

DataMax Software Group Inc.
1101 Investment Blvd.
El Dorado Hills, CA 95762
USA

RFgen User Guide

All Editions
RFgen v5.1



Table of Contents

Introduction	2	Configuring RFgen for Oracle JD Edwards	36
Console Overview	2	1. Obtain the RFgen Open Source database for JDE.	36
Client Overview	4	2. Set the RFgen Application Database.	36
Before You Begin	5	Configure the Data Source	36
Mobile Development Studio Overview	5	3. Set the JDE Connector.	37
Server Configuration Overview	6	4. Set the JDE Data Connector	38
Other Documents	6	5. Load / Update JDE BSNFs and Processing Options	39
Configuring the Server	6	Load the JDE BSNFs (Business Functions)	39
Configuring the RFgen Application Database	7	Load JDE Processing Options	40
Configuring Application Event Logs	11	6. Configure RFgen Administrative Setups for JDE	41
Configuring Application Preferences	11	Configuring RFgen to SAP	44
Configuring Application Services	14	Connection Tab (SAP)	46
Configuring Environment Properties	17	Screen Mapping	47
Function Keys - Environmental Properties	21	Adding A New Web Services Connection	48
Configuring Performance Monitoring	22	Configuring Transaction Management Connection	49
Configuring the Scripting Environment	23	Download Enterprise Objects	50
Configuring Source Control Options	24	Downloading ERP Business Functions	51
Configuring System Queues and Tasks	25	Downloading JDE Processing Options	51
Configuring User Access Control	27	Viewing Enterprise Objects	52
Adding or Removing Administrators	27	Viewing ERP Business Functions	53
New Enterprise Connections	28	Viewing JDE Processing Options	54
Configuring Database Connections	28	Utilities	55
Connection Tab	30	SQL Query Testing	55
Pooling	33	Export or Import Mobile Applications	57
Indexing tab	34		
Availability Tab	34		
Configuring an ERP Connection	35		

Table of Contents

Mobile Development Studio	58	Category: Server Connections	86
Resources for Solution Development	59	Category: Mobile Applications	87
Solution Explorer Right-Click Menu	60	Category: Mobile Database	87
Solution Designer/Explorer Menu Bar	61	Category: Additional Files	88
Source Control Options in the Solution		Mobile Themes	89
Designer	63	Mobile Theme Overview	90
Source Control Menu and Object States	63	Mobile Themes Right-Click Menu	90
Find ShelveSet	64	Mobile Themes: Environment	90
Applications Tree	65	Mobile Themes: Controls	93
Applications Right-Click Menu	65	Mobile Themes: Buttons	93
Applications Menu Bar	66	Mobile Themes – Labels	94
Applications	67	Mobile Themes – List Boxes	94
Data Transactions	68	Mobile Themes: Menus	95
To create a Data Transaction macro	68	Mobile Themes: Panels	97
Data Transactions Edit Menu	69	Mobile Themes: TabControl	97
Device Tables	71	Mobile Themes: TextBoxes	98
Images	72	Mobile Themes: Messages	99
Image Menu	72	Mobile Themes: Searches	99
Language Translations	73	Mobile Themes: Soft Keyboard	100
Language Translations	74	Screen Mapping	101
To Translate Strings into Multiple Locales	74	Scripting Modules	102
To Localize Text in Graphical Displays	76	Soft Keyboards	102
Menus and Roles	77	Adding or Removing Users	104
Menus and Sub Menus	78	To add a user	104
To Add Menus	80	To Remove a User	105
Mobile Profiles	83	Voice Applications	106
Category: Device Configuration	84	Voice Tasks	107

Table of Contents

To Import Voice Tasks	107	The PanelList Object	124
Application Tools and Controls	109	Scrollbar Control Property	125
Prompts/Controls Overview	110	The Signature Control	125
TIPS for Placing and Sizing Objects on a Form	111	The TabControl Control	125
The Button Control	112	The TextBox Control	126
The ButtonList Control	112	The TreeView Control	126
The CheckBox Control	112	Control Properties Tab	127
The ComboBox Control	113	Form Control	129
The DateTime Control	113	Form Properties	129
The DataGrid Control	113	Form Group (VBA Events)	131
The DesktopIcons Control	114	Page Control Properties	131
The Frame Control	114	Page Control Properties	131
The HostScreen Control	114	Graphical Control Properties	132
The Image Control	115	Default Property Details	137
The ImageList Control	115	Edit Property Details	144
The Label Control	115	Scripting Your Application	147
The Layout Control	116	Menu Bar	148
How the Layout Control Works	116	Scripts - Global, Application, and Prompt Events	149
Layout Column Properties	117	Scripting Modules Menu Bar	150
How to Use the Layout Control	118	Application Testing	153
Using the Layout Control for Buttons ..	121	Mobile Device Testing	154
The ListBox Control	121	Mobile Session Menu Bar	154
The Map Control	122	Mobile Application Testing Without the Login Screen	155
The MenuList Control	123	Session Properties	155
The Options Control	123	To Debug Code	156
The Panel Control	123	Remote Device Testing	158

Table of Contents

<table style="width: 100%; border-collapse: collapse;"> <tr><td> Remote Session Menu Bar</td><td style="text-align: right;">158</td></tr> <tr><td> Debugging Remote Devices</td><td style="text-align: right;">159</td></tr> <tr><td> Service Requests Testing</td><td style="text-align: right;">159</td></tr> <tr><td> Transaction Queue Testing</td><td style="text-align: right;">159</td></tr> <tr><td>Device Management</td><td style="text-align: right;">162</td></tr> <tr><td> Device Management Options</td><td style="text-align: right;">162</td></tr> <tr><td> Access / Authorized Devices</td><td style="text-align: right;">163</td></tr> <tr><td> To Configure "Online Access"</td><td style="text-align: right;">164</td></tr> <tr><td> To Authorize or Remove Devices</td><td style="text-align: right;">165</td></tr> <tr><td> To authorize a Batch Client</td><td style="text-align: right;">166</td></tr> <tr><td> Application Deployment</td><td style="text-align: right;">168</td></tr> <tr><td> Build CAB File Settings</td><td style="text-align: right;">169</td></tr> <tr><td> Step 1. Build the CAB File</td><td style="text-align: right;">170</td></tr> <tr><td> Step 2. Select a Method for Transferring the CAB files</td><td style="text-align: right;">171</td></tr> <tr><td> Step 3: Wireless Connection Process</td><td style="text-align: right;">171</td></tr> <tr><td> Step 3. Active Sync Connection Process</td><td style="text-align: right;">173</td></tr> <tr><td> Application Explorer</td><td style="text-align: right;">178</td></tr> <tr><td> Database Explorer</td><td style="text-align: right;">178</td></tr> <tr><td> SQL Explorer</td><td style="text-align: right;">179</td></tr> <tr><td> Remote Log Viewer</td><td style="text-align: right;">180</td></tr> <tr><td>Reports</td><td style="text-align: right;">182</td></tr> <tr><td> Application Logs</td><td style="text-align: right;">183</td></tr> <tr><td> Application Log Menu Bar</td><td style="text-align: right;">183</td></tr> <tr><td> To sort errors</td><td style="text-align: right;">183</td></tr> <tr><td> To revert to a prior Application /Event Log</td><td style="text-align: right;">184</td></tr> <tr><td> To delete an Application/Event Log</td><td style="text-align: right;">184</td></tr> </table>	Remote Session Menu Bar	158	Debugging Remote Devices	159	Service Requests Testing	159	Transaction Queue Testing	159	Device Management	162	Device Management Options	162	Access / Authorized Devices	163	To Configure "Online Access"	164	To Authorize or Remove Devices	165	To authorize a Batch Client	166	Application Deployment	168	Build CAB File Settings	169	Step 1. Build the CAB File	170	Step 2. Select a Method for Transferring the CAB files	171	Step 3: Wireless Connection Process	171	Step 3. Active Sync Connection Process	173	Application Explorer	178	Database Explorer	178	SQL Explorer	179	Remote Log Viewer	180	Reports	182	Application Logs	183	Application Log Menu Bar	183	To sort errors	183	To revert to a prior Application /Event Log	184	To delete an Application/Event Log	184	<table style="width: 100%; border-collapse: collapse;"> <tr><td> Application Statistics</td><td style="text-align: right;">184</td></tr> <tr><td> Performance Monitoring</td><td style="text-align: right;">185</td></tr> <tr><td> Task List</td><td style="text-align: right;">186</td></tr> <tr><td> Script Validation</td><td style="text-align: right;">186</td></tr> <tr><td> Queued Transactions</td><td style="text-align: right;">187</td></tr> <tr><td> Upgrade Report</td><td style="text-align: right;">189</td></tr> <tr><td>Screen Mapping</td><td style="text-align: right;">189</td></tr> <tr><td> How to Make Screen Mapping Work</td><td style="text-align: right;">190</td></tr> <tr><td> Theory of Operation</td><td style="text-align: right;">190</td></tr> <tr><td> Programming Philosophy</td><td style="text-align: right;">190</td></tr> <tr><td> Design Considerations for Screen Mapping</td><td style="text-align: right;">191</td></tr> <tr><td> Screen Mapping Levels</td><td style="text-align: right;">191</td></tr> <tr><td> Logon Security Considerations - Screen Mapping</td><td style="text-align: right;">191</td></tr> <tr><td> System Integrity Considerations</td><td style="text-align: right;">192</td></tr> <tr><td> Keyboard/Special Key Configurations</td><td style="text-align: right;">192</td></tr> <tr><td> Runtime Environment Variables</td><td style="text-align: right;">193</td></tr> <tr><td> Configuring the Host Connection</td><td style="text-align: right;">193</td></tr> <tr><td> Availability</td><td style="text-align: right;">196</td></tr> <tr><td> VT220 Key Mapping</td><td style="text-align: right;">198</td></tr> <tr><td> How to record a macro for a screen map</td><td style="text-align: right;">199</td></tr> <tr><td> Building a Start Menu Macro</td><td style="text-align: right;">200</td></tr> <tr><td> The Send Keys Selection</td><td style="text-align: right;">202</td></tr> <tr><td> Start Menu Macro – Record System Sign on</td><td style="text-align: right;">203</td></tr> <tr><td> Start Menu Macro – Identify System Menu</td><td style="text-align: right;">204</td></tr> <tr><td> Start Menu Macro – Record System</td><td style="text-align: right;">205</td></tr> </table>	Application Statistics	184	Performance Monitoring	185	Task List	186	Script Validation	186	Queued Transactions	187	Upgrade Report	189	Screen Mapping	189	How to Make Screen Mapping Work	190	Theory of Operation	190	Programming Philosophy	190	Design Considerations for Screen Mapping	191	Screen Mapping Levels	191	Logon Security Considerations - Screen Mapping	191	System Integrity Considerations	192	Keyboard/Special Key Configurations	192	Runtime Environment Variables	193	Configuring the Host Connection	193	Availability	196	VT220 Key Mapping	198	How to record a macro for a screen map	199	Building a Start Menu Macro	200	The Send Keys Selection	202	Start Menu Macro – Record System Sign on	203	Start Menu Macro – Identify System Menu	204	Start Menu Macro – Record System	205
Remote Session Menu Bar	158																																																																																																						
Debugging Remote Devices	159																																																																																																						
Service Requests Testing	159																																																																																																						
Transaction Queue Testing	159																																																																																																						
Device Management	162																																																																																																						
Device Management Options	162																																																																																																						
Access / Authorized Devices	163																																																																																																						
To Configure "Online Access"	164																																																																																																						
To Authorize or Remove Devices	165																																																																																																						
To authorize a Batch Client	166																																																																																																						
Application Deployment	168																																																																																																						
Build CAB File Settings	169																																																																																																						
Step 1. Build the CAB File	170																																																																																																						
Step 2. Select a Method for Transferring the CAB files	171																																																																																																						
Step 3: Wireless Connection Process	171																																																																																																						
Step 3. Active Sync Connection Process	173																																																																																																						
Application Explorer	178																																																																																																						
Database Explorer	178																																																																																																						
SQL Explorer	179																																																																																																						
Remote Log Viewer	180																																																																																																						
Reports	182																																																																																																						
Application Logs	183																																																																																																						
Application Log Menu Bar	183																																																																																																						
To sort errors	183																																																																																																						
To revert to a prior Application /Event Log	184																																																																																																						
To delete an Application/Event Log	184																																																																																																						
Application Statistics	184																																																																																																						
Performance Monitoring	185																																																																																																						
Task List	186																																																																																																						
Script Validation	186																																																																																																						
Queued Transactions	187																																																																																																						
Upgrade Report	189																																																																																																						
Screen Mapping	189																																																																																																						
How to Make Screen Mapping Work	190																																																																																																						
Theory of Operation	190																																																																																																						
Programming Philosophy	190																																																																																																						
Design Considerations for Screen Mapping	191																																																																																																						
Screen Mapping Levels	191																																																																																																						
Logon Security Considerations - Screen Mapping	191																																																																																																						
System Integrity Considerations	192																																																																																																						
Keyboard/Special Key Configurations	192																																																																																																						
Runtime Environment Variables	193																																																																																																						
Configuring the Host Connection	193																																																																																																						
Availability	196																																																																																																						
VT220 Key Mapping	198																																																																																																						
How to record a macro for a screen map	199																																																																																																						
Building a Start Menu Macro	200																																																																																																						
The Send Keys Selection	202																																																																																																						
Start Menu Macro – Record System Sign on	203																																																																																																						
Start Menu Macro – Identify System Menu	204																																																																																																						
Start Menu Macro – Record System	205																																																																																																						

Table of Contents

Signoff	Activating Android or iOS Mobile Clients	225
Start Menu Macro – Record Error Recovery	To authorize a Batch Client	225
Start Menu Macro – Test Scripts	Mobile Unity Platform Console	227
Building a Host Screen Macro	Accessing the RFgen Server Console (Mobile Unity Platform Console) and Services	227
Host Screen Macro – Go to the Application Screen	To Activate (Authorize) the Server	228
Host Screen Macro – Identify the Application Screen	Activation via the web	229
Host Screen Macro – Return to the Main Menu	System Certificates	230
Host Screen Macro – Test Scripts	Configuring the Server	231
Building a Host Transaction Macro	Configuring the Server	232
Recording Options	Configuring the RFgen Application Database	233
Building a Screen Mapping Application	Mobile Enterprise Dashboard	237
Install RFgen Client Software	Mobile Enterprise Dashboard Menu	238
About RFgen Client Packages	Overview of Dashboard Views	239
Thin Client Overview	RFgen Server Connections	240
Connect and Deploy to RFgen Clients	To Set the Language in Your Dashboard	241
Client Network Control Service	To Change Your Dashboard's Appearance	242
Third-Party Mobile Device Management Tools	To Configure Your Views	242
Auto Upgrade of Clients	To Monitor and Interact with an Active Client	245
Windows Mobile Configuration	To Broadcast a Message	246
RFgen Configuration	To Suspend or Terminate a Session	247
Mobile Settings	Display Options	247
Server Connections Group	Transaction Management Dashboard	249
Local Database Group	Configuring the Transaction Management Dashboard	249
RFgen Windows CE Client		

