  
SetTimeout, 368  
SetType, 298  
SetValue, 287  
SetVPN, 305  
SetWAN, 306  
Show, 292, 298  
Show Session, 199  
ShowEmptyList, 411  
ShowList, 287, 411  
ShowProgress, 306  
Signature Object, 92  
SIP.GetCurrentType, 296  
SIP.GetTypes, 296  
SIP.Mode, 297  
SIP.GetType, 298  
SIP.Show, 298  
Size, 240, 248  
Skip the Current Entry Default, 111  
sktClose, 440  
sktConnect, 441  
sktGetData, 441  
sktPeekData, 442  
sktSendData, 443  
Sleep, 288  
SM Host Macros, 78  
SM Object Extensions, 354  
SM Transactions, 78  
SM.BeginTrans, 354

SM.CommitTrans, 355  
SM.Connected, 355  
SM.CurScreen, 355  
SM.FindText, 356  
SM.GetArea, 356  
SM.GetAttribute, 357  
SM.GetBackColor, 358  
SM.GetCursor, 358  
SM.GetForeColor, 359  
SM.GetText, 359  
SM.GoToScreen, 360  
SM.IsScreen, 360  
SM.LogOff, 360  
SM.LogOn, 361  
SM.PadInput, 361  
SM.PingHost, 362  
SM.ResetConnection, 362  
SM.SendCTRL, 362  
SM.SendCTRLAlt, 363  
SM.SendKey, 363  
SM.SendKeyAlt, 364  
SM.SetText, 364  
SM.SetTextAlt, 365  
SM.SessionID, 365  
SM.SessionPwd, 366  
SM.SessionUser, 366  
SM.SetBase, 366  
SM.SetCursor, 367  
SM.SetDelay, 367  
SM.SetSession, 367  
SM.setTimeout, 368  
SM.WaitForCursor, 368  
SM.WaitForCursorMove, 368  
SM.WaitForHost, 369  
SM.WaitForScreen, 369  
SM.WaitForText, 370  
SM.WaitForWrite, 371  
SMBeginTrans, 354  
SMCallMacro, 165  
SMWaitForCursor, 164  
SMWaitForHost, 165  
SMWaitForScreen, 165  
SMWaitForText, 164  
Socket Object, 437  
Socket Services, 13  
Soft Input Panel Extensions, 296  
Solutions Tab Overview, 77  
Sorted, 268  
SP Object Extensions, 449  
SP.CallAction, 449  
SP.CallProc, 450  
SP.CallSelect, 451  
SQL, 412  
SQL Query Testing, 56  
SQL Statement (List) Default, 113  
SQLNum, 289  
StackingBar Chart, 395  
Start and Stop the Server, 193  
Start Menu Macro, 174  
Start Test, 52  
Startup Mode, 136  
State, 440  
Stop Test, 54  
Stored Procedure Download, 44  
Stored Procedure Extensions, 449  
StoredProc.CommandText, 452  
StoredProc.CommandTimeout, 452  
StoredProc.CommandType, 452  
StoredProc.CreateParameter, 453  
StoredProc.Data, 453  
StoredProc.DataSource, 454  
StoredProc.Dict, 454  
StoredProc.Execute, 454  
StoredProc.Param, 454  
StoredProc.Param( ).Datatype, 455  
StoredProc.Param( ).Direction, 456  
StoredProc.Param( ).NumericScale, 456  
StoredProc.Param( ).Precision, 457  
StoredProc.Param( ).Size, 457  
StoredProc.Param( ).Value, 457  
StoredProc.ParamCount, 457  
StoredProc.Prepared, 458  
StoredProc.Results, 458  
String Validation, 117, 118  
Strip Validations, 119  
Surface Chart, 397  
SyncApps, 306  
SYS.ConnectionProperty, 311  
SYS.DeleteProperty, 316  
SYS.DisableTimeout, 316