Table of Contents

	249	To add a user	267
Run on Server	250	To Remove a User	268
RFqueue Status Message	250	User Tools: Export or Import Users	269
To Create or Select Your Data Source	251	To Search Users	270
System DataBase is Access	252	VBA Language Extensions	271
System DataBase is Oracle	253	Database Related Extensions	271
System DataBase is SQL	253	BeginTrans	271
System DataBase is SQLite	254	CommitTrans	272
To Change Your Dashboard's Theme	254	Count	272
To Set the Locale in Your Dashboard	255	Execute	273
User Management Console	256	Extract	273
User Management Console Menu	256	LogOff	274
User Management Console Menu Bar	257	LogOn	274
Menu and Roles Overview	259	MakeList	275
To Add Menus	259	OpenResultset	276
User Management Console Menu Fields	261	RedirectDataSource	277
Mobile App Fields	262	RollbackTrans	278
Sub Menu Fields	263	SaveBitmap	278
Menu Item Selection Tips	263	UseDataSource	279
Search for Menus	264	Prompt-Specific Extensions	279
To Export or Import Menus	264	Align	279
To Remove or Rearrange Menus	265	AutoSize	280
To Remove a Menu and its Form	265	BackColor1	280
To Rearrange Menu Icons	265	BackColor2	280
To Select Multiple Icons	266	BackGradient	281
Launch Option Details	266	BorderStyle	281
Adding or Removing Users	267	Caption	282

Table of Contents

Checked	282	Label.AutoSize	293
Children	283	Label.BackColor1	294
Children.Append	283	Label.BackColor2	294
Children.Count	283	Label.BackGradient	295
Children.Item	284	Label.BorderStyle	295
Children.Remove	284	Label.Font.Bold	295
Cloning	284	Label.Font.Italic	296
Defaults	284	Label.Font.Size	296
DisplayOnly	285	Label.Font.Underline	297
Edits	285	Label.ForeColor	297
ErrMsg	286	Label.Height	297
FieldId	286	Label.Left	298
Font.Bold	286	Label.Theme	298
Font.Italic	287	Label.Top	299
Font.Size	287	Label.Width	299
Font.Underline	287	Layout.GetColSizeType	300
ForeColor	288	Layout.GetColVisible	300
Form.PageNo	288	Layout.GetColWidth	300
Format	289	Layout.GetRowHeight	300
Height	290	Layout.GetRowSizeType	300
Image.Bitmap	290	Layout.SetColSizeType	300
Image.DisplayMode	291	Layout.SetColVisible	300
Image.GetBitmap	291	Layout.SetColWidth	301
Image.LoadResource	292	Layout.SetRowHeight	301
Image.Path	292	Layout.SetRowSizeType	301
Index	293	Layout.SetRowVisible	301
Label.Align	293	Left	301

Table of Contents

List	302	List.Column(x).Format	313
List.AddColSet	302	List.Column(x).ImageHeight	313
List.AddItem	302	List.Column(x).ImageWidth	314
List.AddItemEx	303	List.Column(x).Italic	314
List.Cell	304	List.Column(x).MarginBottom	314
List.Cell(x,y).BackColor1	304	List.Column(x).MarginLeft	314
List.Cell(x,y).BackColor2	304	List.Column(x).MarginRight	315
List.Cell(x,y).BackGradient	305	List.Column(x).MarginTop	315
List.Cell(x,y).Bold	305	List.Column(x).ScaleDecimals	316
List.Cell(x,y).Expanded	306	List.Column(x).Style	316
List.Cell(x,y).ForeColor	306	List.Column(x).TrimSpaces	316
List.Cell(x,y).Indent	307	List.Column(x).Underline	317
List.Cell(x,y).Italic	307	List.Column(x).Visible	317
List.Cell(x,y).Underline	307	List.Column(x).Width	317
List.Cell(x,y).Value	308	List.Columns	318
List.Clear	308	List.Count	318
List.Column(x)	308	List.Data	318
List.Column(x).Align	309	List.Index	319
List.Column(x).AutoSize	309	List.InsertItem	319
List.Column(x).BackColor1	309	List.InsertItemEx	320
List.Column(x).BackColor2	310	List.PageDown	321
List.Column(x).BackGradient	310	List.PageUp	321
List.Column(x).Bold	311	List.ScrollDown	321
List.Column(x).Caption	311	List.ScrollUp	322
List.Column(x).DisplayOnly	311	List.SetColumn	322
List.Column(x).FontSize	312	List.Sorted	323
List.Column(x).ForeColor	312	List.RemoveColSet	323

Table of Contents

List.RemoveItem	323	Tab.CurBtnBackColor1	332
List.RowSelector	323	Tab.CurBtnBackColor2	332
List.SetDefaultColSet	324	Tab.CurBtnBackStyle	333
List.Value(x)	324	Tab.CurBtnBevel	333
Map.CenterMap	324	Tab.CurBtnBorderColor	333
Map.GetAddress	325	Tab.CurBtnBorderStyle	333
Map.GetDirections	325	Tab.CurBtnForeColor	333
Map.GetLatLng	326	Tab.OpenTab	333
Map.PlanRoute	326	Text	333
Map.Zoom	327	Top	334
PageNo	327	TreeView	334
Password	327	Type	335
Required	328	Visible	336
The RFPrompt	328	Width	336
ScrollBars	328	Application-Based Extensions	336
SelLength	329	Balloon	337
SelStart	329	CallForm	337
SetFocus	330	CallMacro	337
Tab	330	CallMenu	338
Tab.AltBtnBackColor1	331	ChangeLoginForm	338
Tab.AltBtnBackColor2	331	ChangeUserPassword	339
Tab.AltBtnBackStyle	331	ClearValues	339
Tab.AltBtnBevel	332	ClientType	339
Tab.AltBtnBorderColor	332	ConnAvailable	339
Tab.AltBtnBorderStyle	332	ExecuteMenuSelection	340
Tab.AltBtnForeColor	332	ExitForm	340
Tab.Caption	332	ExitSession	340

Table of Contents

GetInput	340	MenuStrip Extensions	352
GetString	341	AppendItem	352
GetValue	342	Clear	353
IpAddress	342	Count	353
Locale	343	Enable	353
.LogError	343	Item	354
.LogErrorEx	344	Refresh	354
MakeList	344	RemoveItem	354
MsgBox	345	RemoveItemByName	354
PromptCount	346	Reset	355
PromptNo	346	SetItem	355
Reload	346	Show	355
SendChar	347	Mobile Device Extensions	356
SendKey	347	ClickAndSkipPrompts	356
SetDisplay	348	ClickCoordinates	356
SetFocus	348	EnableGPS	356
SetMenu	349	ForceLocal	357
SetValue	349	GetGPSInfo	357
ShowList	349	GetTimeInfo	359
Sleep	350	GoOffline	359
SQLNum	350	GoOnline	360
Theme	350	Id	360
TimerEnabled	350	IsOffline	361
TimerInterval	351	IsOnline	361
User	351	Platform	361
Update	351	PlaySound	362
UserProperty	352	PrinterOff	362

Table of Contents

PrinterOn	362	Print	373
ReadFile	363	Refresh	373
ScanEnable	363	ResetCursor	373
ScanTrigger	363	ReverseOff	373
Send	364	ReverseOn	374
SendCommPort	364	Width	374
SetCameraOption	364	Soft Input Panel Extensions	374
SetCommPort	365	GetCurrentType	374
Shell	365	GetTypes	374
TakePicture	366	Mode	375
Vibrate	367	SetType	376
WriteFile	367	Show	376
Execute	367	Server-Based Extensions	377
DeviceObject	369	CallMacro	377
Create	369	CommandTimeout	377
Execute	369	Connect	378
LastError	370	Disconnect	378
Name	370	ExecuteSQL	378
Release	371	GeoGetAddress	379
ReturnValue	371	GeoGetDirections	379
Screen Display Extensions	371	GeoGetLatLng	380
Bell	371	GeoGetMap	380
Clear	372	GeoPlanRoute	380
ClearEOL	372	GetTable	381
ClearEOP	372	IsConnected	382
DrawLine	372	MakeList	382
Height	373	Ping	383

Table of Contents

QueueMacro	383	DisableTimeout	397
ReadFile	383	EnvironmentProperty	397
SendQueue	384	GetConnection	397
SendTable	384	GetProperty	398
SetCredentials	385	SelectKeyboard	399
SetHost	385	SendMessage	399
SetVPN	386	SetProcOptionFields	399
SetWAN	386	SetProperty	400
ShowProgress	387	UserList	400
SyncApps	387	ValidateWinUser	401
SyncAppsEx	387	Transaction Management Extensions	401
WriteFile	388	AbortTrans	401
System Error Extensions	388	CheckStatus	402
AddError	388	GetItems	402
AppName	389	GetItemsEx	403
Clear	389	MacroName	404
Count	389	MoveQueue	404
Description	390	QueueMacro	405
DevGUID	390	QueueName	405
IpAddress	390	SqNo	405
NativeError	391	Enterprise Resource Planning Extensions	406
Number	391	BeginTrans	406
UserName	391	CommitTrans	406
System Level Extensions	392	LogOff	407
CheckBoxNoClickFromCode	392	LogOn	407
ConnectionProperty	392	MakeList	408
DeleteProperty	396	ReadData	409

Table of Contents

RollbackTrans	410	GetAttribute	419
SetHardRelease	410	GetBackColor	420
SetSession	410	GetCursor	421
Printer Extensions	411	GetForeColor	421
Activate	411	GetText	422
Copies	411	GoToScreen	422
EndDoc	412	IsScreen	423
FontBold	412	LogOff	423
FontItalic	412	LogOn	424
FontName	413	PadInput	424
FontSize	413	PingHost	425
FontStrikeThru	413	ResetConnection	425
FontUnderline	413	SendCTRL	425
GetName	414	SendCTRLAlt	425
NewPage	414	SendKey	426
Orientation	414	SendKeyAlt	427
PageWidth	415	SendText	427
Print	415	SendTextAlt	427
PrintQuality	415	SessionID	428
PrintRaw	416	SessionPwd	428
Screen Mapping Extensions	417	SessionUser	429
BeginTrans	417	SetBase	429
CommitTrans	417	SetCursor	429
Connected	417	SetDelay	430
CurScreen	418	SetSession	430
FindText	418	SetTimeout	430
GetArea	419	WaitForCursor	431

Table of Contents

WaitForCursorMove	431	HiLo	445
WaitForHost	431	HiLo OpenClose	447
WaitForScreen	432	Pie	449
WaitForText	432	Plot	450
WaitForWrite	433	Polar	452
Chart Object	433	Radar	453
AddDataPoint	434	StackingBar	455
AxesFont	435	Surface	457
AxesLabel	435	SearchList Object	458
AxesTitleFont	435	AddItem	458
BackColor	436	BindControl	459
ForeColor	436	Cell	459
Header	436	Cell(x,y).BackColor1	460
HeaderFont	437	Cell(x,y).BackColor2	460
Height	437	Cell(x,y).BackGradient	460
Image	437	Cell(x,y).Bold	461
LegendFont	438	Cell(x,y).ForeColor	461
SeriesColor	438	Cell(x,y).Italic	461
SeriesLabel	438	Cell(x,y).Underline	462
ThreeD	439	Cell(x,y).Value	462
Type	439	Clear	462
Width	439	Column	463
Chart Examples	439	Column(x).Align	463
Area	440	Column(x).Autosize	463
Bar	441	Column(x).BackColor1	463
Candle	442	Column(x).BackColor2	464
FilledRadar	444	Column(x).BackGradient	464

Table of Contents

Column(x).Bold	465	ReturnAllRows	473
Column(x).Caption	465	SetBind	473
Column(x).DisplayOnly	465	SetColumn	474
Column(x).Expanded	465	ShowEmptyList	475
Column(x).FontSize	466	ShowList	475
Column(x).ForeColor	466	SQL	475
Column(x).Format	466	Value(x)	475
Column(x).ImageHeight	467	Embedded Procedure Object	476
Column(x).ImageWidth	467	Embedded Procedures	476
Column(x).Indent	468	Embedded Procedure Properties and Methods	479
Column(x).Italic	468	Clear	479
Column(x).MarginBottom	468	ColumnCount	479
Column(x).MarginLeft	468	ColumnName	479
Column(x).MarginRight	468	DataSource	480
Column(x).MarginTop	469	DebugLog	480
Column(x).ScaleDecimals	469	DisableParam	480
Column(x).Style	469	Execute	481
Column(x).TrimSpaces	470	ExecuteMethod	481
Column(x).Underline	470	LogMode	481
Column(x).Visible	470	Name	481
Column(x).Width	470	Param	482
Columns	471	ParamCount	482
Count	471	ParamEx	482
Index	471	ParamName	483
List	472	Queue	483
MaxRows	472	QueueName	484
Normalize	472		

Table of Contents

QueueOffline	484	IsEOF	493
QueueSeqNo	484	MoveFirst	493
RowCount	485	MoveLast	493
SetKeyFields	485	MoveNext	493
JDE Processing Option	486	MovePrevious	493
JDEProcOpt	486	MoveTo	494
Count	486	Param	494
ParamName	487	ParamCount	494
ProgramId	487	ParamName	495
TemplateId	488	RowCount	495
Value	488	SchemaId	496
Version	489	Dynamic Array Extensions	496
Smtp Extension	490	DCount	496
Attach	490	Del	497
Bcc	490	Ext	499
Cc	490	FixLeft	500
Clear	490	FixRight	500
From	491	Ins	501
Host	491	LField	502
Message	491	Locate	503
Port	491	LocateAdd	504
Send	491	LocateDel	504
Subject	492	Rep	505
To	492	RField	506
DataRecord Object	492	Socket Object	506
AddNew	493	BytesReceived	507
Clear	493	LocalHostName	507

Table of Contents

LocalIP	507	Request	518
Protocol	508	SendTimeout	519
RemoteHost	508	Execute	519
RemoteHostIP	508	Login	519
RemotePort	509	Logout	519
State	509	Stored Procedure Extensions	520
sktClose	510	CallAction (not used with Sybase)	520
sktConnect	510	CallProc (must be used with Sybase)	520
sktGetData	511	CallSelect (not used with Sybase)	522
sktPeekData	512	Database Stored Procedure Object	522
sktSendData	513	CommandText	523
OnClose	513	CommandTimeout	523
OnConnect	514	CommandType	523
OnDataArrival	514	CreateParameter	524
OnError	514	Database Stored Procedure Object	524
OnSendComplete	514	DataSource	524
OnSendProgress	515	Dict	525
Web Object	515	Execute	525
ConnectTimeout	515	Param Object	525
DataSource	515	Param().Datatype	526
EndPoint	516	Param().Direction	527
HeaderValue	516	Param().NumericScale	528
InParam	516	Param().Precision	528
OutParam	517	Param().Size	528
QueryType	517	Param().Value	529
ReceiveTimeout	517	ParamCount	529
Reply	518	Prepared	529

Table of Contents

Results	529	Asc Function	542
Initialization Files	530	Assign Instruction	542
RFgen.ini	530	Assign Operators	543
LoadBalanced Server Parameters	530	Atn Function	544
Routing Queues Between Servers and Mobile Clients	530	Attribute Definintion/Statement	544
Purge Data From Reports	531	Beep Instruction	545
Override Transaction Macro Internal Timeout Limit	531	Begin Dialog Definition	546
Reset Transaction Queues	531	Boolean Data Type	546
Reset Pooled Connections	532	Byte Data Type	546
Make DB2 Tables Visible	532	Call Instruction	547
Check JDE Connection Before Execut- ing a Business Function	532	CallByName Instruction	547
Reset JDE Child Processes	533	CallersLine Function	548
ERPDialogs.ini	533	CancelButton Dialog Item Definition	548
GPRS.ini	534	CBool Function	549
Client.ini	535	CByte Function	549
WWB.NET: Overview	536	CChar Function	549
#Language Special Comment	537	CDate Function	550
#Reference Special Comment	538	CDbl Function	550
#Uses Special Comment	539	CDec Function	550
AboutWinWrapBasic Instruction	539	Char Data Type	551
Abs Function	540	ChDir Instruction	551
AddHandler Instruction	540	ChDrive Instruction	551
AddressOf Operator	541	CheckBox Dialog Item Definition	552
Any Data Type	541	Choose Function	553
AppActivate Instruction	542	Ch' Function	553
		CIInt Function	553
		Class Module	554

Table of Contents

Class_Initialize Sub	555	DDEExecute Instruction	566
Class_Terminate Sub	555	DDEInitiate Function	567
Clipboard Instruction/Function	555	DDEPoke Instruction	567
CLng Function	556	DDEReques' Function	568
CObj Function	556	DDETerninate Instruction	568
Code Module	556	DDETerninateAll Instruction	568
ComboBox Dialog Item Definition	557	Debug Object	569
Comman' Function	558	Decimal Data Type	569
Const Definition	558	Declare Definition	570
Cos Function	559	Decode64 Function	571
CreateObject Function	559	Decrypt64 Function	571
CSByte Function	560	Delegate Definition	572
CShort Function	560	DeleteSetting Instruction	572
CSng Function	560	Dialog Instruction/Function	573
CStr Function	561	DialogFunc Prototype	574
CurDi' Function	561	Dim Definition	575
CType Function	561	Di' Function	576
CUInt Function	562	DirectCast Function	576
CULng Function	562	DlgControlId Function	577
CUShort Function	562	DlgCount Function	578
Date Data Type	563	DlgEnable Instruction/Function	578
DateAdd Function	563	DlgEnd Instruction	579
DateDiff Function	564	DlgFocus Instruction/Function	580
DatePart Function	564	DlgListBoxArray Instruction/Function	581
DateSerial Function	565	DlgName Function	582
DateValue Function	565	DlgNumber Function	582
Day Function	566	DlgSetPicture Instruction	583

Table of Contents

DlgText Instruction/Function	584	FileLen Function	600
DlgType Function	585	FileOpen Instruction	600
DlgValue Instruction/Function	586	Fix Function	601
DlgVisible Instruction/Function	587	For Statement	601
Do Statement	588	For Each Statement	602
DoEvents Instruction	588	Forma' Function	602
Double Data Type	589	Format Predefined Date	603
DropListBox Dialog Item Definition	589	Format Predefined Number	603
Encode64 Function	590	Format User Defined Date	604
Encrypt64 Function	590	Format User Defined Number	605
End Instruction	591	Format User Defined Text	606
Enum Definition	591	FreeFile Function	606
Environ Function	592	Friend Keyword	607
EOF Function	592	Function Definition	607
Erase Instruction	593	Get Instruction	608
Err Object	593	GetAllSettings Function	608
Error Instruction	594	GetAttr Function	609
ErrorToString Function	595	GetChar Function	609
Eval Function	595	GetFilePath' Function	610
Event Definition	596	GetObject Function	610
Exit Instruction	596	GetLocale Function	611
Exp Function	597	GetSetting Function	611
False Keyword	598	GetType Operator	612
FileAttr Function	598	Goto Instruction	612
FileClose Instruction	598	GroupBox Dialog Item Definition	612
FileCopy Instruction	599	He' Function	613
FileDateTime Function	599	Hour Function	613

Table of Contents

If Statement	614
IIf Function	615
Imports Definition	615
Input Instruction	615
InputBo' Function	616
InputStrin' Function	616
InStr Function	617
InStrRev Function	617
Int Function	618
Integer Data Type	618
Is Operator	618
IsArray Function	619
IsDate Function	619
IsDBNull Function	619
IsError Function	620
IsNot Operator	620
IsNothing Function	621
IsNumeric Function	621
IsReference Function	622
Join Function	622
KeyName Function	622
Kill Instruction	623
LBound Function	623
LCas' Function	624
Lef' Function	624
Len Function	624
Like Operator	625
LineInput Function	625
ListBox Dialog Item Definition	626
Loc Function	627
Lock Instruction	627
LOF Function	628
Log Function	628
Long Data Type	629
LSe' Function	629
LTri' Function	629
MacroCheck Function	629
MacroCheckThis Function	630
MacroDi' Function	630
MacroRun Instruction	631
MacroRunThis Instruction	631
Main Sub	632
Me Object	632
Mi' Function/Assignment	632
Minute Function	633
MkDir Instruction	633
ModuleLoad Function	634
ModuleLoadThis Function	634
Month Function	635
MonthName Function	635
MsgBox Instruction/Function	636
MultiListBox Dialog Item Definition	637
New Operator	638
Nothing Keyword	638

Table of Contents

Now Function	638
Object Data Type	639
Object Module	639
Object_Initialize Sub	640
Object_Terminate Sub	640
Oc' Function	641
OKButton Dialog Item Definition	641
On Error Instruction	642
Operators	642
Option Definition	644
OptionButton Dialog Item Definition	645
OptionGroup Dialog Item Definition	645
Picture Dialog Item Definition	646
Print Instruction	647
PrintLine Instruction	647
Private Definition	648
Private Keyword	648
Property Definition	649
Public Definition	649
Public Keyword	650
PushButton Dialog Item Definition	650
Put Instruction	651
QBColor Function	652
RaiseEvent Instruction	652
Randomize Instruction	653
ReDim Instruction	653
Rem Instruction	654
RemoveHandler Instruction	654
Rename Instruction	655
Replac' Function	655
Reset Instruction	656
Resume Instruction	656
Return Instruction	657
RGB Function	657
Righ' Function	658
RmDir Instruction	658
Rnd Function	658
Round Function	659
RSe' Function	659
RTri' Function	660
SaveSetting Instruction	660
Second Function	660
Seek Instruction	661
Seek Function	661
Select Case Statement	662
SendKeys Instruction	663
SetAttr Instruction	664
SetLocale Instruction	665
Sgn Function	665
Shell Function	666
SByte Data Type	666
Short Data Type	666
ShowPopupMenu Function	667
Sin Function	667

Table of Contents

Single Data Type	668	TypeOf Operator	681
Spac' Function	668	UBound Function	681
Split Function	668	UCas' Function	682
Sqr Function	669	UInteger Data Type	682
Static Definition	669	ULong Data Type	682
Stop Instruction	669	Unlock Instruction	683
St' Function	670	UShort Data Type	683
StrCom' Function	670	Val Function	684
StrCon' Function	671	Using Statement	684
StrDu' Function	672	VarType Function	684
String Data Type	672	VbTypeName Function	686
StrRevers' Function	673	Wait Instruction	686
Structure Definition	673	Weekday Function	686
Sub Definition	674	WeekdayName Function	687
SystemTypeName Function	675	While Statement	687
Tan Function	675	Win16 Keyword	688
Text Dialog Item Definition	675	Win32 Keyword	688
TextBox Dialog Item Definition	676	Win64 Keyword	688
Timer Function	677	With Statement	688
TimeSerial Function	677	WithEvents Definition	688
TimeValue Function	678	Write Instruction	689
Throw Instruction/Function	678	WriteLine Instruction	689
Tri' Function	678	Year Function	690
True Keyword	679	Objects Overview	690
Try Statement	679	Error List	691
TryCast Function	680	Terms	693
TypeName Function	680		

Introduction

RFgen's Mobile Unity Platform™ provides software programs that enables database management systems and Enterprise Resource Planning (ERP) system users to:

- Easily develop advanced Radio Frequency data collection (RFDC) applications
- Deploy RFDC applications to mobile devices
- Collect and transact data while connected to or disconnected from the network
- Manage and monitor the server, ERP connections, RFgen clients/users, and transaction processes.

RFgen Mobile Development Studio provides the software tools and platform your developers can use to design, test, and deploy (or customize) mobile applications.

Before you can begin using the Mobile Unity Platform or Mobile Development Studio, you will need to configure the RFgen server, database application, database connections and applicable ERP connections.

Console Overview

The RFgen's Mobile Unity Platform, by default installs the following management consoles and dashboards:

- Mobile Unity Platform Console
- Mobile Enterprise Dashboard
- Transaction Management Dashboard
- User Management Console

These different consoles/dashboards allow varying levels of access to the RFgen server, services and sessions.

Mobile Unity Platform Console



The **Mobile Unity Platform Console** is primarily used to start and stop server services, and configure server services, databases, ERP connections, environment settings, and track server status information. It also shows the license or authorization status of the server and clients. The Maintenance Mode feature enables you to prevent new users from connecting to the server if you plan on bringing the server down for maintenance and are waiting until all clients have disconnected from the server.

Mobile Enterprise Dashboard

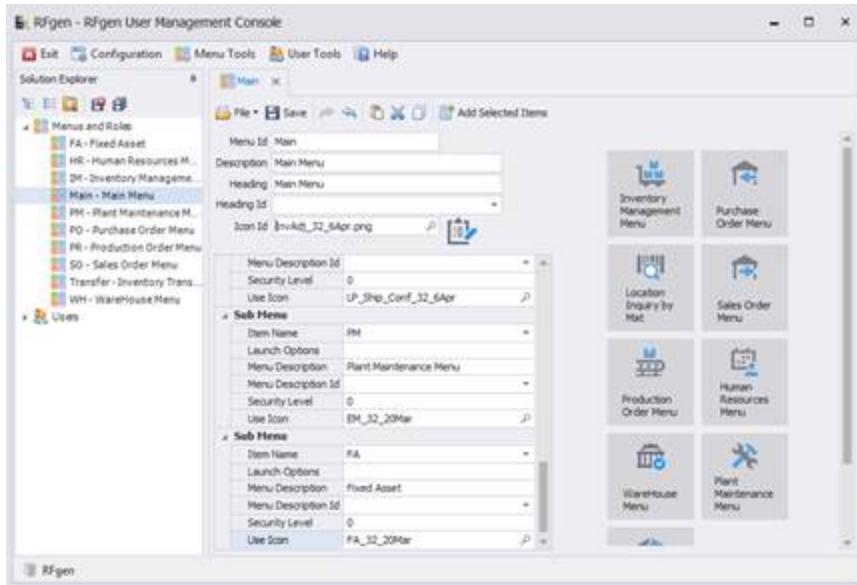
A screenshot of the RFgen Mobile Enterprise Dashboard. The top menu includes Exit, Configuration, Display Options, Find Session, and Help. Below the menu are four buttons: Mobile Users, Voice Users, System Tasks, and Shared Resources. A search bar with "Enter text to search..." and "Find" and "Clear" buttons is present. A table lists user activity: User Id (sam), User Name (Samual User), Type (Mobile Session), App Id (FIMCC0200), App Description (Cycle Count), Task Id (21312), and Last Activity (8/28/2019 10:24:4). A detailed view for user "sam" shows Local Resources with a table for SAP: Source (SAP), State (Active), Created At (8/28/2019 10:22:24 AM), Last Activity (8/28/2019 10:24:31 AM), Task Id (21312), and Average Execution Time. The bottom of the screen shows the URL "myserver.rfgen.com".

The **Mobile Enterprise Dashboard**, enables you to monitor and manage remote sessions running under the server. This includes:

- **Mobile User** sessions
- **Voice User** sessions for users of voice applications
- **System Tasks** sessions by the system or data source (i.e. SAP)
- **Shared Resources** for viewing pooled sessions (where you have license pooling setup for an ERP)

Through this dashboard, the administrator can perform tasks such as joining a session, send messages to a user, and suspending or terminating their session. Specifics about a session can also be collected. For example, if you want to see how long a task is taking to execute, you can look for this information on the System Tasks tab.

User Management Console



The **User Management Console** – Installation of this component is optional. It provides tools to add, remove and monitor users, and create and modify the menus that enable users to access applications. It also allows you to observe or control another user's work being done via the client. You can also temporarily stop one or many users and permanently shutting down one or many users provides complete control over the RFgen network.

Client Overview

RFgen provides client software which must be installed to the device prior to deploying your mobile application to the device.

The four basic device platforms are [Android](#), [Apple iOS](#) (but not Macintosh or Apple computer platforms), [Windows desktop systems](#), and the compact embedded, [Windows CE](#).

For more details on installing or upgrading these software packages, refer to the Installation and Upgrade Guide.

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.

Mobile Development Studio Overview

The **RFgen Mobile Development Studio** contains the tools, resources and functions to develop mobile applications that can scan barcodes and transact data while connected to the RFgen server or scan and transact data while offline (and upload changes to the back end upon reconnection.)

Basic Implementation Process

1. [Configure the RFgen Server.](#) Once you've installed your Mobile Unity Platform and/or Mobile Development Studio (Dev Studio), you'll need to connect to an RFgen open source database or setup a new database for the storage of mobile applications. You'll also need to connect with your ERP data source, and applicable transaction system or web services and databases so validations and changes can be communicated and transacted. And, depending on your ERP, you may need to download specific business functions, tables and /or objects that also support and regulate updates to your ERP data. If you plan on developing applications, you'll need to configure your development environment and user preferences as well.
2. [Design your applications.](#) The Studio is based on and compatible with Microsoft Visual Basic for Applications and also uses COM and .NET for scripting purposes. The Solution Designer in the Studio provides graphical user interface objects (Textbox control, Panellists etc), properties, code modules for logins, data entries, retrieval of data, data validations etc.) and many other development features one would need to design and script an application. If you plan on localizing your applications, RFgen also provides Google-based translations for your strings and facilities to import and store images you may want to add to your apps.
3. [Setup menus](#) for each application. Remember to setup at least one user and assign the user to the menus. This enables you to test your applications in the Dev Studio and organize which users will be allowed access to specific applications or groups of applications.
4. [Test and debug your applications.](#) You can test your application in the Test tab. This module also include debugging features.
5. [Create Mobile Profiles.](#) A mobile profile is a collection of applications, menus, and client configuration settings that enables the client to function as a Thin or Fat (Batch client). If you are creating a profile for Windows CE devices, you will also need to create CAB files. Android and iOS do not need CAB files.
6. [Install RFgen client software.](#) The RFgen Client software enables communication with the server. This software is needed for device-to-server communication purposes. You download it from the RFgen portal or Apple Apps or Google Apps site and install it to your client(s).

7. [Connect and deploy Mobile Profiles](#). Once the RFgen Client software is installed, you connect with the server which upon successful connection will download the requested profile to your client.

Note: If an Offline (Batch) Profile is installed, manual authorization of Offline (Batch) clients is required regardless of whether "Restrict Online Access" was setup.

Server Configuration Overview

- Configure the RFgen Application Database
- Configure Your Server Environment
- Connect to an ERP, host system, or web service
- Connect and configure your data source
- Connect and configure to a Transaction Manager

After you have installed the Mobile Development Studio or Mobile Unity Platform Console, configured application data source(s), and added new Enterprise connections, the following steps are generally implemented for ODBC or 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. The Database transaction table field definitions are downloaded to RFgen. (See **Enterprise Connections > Download Enterprise Objects**.)

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.