SYS.EnvironmentProperty, 316  
SYS.GetConnection, 317  
SYS.GetProperty, 317  
SYS.SendMessage, 318  
SYS SetProperty, 318  
SYS.UsePixels, 319  
SYS.UserList, 319  
SYS.ValidateWinUser, 319  
SysErr.AddError, 308, 309  
SysErr.AppName, 308, 309  
SysErr.Clear, 308  
SysErr.Count, 308  
SysErr.Description, 309  
SysError.NativeError, 310  
SysError.Number, 310  
SysError.UserName, 310  
System Date Default, 108  
System Error Extensions, 308  
System Level Extensions, 311  
System Object Extensions, 311  
System Time Default, 108

## T

Tab Width, 23  
TakePicture, 335  
Task List, 73  
TCP/IP Direct, 17  
Telnet, 193, 197  
Telnet Services, 13  
Terminate, 234  
Terminate a session, 201  
Test the Main Menu Macros, 180  
Testing Menu, 51  
Text, 271  
Text Default, 108  
Text Validation, 115  
TextBox Object, 92  
Theme, 249  
Thin Client, 136, 210  
ThreeD, 376  
TimerEnabled, 288  
TimerInterval, 288  
TM Object Extensions, 339  
TM.AbortTrans, 339

TM.CheckStatus, 339  
TM.GetItems, 339  
TM.GetItemsEx, 340  
TM.MacroName, 341  
TM.MoveQueue, 342  
TM.QueueMacro, 342  
TM.QueueName, 343  
TM.SeqNo, 343  
Top, 250, 271  
Transaction Macros, 78  
Transaction Management Dashboard, 204  
Transaction Management Extensions, 339  
Transaction Module Editing Menu Bar, 132  
Transaction Testing, 54  
Transactions, 78  
Transactions List, 130  
Translate Default, 108  
Translations, 158  
Type, 272, 376

## U

Underline, 241, 248  
Undo Save/Delete Action, 69  
Unload, 234  
Upgrade Log, 74  
UseDataSource, 326  
UsePixels, 319  
User, 138, 289  
User Default, 114  
User Management Console, 206  
User Management Console Menus Tab, 207  
User Management Console User Accounts, 206  
UserList, 319  
UserName, 310  
UserProperty, 289  
Users, 78  
Users List, 78  
Utilities Menu, 65

## V

ValField, 272  
ValidateWinUser, 319  
ValTable, 273  
Value, 412  
VB Event Click, 124  
VB Event GotFocus, 124  
VB Event Keypress, 125  
VB Event Load, 125  
VB Event LostFocus, 125  
VB Event OnBackup, 125  
VB Event OnConnect, 125  
VB Event OnCursor, 125  
VB Event OnDisconnect, 125, 127  
VB Event OnEnter, 126  
VB Event OnEscape, 126  
VB Event OnFkey, 126  
VB Event OnLocale, 125, 126, 127  
VB Event OnReadData, 126  
VB Event OnRefresh, 126  
VB Event OnReturn, 126  
VB Event OnRowColChange, 126  
VB Event OnScan, 126  
VB Event OnSearch, 126  
VB Event OnSpeech, 127  
VB Event OnTimer, 127  
VB Event OnUpdate, 127  
VB Event OnVocollect, 127  
VB Event UnLoad, 127  
VBA Declarations, 226  
VBA Events, 226  
VBA Global Variables/Objects, 225

VBA Language Extensions, 235  
VBA Module Editing Menu Bar, 128  
VBA Modules, 78  
VBA Modules List, 128  
View Enterprise Objects, 46  
View Transaction Queues, 74  
Viewing ERP Business Functions, 47  
Visible, 273  
Visual Basic Scripts, 224  
Vocollect Services, 14  
Vocollect Tasks, 159  
VT220 Key Mapping, 172

## W

WaitForCursor, 368  
WaitForCursorMove, 368  
WaitForHost, 369  
WaitForScreen, 369  
WaitForText, 370  
WaitForWrite, 371  
Web Object, 445  
Width, 250, 274, 296, 376  
Win32.bas, 78  
Windows (Graphical) Clients, 4  
Windows Desktop Client, 6, 210  
WriteFile, 307, 336

## X

XML Language Extensions, 23  
XML Services, 14