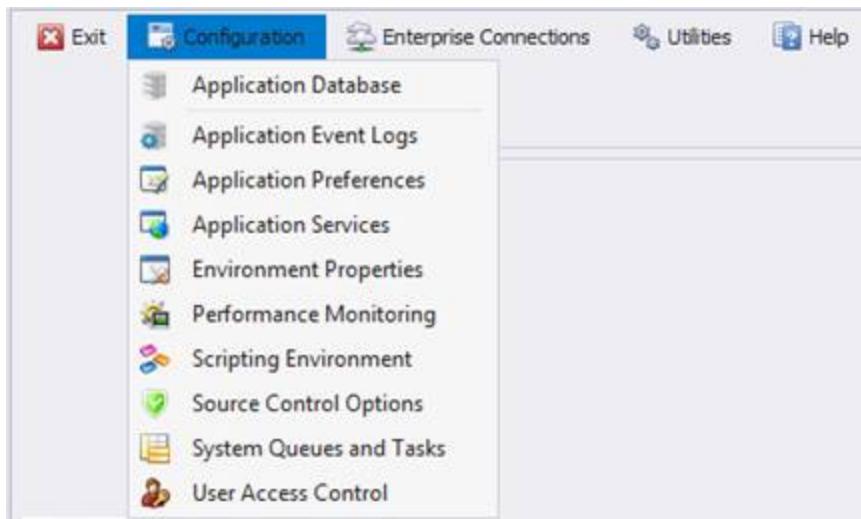
3. Data entry Applications are made for each transaction table. Pre-programmed data properties such as 'defaults', 'validations/edits', 'table validations', etc. are added, as required.

Next Steps: Setup Users and Menus in Mobile Development Studio)

Other Documents

For information on how to install RFgen products/packages, refer to the **RFgen Installation and Upgrade Guide**. This guide is available from the RFgen web portal or your Windows system Start > RFgen v51 > Documentation folder.

Configuring the Server



The Configuration Menu functions are as follows:

The **Application Database** configures the database storing all the solution objects.

The **Application Event Logs** database configuration is used to log events and reset connections for specific databases.

The **Application Preferences** is for user interface themes and locale, default design mode, and scripting.

The **Application Services** screen include port configurations, load balancing, server run mode, service credentials, security and controller failover monitoring.

The **Environment Properties** settings including system options, timeout values, Pre/Post-amble scanner defaults, embedded graphical menu options, function key options, system properties and the device authorization scheme.

The **Performance Monitoring** sets thresholds for backend communication for debugging purposes.

The **Scripting Environment** settings to allow direct access to Active Directory Objects (ADO) and XML language extension parameters and have them globally loaded into BAS files.

The **Source Control Options** allow developers to use a third-party source control product if its plug-in is supported.

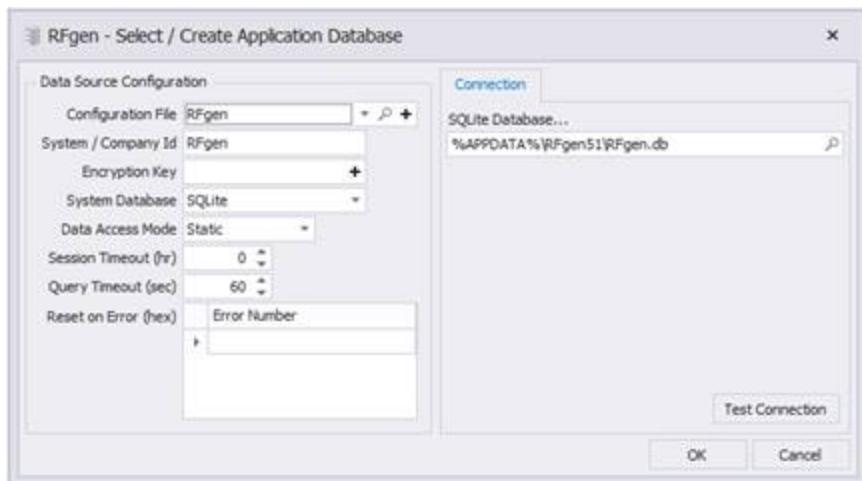
The **System Queues and Tasks** configure all queues and timed event macro executions.

The **User Access Control** console allows you to authenticate connections between the RFgen Server and the **Mobile Platform Unity Management Console**, and **User Management Console**.

Configuring the RFgen Application Database

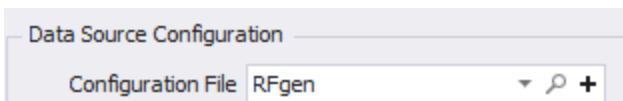
In order to provide a database for storing and maintaining your RFgen Mobile Applications (which help run your Transactions), you need to connect to a database application/server/system to your RFgen server/system.

From the Mobile Development Studio or Mobile Unity Platform Console: Click on **Configuration > Application Database**.



Data Source Configuration Values

By default, a Configuration File called 'RFgen.rfc', defines the profile of the solution database, as shown below.



If you need to change the rfc file or select a different rfc file, you can use the list, search or plus (+) icons to browse to the %APPDATA%\ProgramData\RFgen51 folder.

An '**Encryption Key**' entry provides users the ability to encrypt their Application Database. This feature 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 **Encryption Key** textbox, and click the + 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.

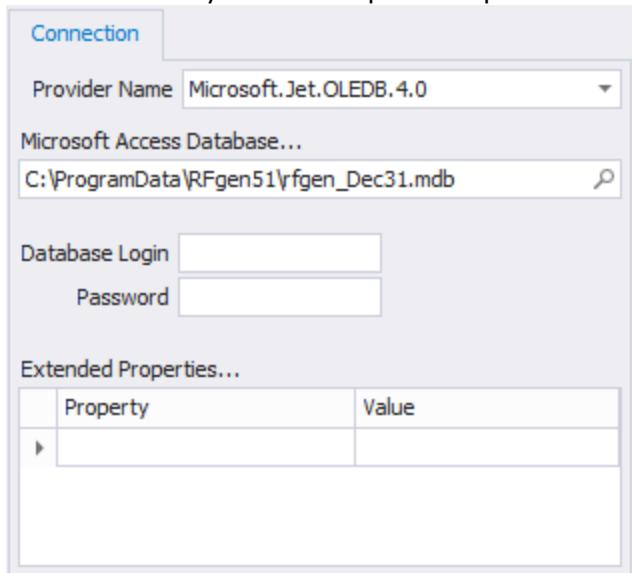
Data Access Mode sets the cursor to either Static or Dynamic when retrieving data from the database. Usually, Static is best because it is fast and safe. However, if you have a database like Pervasive that will actually make a copy of the data from the database system to the RFgen system when using a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

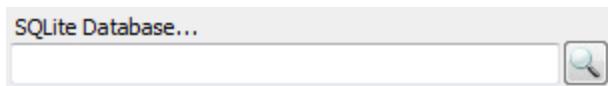
In the case of Access most of these fields are not necessary.

Property	Value
VCharNull	1
distribtx	0
enlist	FALSE

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.

Property	Value

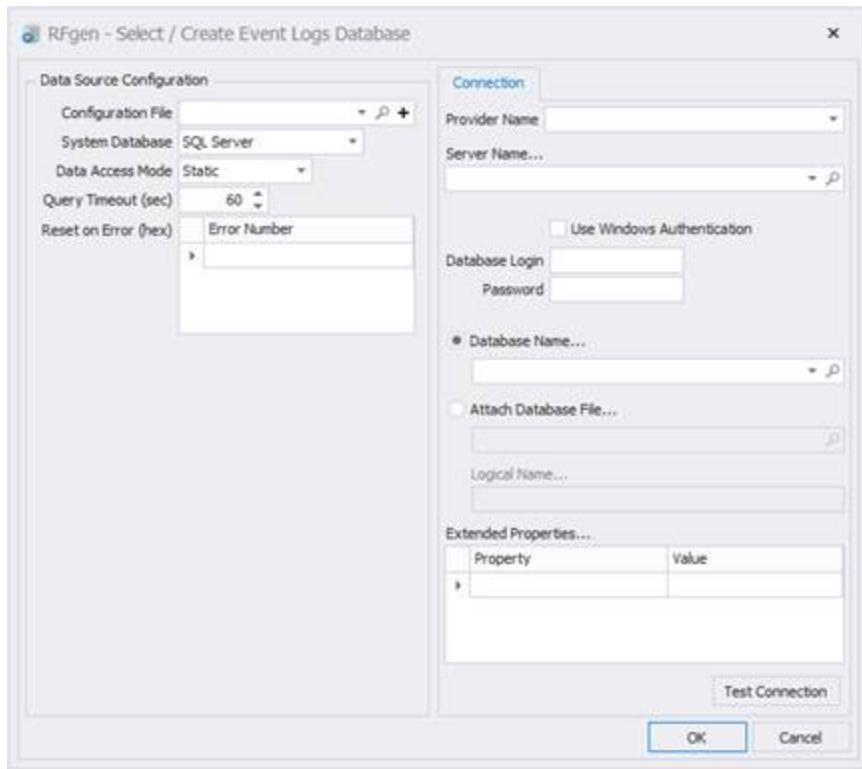
For **SQL** Server specify the Provider Name, Server Name and Database Name. The Use Windows Authentication option will take advantage of the Active Directory when connecting to the database. If you want to connect directly to the MDF file itself, specify the Attach Database File option and locate the 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.

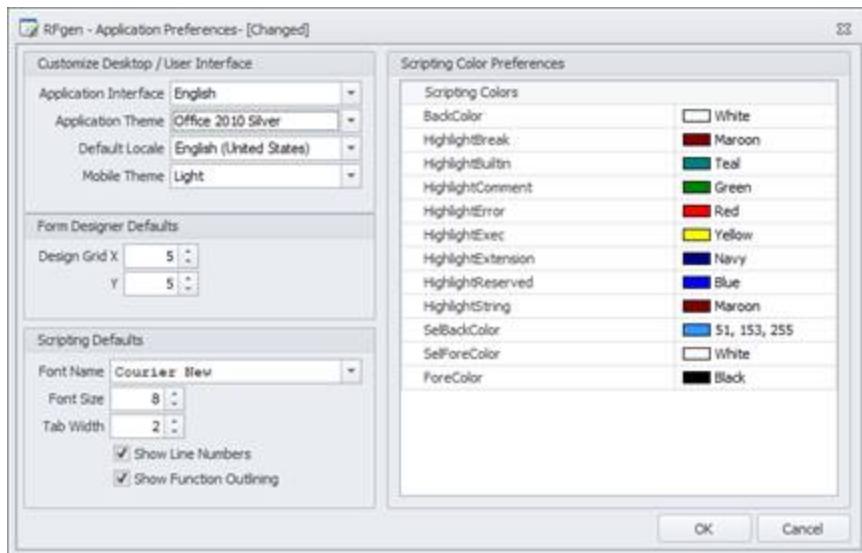
Finally click on the **Test Connection** button to verify connectivity. If the database has not already been setup to support the solution tables they will be created at this time. Clicking the Save Changes button will also create what is necessary but won't test the connection. Either button will also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

Configuring Application Event Logs

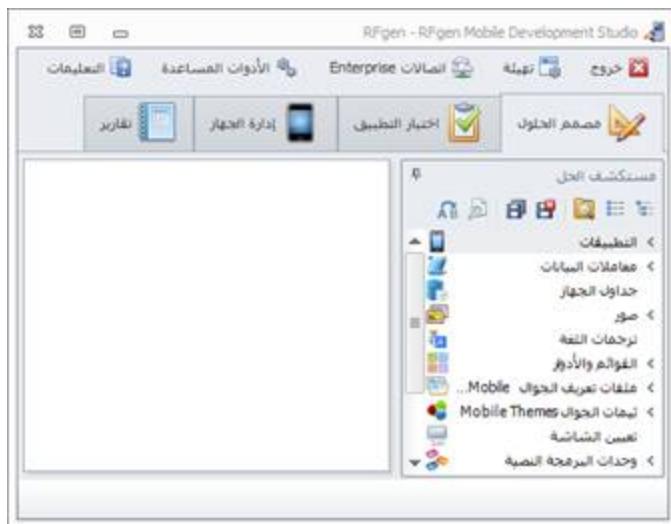


In the same way that the Application Database configuration window uses a **Configuration File** to pre-fill all the other settings, the Application Event Log configuration uses the same format. Specify any database and the event log tables will be created automatically.

Configuring Application Preferences



Customize Desktop/User Interface

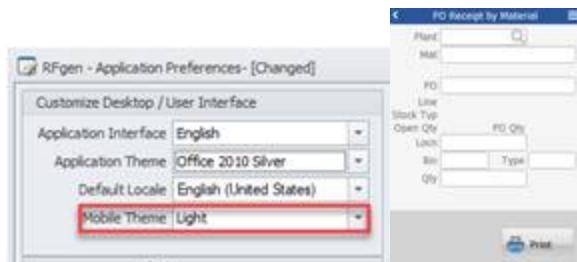


The **Application Interface** changes language the RFgen Mobile Development Studio, RFgen Server and its consoles into the language defined in this field. The options are: Arabic, Chinese, French, Japanese and Spanish.



The **Application Theme** option changes the coloring theme your RFgen Mobile Development Studio windows. For example, a black versus a light background as shown above.

The **Default Locale** is no longer used. To translate your mobile applications into a specific locale, refer to the Solution Explorer > Language Translations.



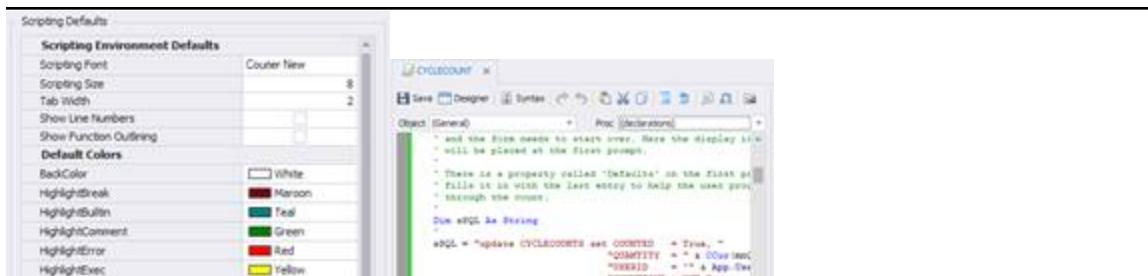
The **Mobile Theme** option sets the default theme that is used for the design and presentation of a Mobile Application. (The theme you create or modify is under Solution Explorer > Mobile Themes.)

Form Designer Defaults

The Form Designer Defaults apply to the form which you use in the **Solution Explorer > Applications >** designer section to create the user interface.

Design Grid X and Y provide the default, X and Y width and height in pixels for the Form, that is, the area to be occupied by controls (i.e. buttons, layout controls, and text etc.). Note that if your X and Y settings are too low (ie. X = 40), the prompt may not "stick" when you try to drag it onto your form in the Designer.

Scripting Defaults

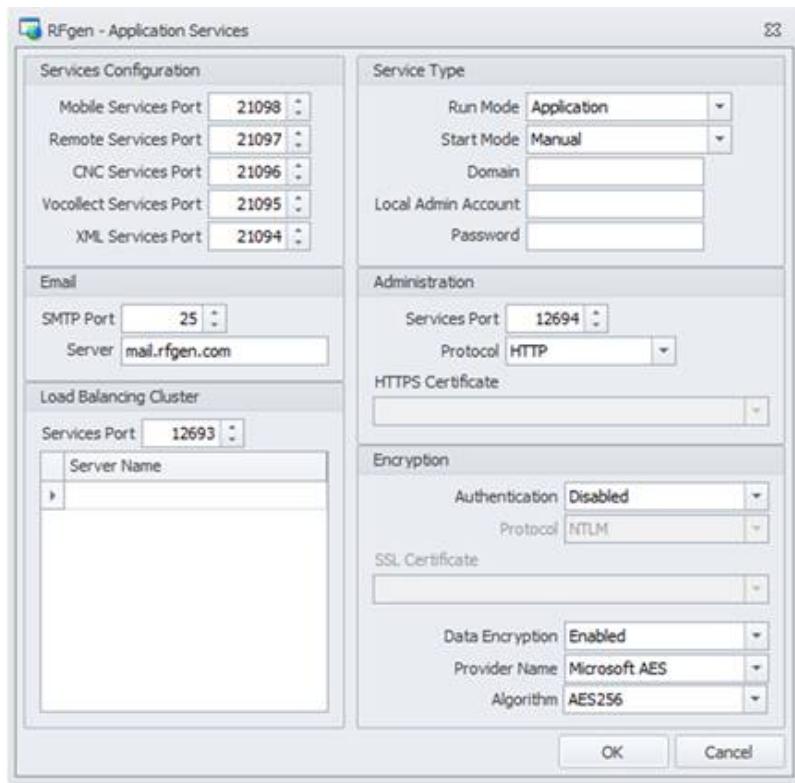


The **Scripting Font**, **Scripting Size**, and **Tab Width** values affect the code windows.

The **Show Line Numbers** and **Show Function Outlining** checkboxes turn on these settings.

The **Default Colors** allow you to customize the coloring of various elements/types of scripts. For example, you can color all text strings as "Maroon".

Configuring Application Services



The **Mobile 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 **CNC Services Port** enables the server to perform actions on the mobile client when the device is not in its cradle. The Client Network Control service enables the server to listen for client requests and synchronize information between the server and the client. 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 / 3rd 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.

Email

There is a language extension called SMTP that can be used to send email under any circumstances required. The **Server** and **Services Port** properties here can be defaulted and automatically used by the SMPT object or left blank and entered as part of the scripting.

Load Balancing Cluster

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 Vocollect connections are not load balanced.

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 Vocollect connections are not load balanced.

Service Type

The recommended configuration for running RFgen as a service is to use a local administrator to the PC where RFgen is installed.

The **Run Mode** lets you choose whether RFgen runs as an Application or a Service. **Application mode** runs

a single instance of the RFgen server as a Windows program after a user logs into the RFgen server and the server service is launched (automatically or manually) from the desktop. **Service mode** runs the RFgen server as a background process and does not interact with the desktop. The start/stop of the RFgen server is not dependent on a user logging into Windows and having it launched from a desktop.

If you choose **Application** and the Start Mode of **Automatic**, RFgen will start RFgen as an application when a user logs in to the Windows desktop.

If you choose **Service** and the Start Mode of **Automatic**, RFgen services will be configured in the Window's services list. This means the RFgen server services will start automatically when the Windows server is started.

If you choose a **Manual** start for Application or Service, neither will start until you start them manually.

The **Domain** field which contains the name of the local PC.

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.

Administration

The Services Port typically uses 12694. This port is used for communication between the server and the *Mobile Enterprise Dashboard*. You can set the communication protocol as HTTP or HTTPS. If using HTTPS, the certificate for HTTPS needs to be installed to the local computer certificate store in order for it to appear under HTTPS Certificate box.

Encryption

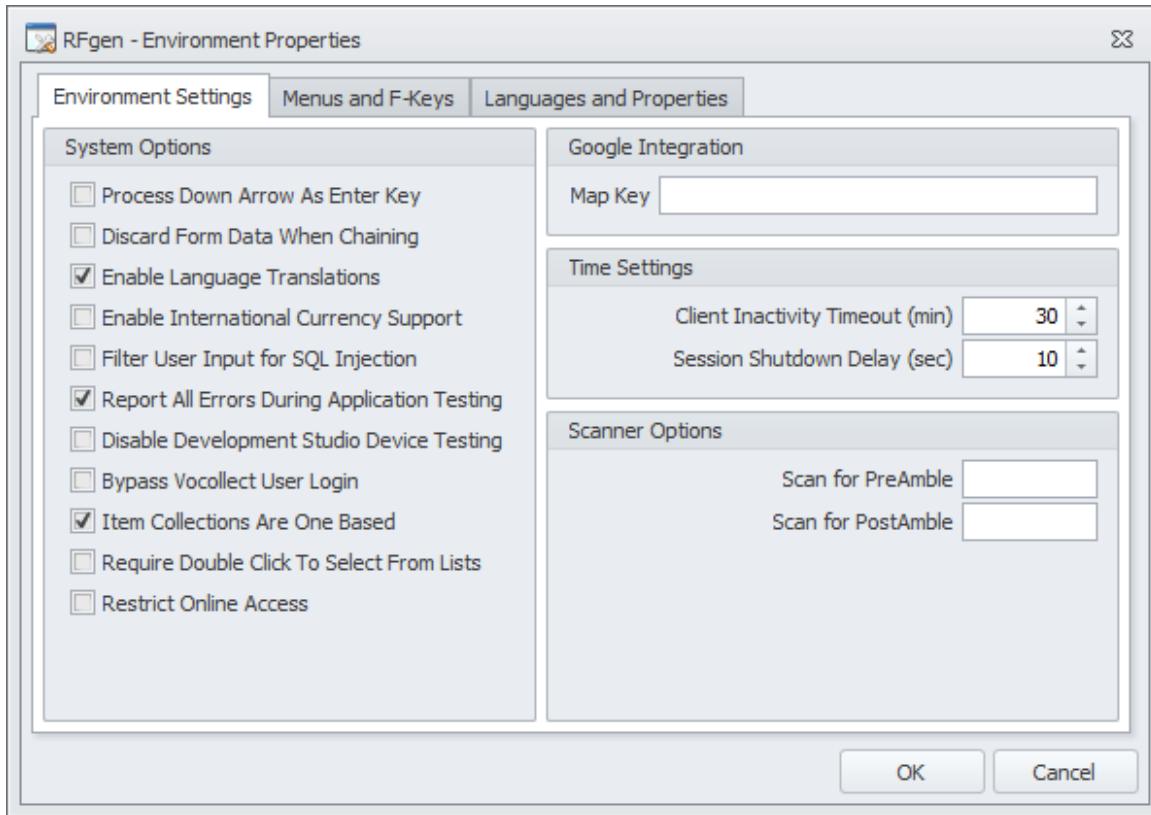
The two categories are device authentication and data encryption settings.

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. For the **Provider Name**, 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.

Configuring Environment Properties



Environment Tab - System Options

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, dis-associate, 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 an application 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.

List Items Collection is One Based tells RFgen, whether to begin the count of a series of values at one or zero. The default for Listbox and PanelList controls is a zero. 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.

Restrict Online Access controls how the server accepts or rejects device connection requests in the Mobile Development Studio > Device Management > Device Authorization screen, and in the Mobile Unity Platform Console > Device Authorizations screen. If *Restrict Online Access* is checked, all online clients' connection requests will be rejected by the server unless the server finds they have been approved at a prior time. This also displays the "*Online Access*" column in the Device Authorization screen. If its unchecked, all devices requesting a connection will be automatically approved unless the client has a Mobile Profile (batch profile) installed. Mobile Clients with a Mobile (Batch) profile will require manual approval (a checkmark under *Offline Authorization*), on the **Device Authorization** screen regardless of whether *Restrict Online Access* is checked or unchecked.

Environment Tab – Google Integration



If you create an application that uses the Google Map, Route Planning, or Google Geo-Location and Tagging Support functionality, an activation license key from Google is required. To learn more about obtaining an activation license key, go to the following URL and click on "Paid".

<https://developers.google.com/maps/pricing-and-plans/>

and

<https://developers.google.com/maps/documentation/javascript/adding-a-google-map>

You will need to acquire an API key from Google and register your application in the Google API Console.

Note that the Map layout control in the Toolbox can be used to integrate the Google maps/Google GPS/Tagging Support into your application.

Environment Tab – Time Settings

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.

Environment Tab – Scanner Options

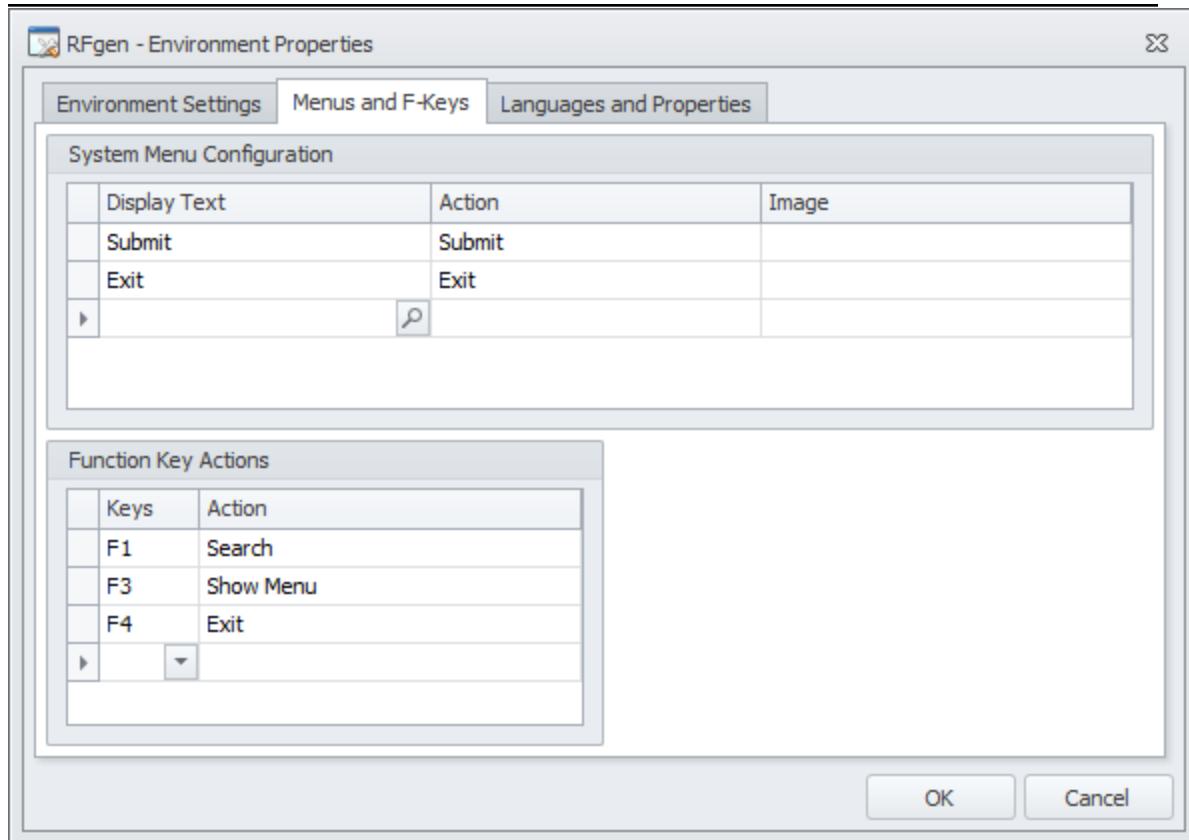
Scan for Pre-Amble and **Post-Amble** filter entries are character strings that are automatically sent from a scanner. They 'surround' the scanned data. They are optional and neither is required.

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.

Menu Strip (Systems Menu)



The **Environmental Properties > Menu Strip** screen enables you to add and remove the functions that appear on the menu strip that displays on the top, bottom or side of a mobile device -- when the menu strip button is pressed. Activation and deactivation of the menu strip is controlled by the *MenuState* property under **Solution Explorer > Mobile Themes > [Android, Apple, or Standard] > Environment: Systems Menu**. The menu strip only supports applications in graphical mode.

To view the list of supported Actions (functions), click on the down arrow in the Actions column.

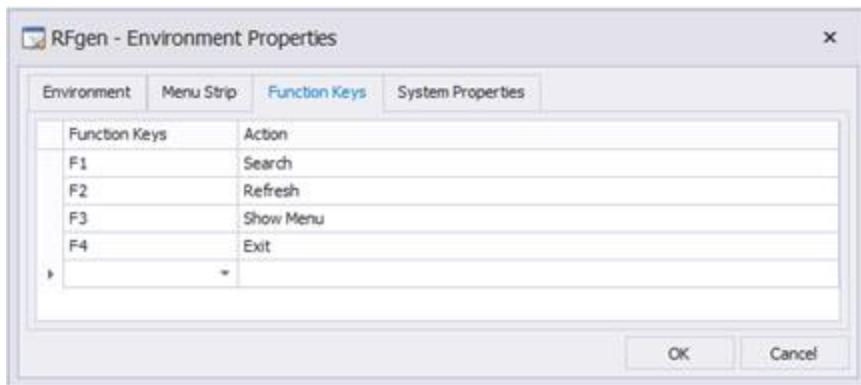
Clear	Clears the data entered for the current prompt
Exit	Exits the current process (same as F4 normally does)
Options	Displays a list of options that are valid for the device
Refresh	Refreshes the display screen
Search	Executes the OnSearch event if one exists for the prompt

Show Menu Displays a secondary menu

Submit Enters the data appearing for the current prompt

For information about viewing and testing the menu strip functions, refer to *Application Testing – RFLogin* in the RFgen User Guide.

Function Keys - Environmental Properties



The following Function Keys are preset defaults and can 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.

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.

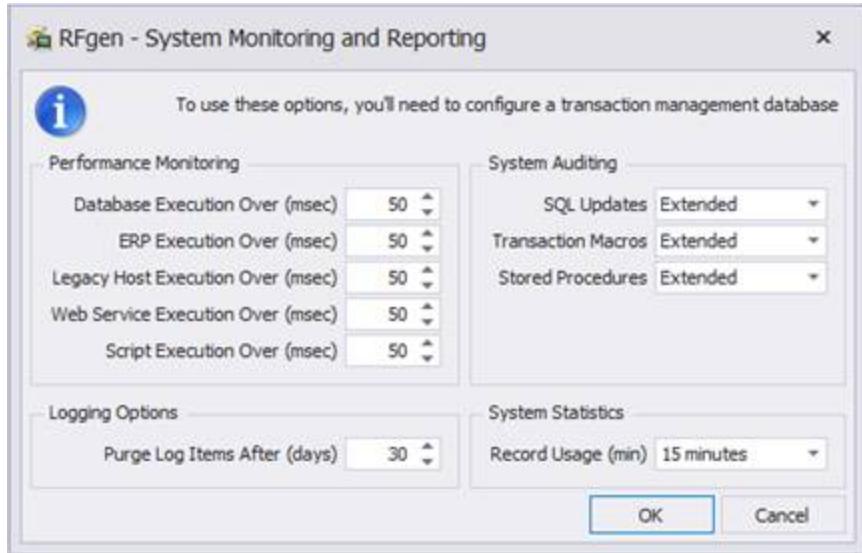
Run your mouse arrow over the function keys to see their function displayed. Screen tips will appear so that you won't have to memorize their usage.

Note: F4 entered anywhere in the application returns you to your user menu. Selecting Logout or F4 on your menu returns 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 allows 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.

Configuring Performance Monitoring



Performance Monitoring

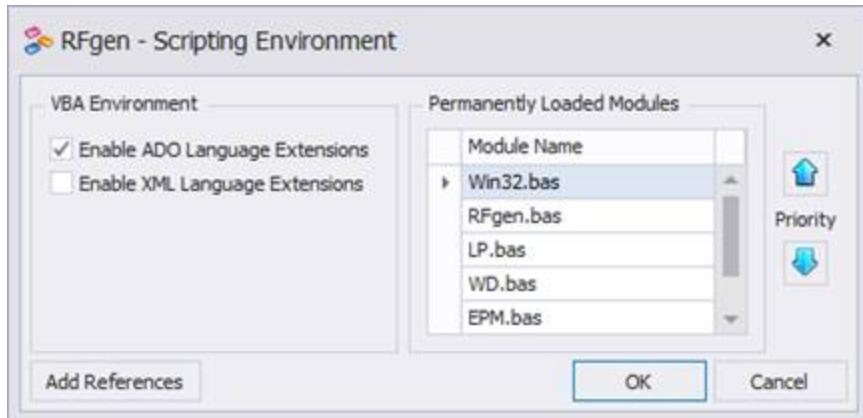
If specific connectors are taking too long to process a request, these properties can be configured to capture processing requests that take over a certain amount of time. Set the property to zero to disable. Some connections usually take longer than others. For example, the database execution time will 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 The **Purge Log Items After (days)** helps prevents the database from getting too big. Turning this feature off is not recommended.

System Auditing The **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 panel to generate graphs of data connection information.

Configuring the Scripting Environment



VBA 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.

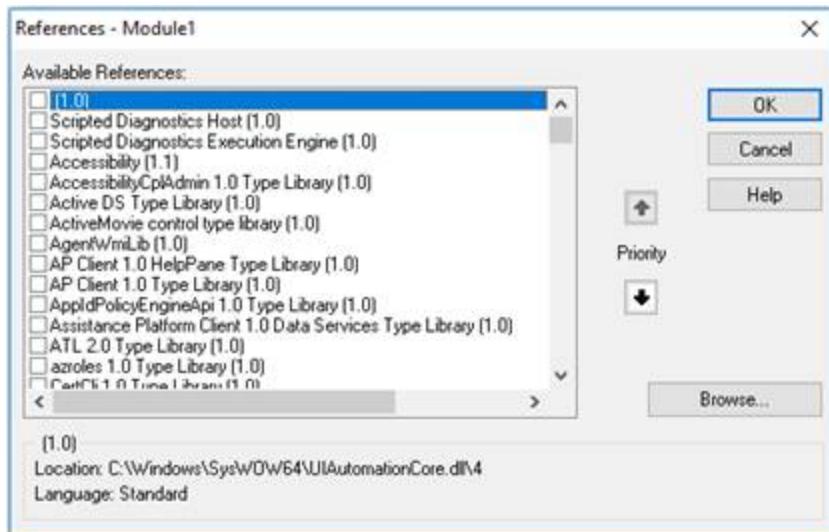
Enable XML Language Extensions provides additional parameters for manually specifying XML communication settings.

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.

Permanently Loaded Modules

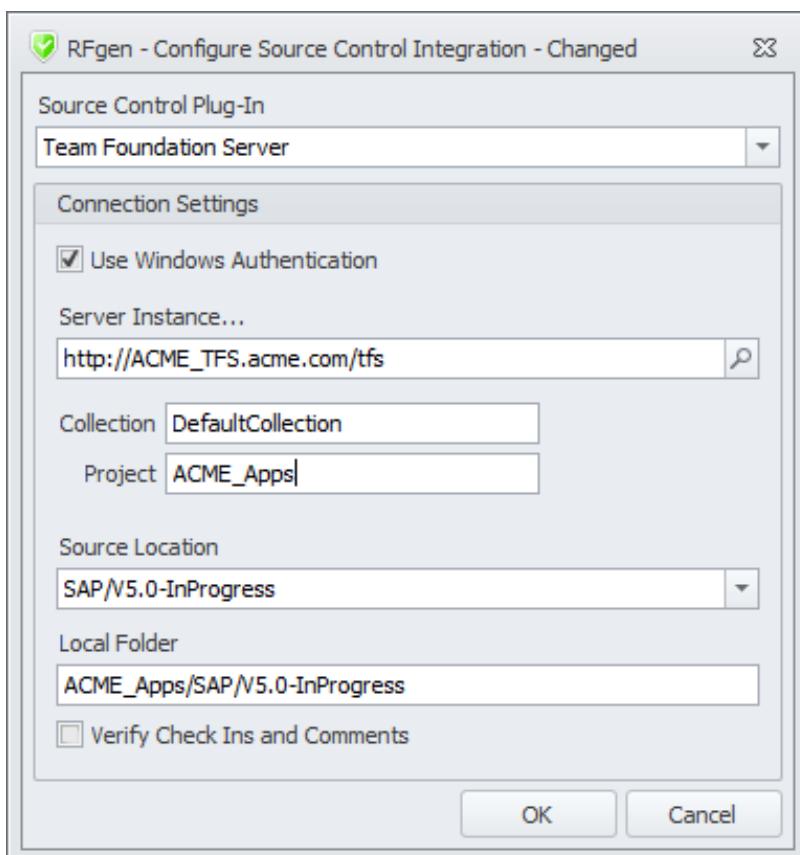
The **Win32** and **RFgen BAS** files are always loaded for each transaction. If another BAS file is created and the programmer does not want to place the code into either of these pre-loaded modules, then it may be added to this list. The Win32.bas is typically used to store global variables. The RFgen.bas 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** button will globally add Global Assembly Cache (GAC) classes to the RFgen solution. This is the window that appears.



Configuring Source Control Options

The **Source Control Integration** lists source control products that can be used to provide the development process to check-in / check-out developed objects such as users, menus, applications and images.

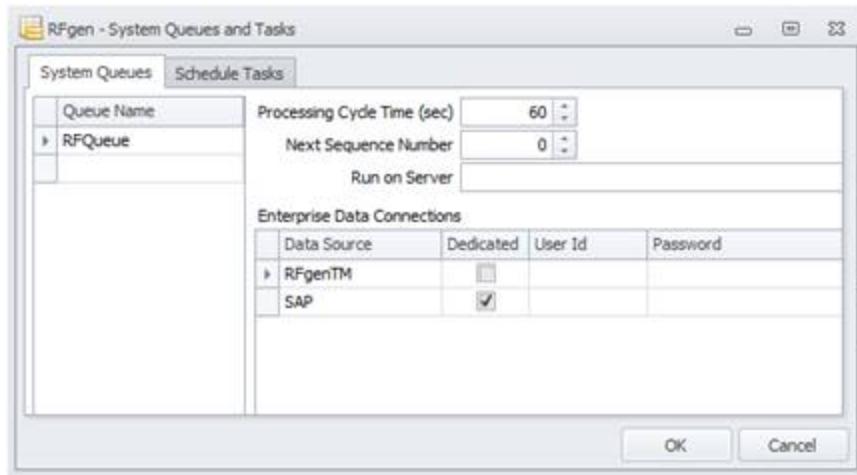


Usually the product such as the Microsoft Visual Studio needs to be installed on the local developer's PC to function.

The source control server instance and remaining properties are identical settings as those in the source control product itself. For integration with Microsoft Team Foundation Server, refer to *RFgen 5.1 Installation and Upgrade Procedures*.

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



System Queues Tab

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 Time** 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.

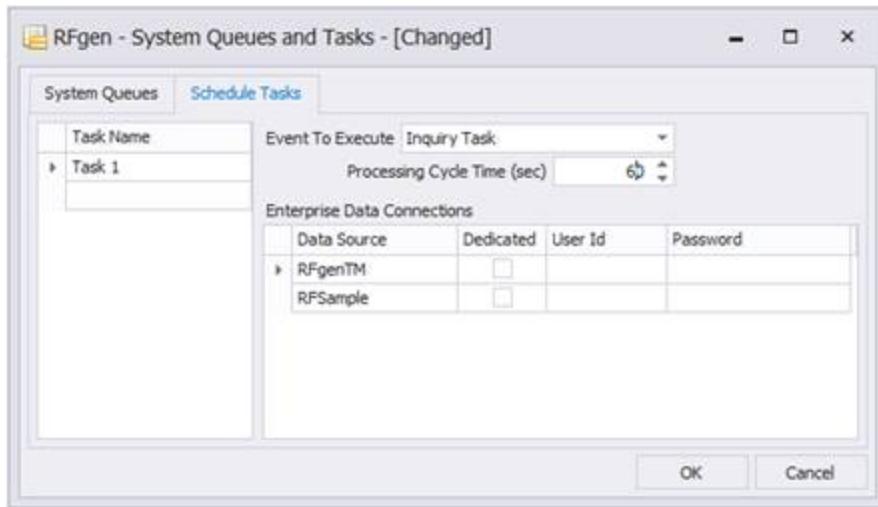
The **Next Sequence Number** option will allow the user to change the sequence number used when queuing or making entries in the logs.

The **Run on Server** is used to specify a server to prevent your queued transactions and events from running on all servers that are actively used for load balancing. If this field is left blank, RFgen will not check for any server changes and run all processes on all server(s) connected to the RFgen database. If the server IP address or "Local Host" is present, then RFgen checks which server to use and runs processes against the assigned server.

Enterprise Data Connectors

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.

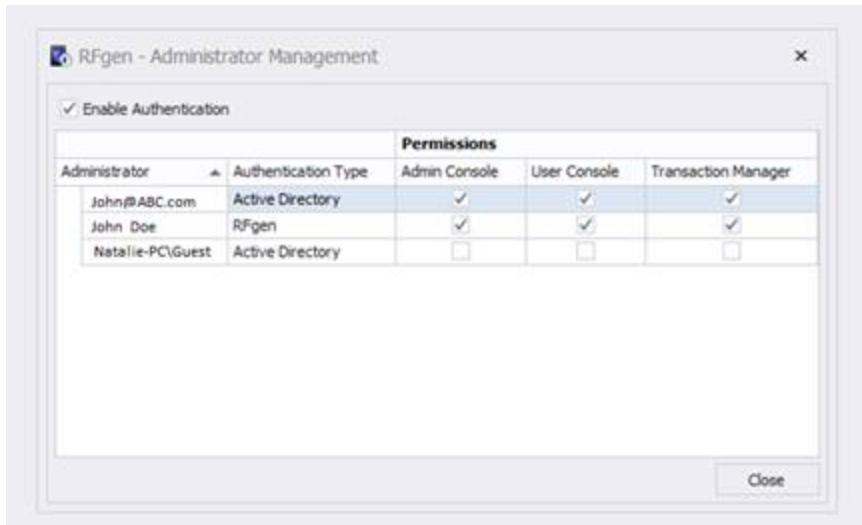
Note: Each queue process uses one client license because it is, in essence, an automated user performing tasks against the server and the backend systems. For example, a 10-client licensed system with three separate queue processes will only allow up to seven concurrent devices.



Schedule Tasks

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 seconds 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 User Access Control



The **User Access Control** feature authenticates users who are accessing the RFgen server from the *Admin Console* (the Mobile Enterprise Dashboard), the User Console (*User Management Console*), or the Transaction Manager (*Transaction Management Dashboard*).

This screen can be accessed from the *Mobile Development Studio*, **Configuration > User Access Control** menu.

When **Enable Authentication** is checked, all Mobile Enterprise Console (Admin Console), User Management Console and/or Transaction Manager Dashboard user sessions are authenticated. When its unchecked none of these console/user sessions are authenticated.

The **Administrator** column lists users who are allowed to access the server via the Admin Console, User Console, or Transaction Console. To add an administrator, the person's user account must exist in Active Directory or exist as a Local User Account. *RFgen Administrators* is a special account that include users who were added from the Local Users Account. *Active Directory Administrators* are users whose accounts are managed in Active Directory.

The **Authentication Type** column lists either *RFgen* or *Active Directory*.

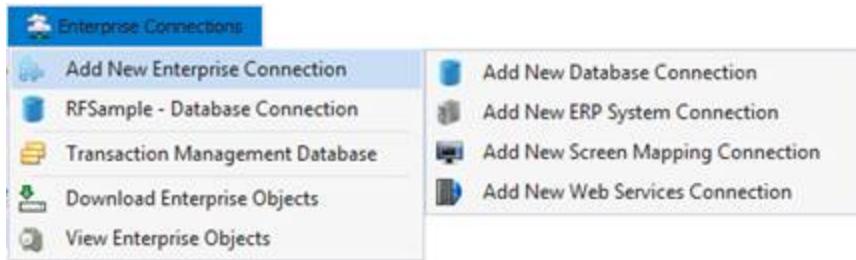
When checked, the user has access to the server from the specified **Admin Console**, **User Console**, and/or **Transaction Manager Console**.

Adding or Removing Administrators

1. From the RFgen Server: Click on Configuration > User Access Control.
2. Right-click on the Administrator column and select **Add RFgen Administrator...** or Add Active Directory Administrator.

3. The screen Enter RFgen credentials displays.
4. Enter the User's Name and Password and click OK. The user's name appears under the Administrator column as well as the Authentication type used. Check which console the user should be able to access the server and click OK to save changes.

New Enterprise Connections



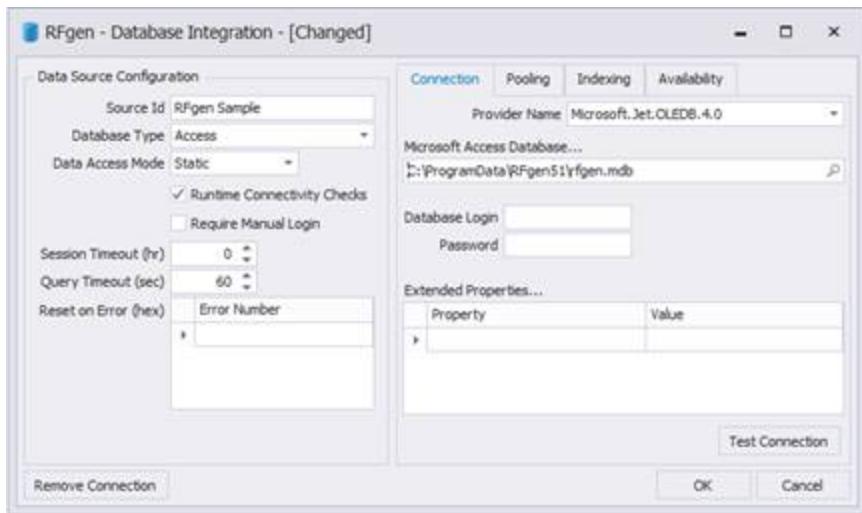
The **Add New Enterprise Connection** provides a submenu of options for connecting to database providers, enterprise resource providers, legacy host systems (via a screen map), or a web service.

The **Transaction Management Database** option is used to create a connection with a transaction management database. For details, see "Configuring Transaction Management Database".

The **Download Enterprise Objects** option enables you to download all or specific types of objects from a connected database. The **View Enterprise Objects** option enables you to select and display previously downloaded objects. For details, see **Download Enterprise Objects or View Enterprise Objects**.

Configuring Database Connections

An unlimited number of data connections can be added to the solution. From the Mobile Development Studio, select **Enterprise Connections > Add New Database Connection** (or on an existing Connection) to establish (or modify) communications settings for the database.

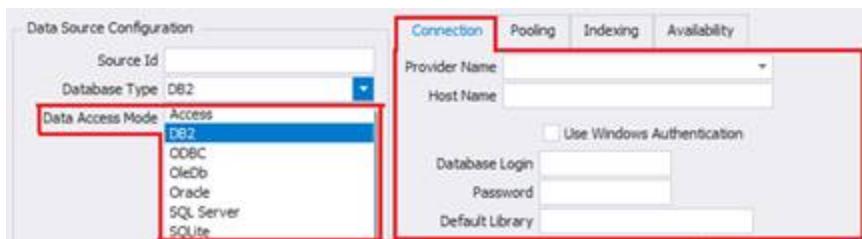


Data Source Configuration

To configure the Mobile Development Studio to connect with your own application database, use a SQL-compliant database. 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 check your 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 content of the Connection screen, to show database specific configuration fields.



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 **Runtime Connectivity Checks** and **Require Manual Login** are enabled if checked.

If **Require Manual Login** is checked, this means the client will not be allowed to connect to the database unless the user on the client is authorized. For example, with JD Edwards, if **Require Manual Login** is checked, when the user logs in to process a transaction, his/her login user ID is checked against the Named User list; If the Named User list is not setup correctly in RFgen (or JD Edwards), or, if the user isn't on the Name User list, a database error may result.

If **Require Manual Login** is unchecked, the client will be allowed to connect with the database automatically.

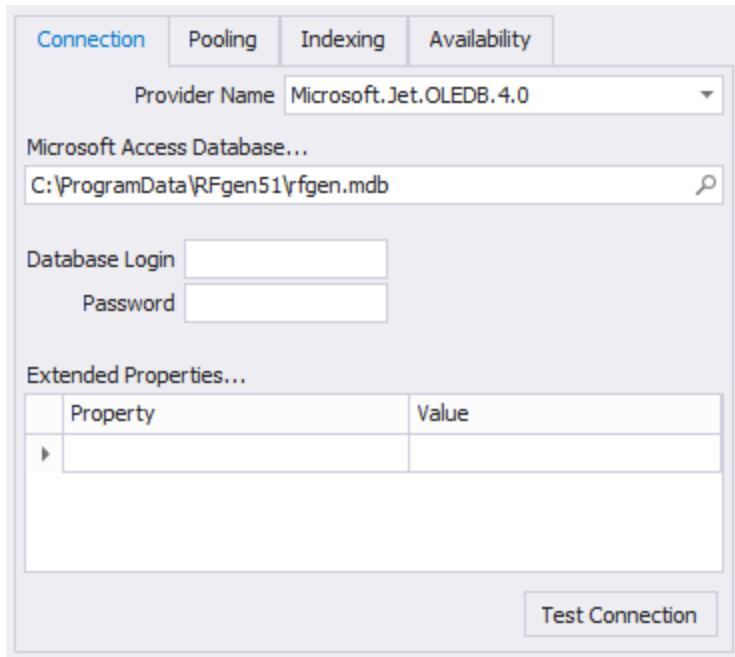
The **Session Timeout** value (in hours) will disconnect and reconnect to the database at the specified interval. This may be required if the database is configured to not allow a connection that never times out.

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

Connection Tab



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.

The screenshot shows the 'Connection' tab of a configuration dialog. The 'Provider Name' is set to 'IBMDA400'. Other fields include 'Host Name' (empty), 'Use Windows Authentication' (unchecked), 'Database Login' (empty), 'Password' (empty), 'Default Library' (empty), 'CCSID Conversion' (set to 'False'), 'Catalog Library List' (*USRLIBL), 'Initial Catalog' (*SYSBAS), 'SQL Package Name' (QGPL), 'Block Size' (4096), and 'Query Optimize Goal' (2). Below this is a section titled 'Extended Properties...' containing a table:

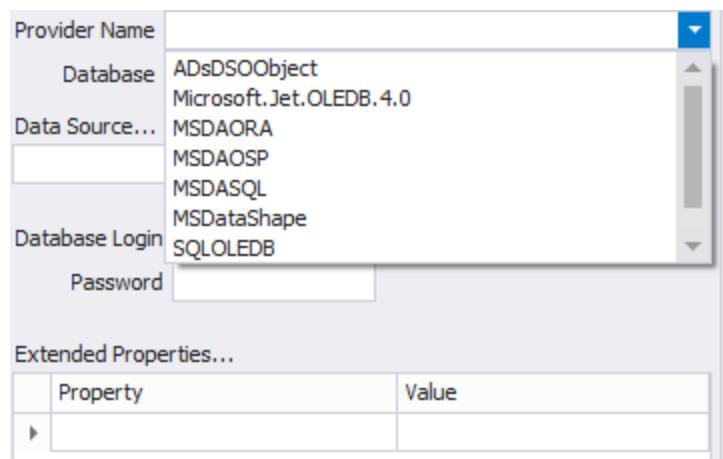
Property	Value
Block Fetch	TRUE
Convert Date Time to Char	0
SSL	DEFAULT
Sort Sequence	n

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.

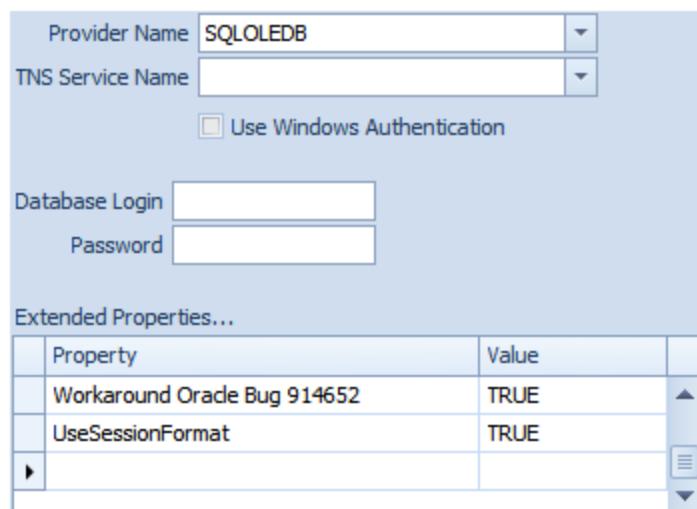
The screenshot shows a configuration dialog for a 'Data Source'. It includes fields for 'Data Source' (dropdown menu with a search icon), 'Database' (dropdown menu), 'Database Login' (empty), and 'Password' (empty).

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 Name 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.

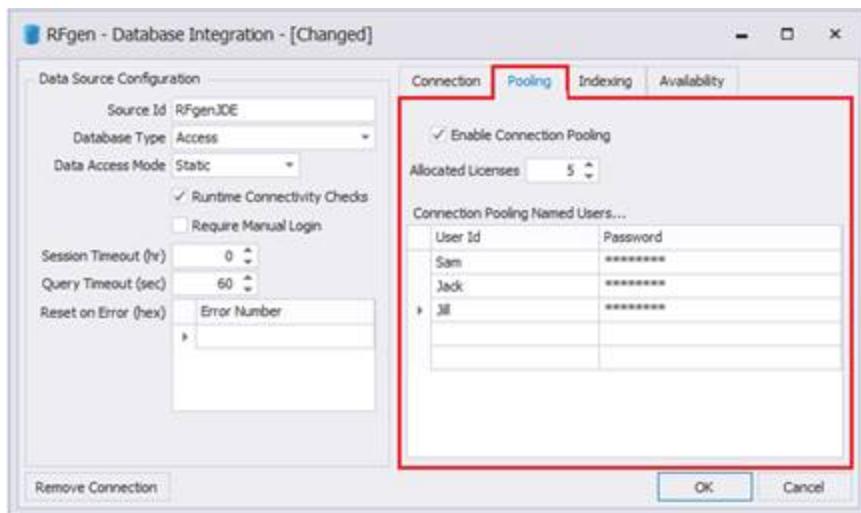


For SQLite database connections just specify the DB file itself. There are no other settings. You can specify a location and name that does not exist and clicking the Test Connection button will create the database for you.

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.

Pooling

Pooling refers to configuring multiple database connectors into one licensed connection for license conservation purposes. Pooled sessions can setup for Named Users if the pooled connection is for Oracle JD Edwards.

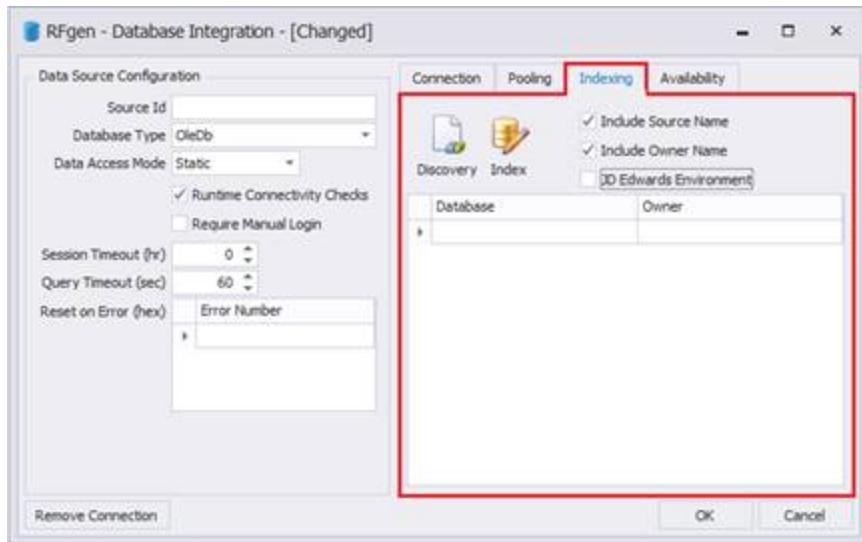


The **Enable Connection Pooling** checkbox turns on Pooling.

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.

Indexing tab



This has advanced features for connecting to the database.

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

The **Index**  icon means the server performs 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.

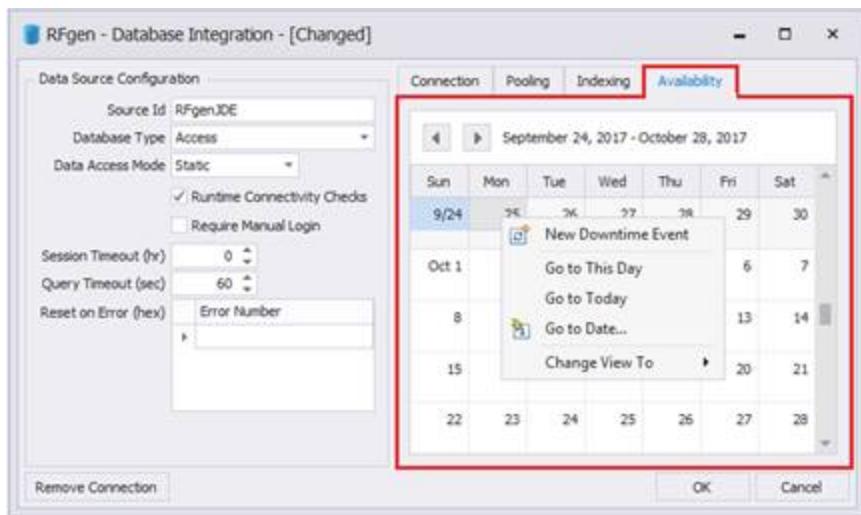
The **Source or Owner Name** information are options to more uniquely qualify the tables or database structure in case other connected databases have tables that are named the same when downloading the tables. For example, your main database may be SQL Server or Oracle, but you have a need to connect 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 **Database** and **Owner** 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 **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.

Availability Tab

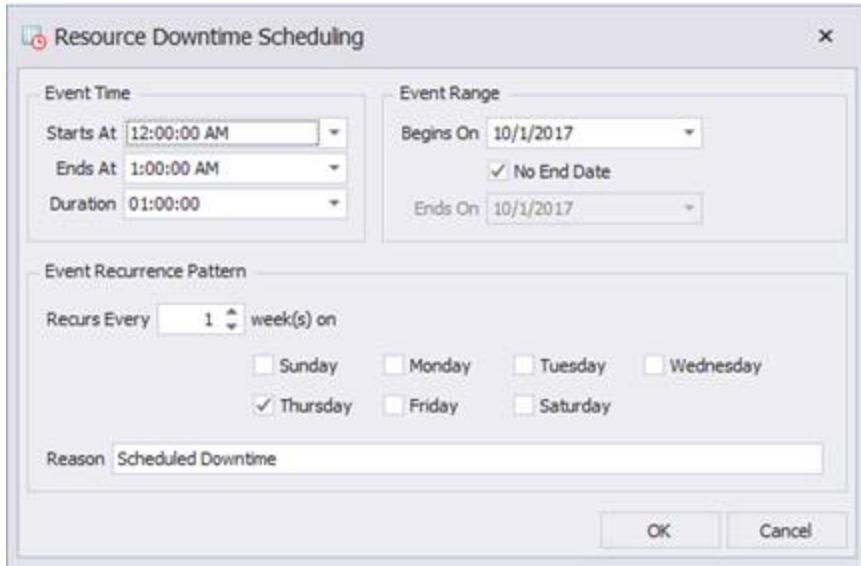
The **Availability** option is used to schedule down time for maintenance purposes.



To schedule downtime, right-click on the date or days in the calendar and select the appropriate item from the menu.

In this example, the **New Downtime Event** was selected.

In the Event Time box, the connection will be unavailable for 30 minutes, every Thursday, between 12 AM and 1 AM beginning Oct 1, 2017 and will reoccur until an End Date is supplied.



Configuring an ERP Connection

To configure an ERP connection, select from **Enterprise Connections > Add New Enterprise Connection > Add New ERP System Connection**.

If you are connecting to Oracle JD Edwards, see [Configuring for Oracle JD Edwards](#).

If you are connecting to SAP, see Configuring for SAP.

Configuring RFgen for Oracle JD Edwards

A RFgen/ERP solution set consists of the following items:

- The “engine” provided by RFgen Mobile Development Studio or Mobile Unity Platform product
- the RFgen Application Database
- the customer’s datasets.

We used the *Mobile Development Studio* to perform our setups, but the *Mobile Unity Platform Console* can similarly be used as well.

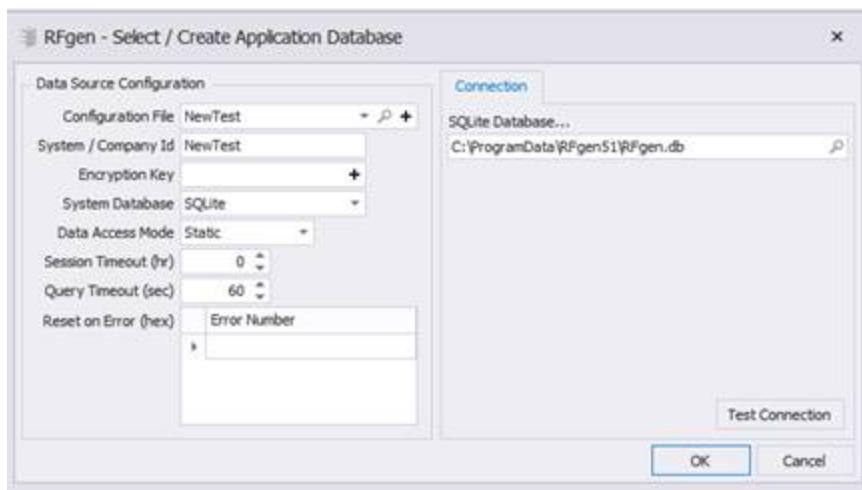
RFgen ‘from the box’ will load the engine components. Obtain the most current product versions from the RFgen web site. Once installed (and tested!), the user is ready to setup the ERP-specific source dataset. When setting up a new RFgen source dataset it does not matter what your existing database type is as it can be updated post installation.

1. Obtain the RFgen Open Source database for JDE.

The most current version of the RFgen/JDE Opensource database is always available from the support team at RFgen (support@rfgen.com) for all users and partners currently under a support contract. This will consist of a single database. This database needs to be loaded onto the users RFgen system and place into a folder where the RFgen admin will have read/write privileges. For new customers, all of these steps are normally completed by your RFgen consultant.

2. Set the RFgen Application Database.

Select **Configuration > Application Database** from the Mobile Development Studio menu. Your display may look different from the image below.



Configure the Data Source

Configuration File: Provide a new configuration file name by selecting the ‘+’ sign. We recommend a name that has some type of meaning. In this case we will use ‘JDE Test’

System/Company Id: Place an appropriate name into this field. A RFgen Partner might place their customer's ID or name into this field. In this example we will use 'RFgen Cass'.

Encryption Key: Do not setup an encryption key at this time as your starting environment is not encrypted.

System Database: Select the database type. Currently we are releasing our starting databases as .db data-sets, so select SQLite.

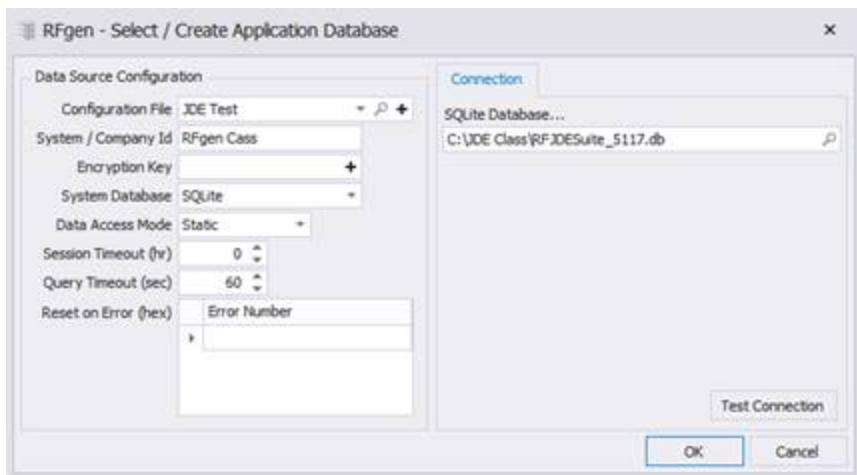
Data Access Mode: Can be Static or Dynamic. Use the default "Static."

SQLite Database: Select the RFgen Application Database from Step 1.

Leave the other settings defaulted.

Session Timeout (hr): This will disconnect and reconnect to the ERP at the specified interval. This may be required if the ERP is configured to disallow a connection that never times out. The default is 0 hours.

Click **Test Connection**. If the "Connection successful" message displays, click **OK** to exit the message, then click **OK** to save changes.



At this point RFgen will attempt to make the following connections:

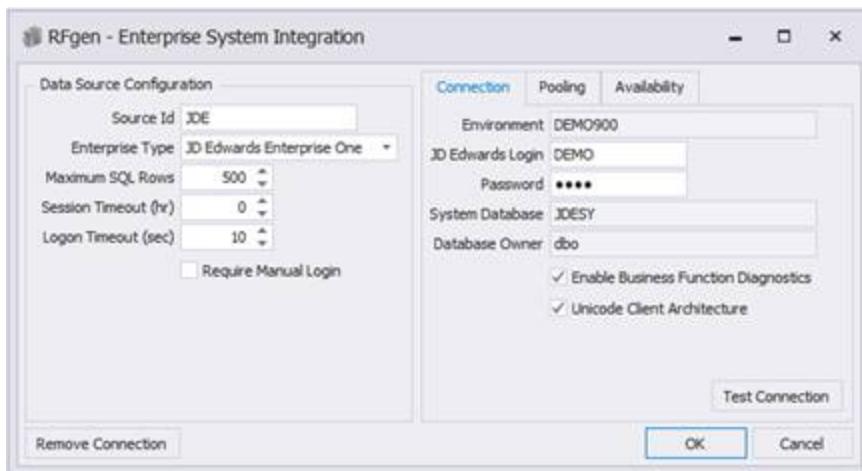
- RFgen (This is the base set.)
- RFgenTM (This is the optional transaction manager.)
- JDE (This is the JDE connector.)
- JDE_DEMO (This is our example, JDE database.)

Since our setup is still incomplete, the connection status icon displays a warning.



3. Set the JDE Connector.

Select **Enterprise Connections > JDE – JD Edwards Connection** from the menu.



Data Source Configuration

Source Id: For JD Edwards, please leave this value as 'JDE'.

Enterprise Type: For JDE Enterprise environments, leave the value as 'JD Edwards Enterprise One'.

JD Edwards Login and Password: Enter a JDE Administrator during the original install. This may be changed later.

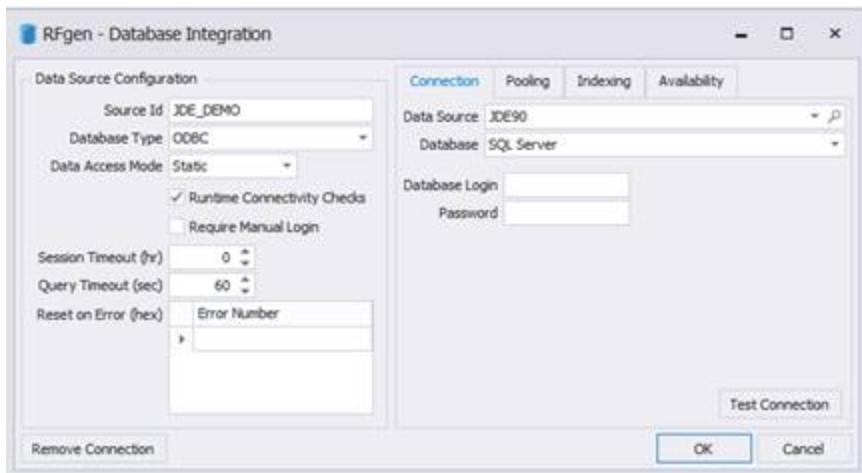
NOTE: The Environment, System Database and Database Owner are pulled from the customers JDE.INI file and cannot be changed.

The other default values should display and match the screen above for original setup. Be sure the Require Manual Login is not checked on.

Test and Save. Selecting the 'Test Connection' button will connect to the user's JDE system using the JDE Development Client (aka JDE FAT client) that was installed on the system. If you receive the 'Connection successful' message, click OK to exit the message, then select the OK button to save the change.

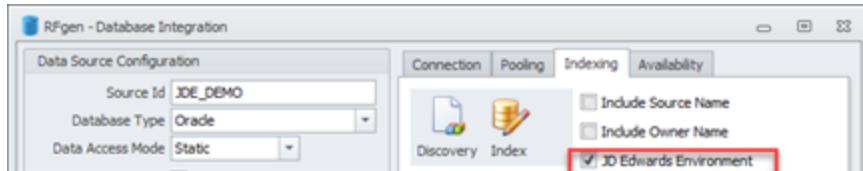
4. Set the JDE Data Connector

Select **Enterprise Connections > JDE_DEMO** from the menu.



The Source Id is the name you wish to use. The Database Type and Source need to be correctly entered based on the customer's JDE Development client database connection information. The specific questions will differ based on the database type selected. Most common for JDE is either an ODBC connection (as noted above) or an Oracle connection. Contact your JDE database administrator and/or your Oracle administrator for this information.

To validate your settings, click **Test Connection**. After the "Connection successful" message displays, click on the **Indexing tab**. As JDE contains multiple databases, select the JD Edwards Environment checkbox first.



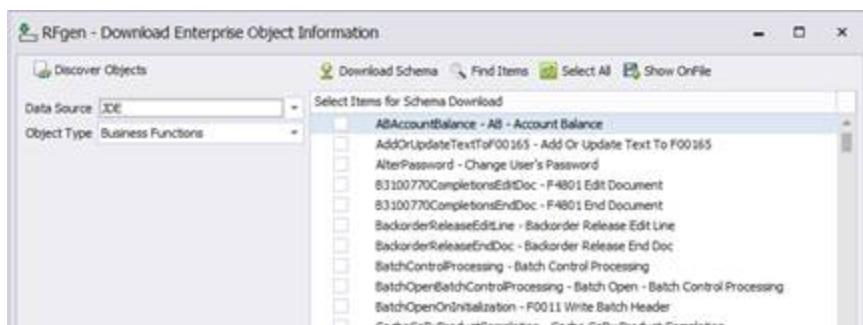
... then the **Discovery** icon. This will list the JDE datasets. After the list is populated, select the Index icon, which will index those datasets to allow RFgen to quickly read the JDE data. Upon completion, click the OK button save the change. The footer should show both connections valid.



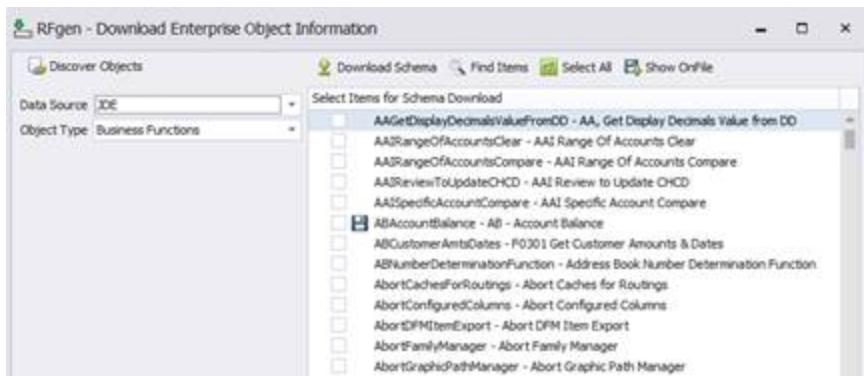
5. Load / Update JDE BSNFs and Processing Options

Load the JDE BSNFs (Business Functions)

- a. Select **Enterprise Connections > Download Enterprise Objects** from the menu.
- b. Select **Data Source:** JDE, then select, **Object Type:** Business Functions.
- c. Tap the **Discover Objects** icon.

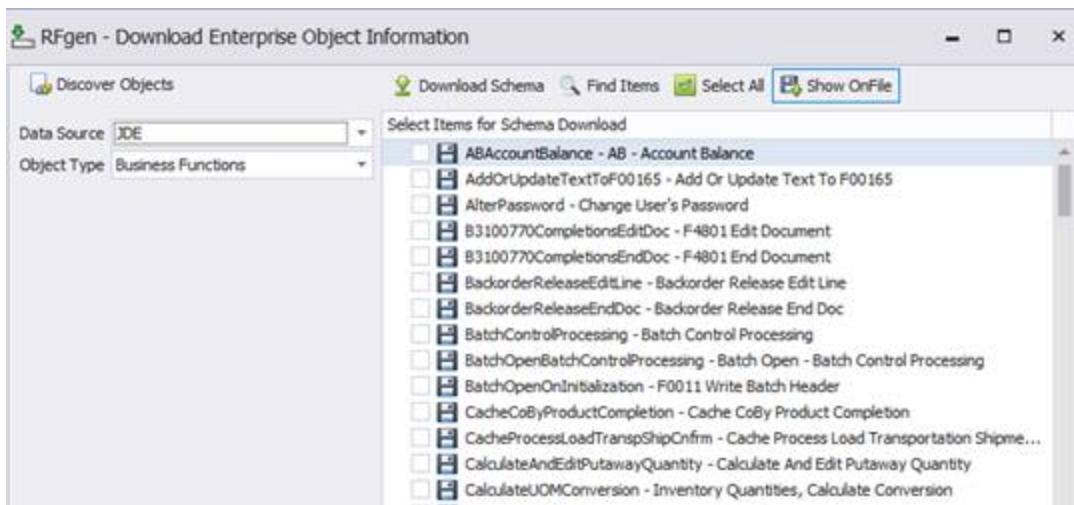


This will open JDE and pull all of the JDE BSNFs from the customer's JDE-connected JDE environment. Upon completion, the system will show all BSNFs in alphabetical order. In the example below, note the ABACcountBalance item contains a small disk icon.



This disk icon shows that this BSNF is one currently used by your RFgen environment.

For the original setup, select the **Show OnFile** icon.

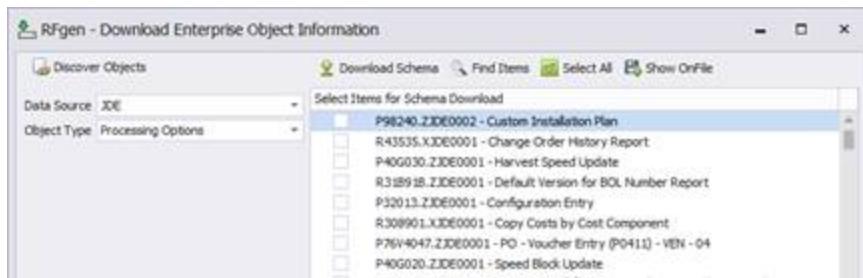


With ONLY THOSE **ONFILE** SHOWING, click the **Select All** icon, which checkmarks all the listed items. Next, select the **Download Schema** icon. This will update RFgen to match the current JDE BSFNs.

Note: *This step should be completed with each JD Edwards new release, or anytime an effected BSFN is changed by JDE.*

Load JDE Processing Options

- Select the menu item Enterprise Connections | Download Enterprise Objects.
- Select **Data Source: JDE** and **Object Type: Processing Options**.
- Click the **Discovery Object** icon.



This will open JDE and pull all of the Processing Options from the customers JDE connected JDE environment. Upon completion, the system will show these in alphabetical order.

With **ONLY THOSE ONFILE SHOWING**, click the **Select All** icon, which will checkmark all the listed items. Next, select the **Download Schema** icon. This will update RFgen to match the current JDE Processing Options. ***This step should be completed with each JD Edwards new release, or anytime an effected processing option is changed by JDE.***

6. Configure RFgen Administrative Setups for JDE

This section reviews the Administrative application that comes from RFgen's Open Source database of applications for JDE. This process is only needed for initial setup of JDE customer for RFgen or when modifications are made. The best way to access this is application and its many options is through the RFgen Desktop Client.

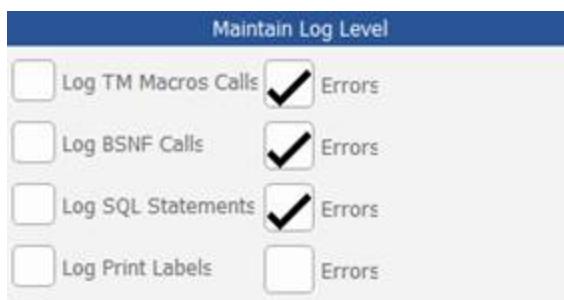
Note: RFgen comes preset with the Setup Admin user ID of 'adm' and no password. We highly recommend that this password be changed if using this on a production system.

a. Sign in. Sign into RFgen using the RFgen Desktop Client and the id of 'adm'. Note that RFgen contains many optional features such as Offline Inventory, Licensing Plating, Warehouse Director and Vocollect voice processing. Also included are some shortcuts to setup items like the JDE Fiscal Date Patterns or access to special processing option setups.

b. Configure RFgen Administrative Details.

This section must be complete for all setups. Once completed, it does not need to be re-visited unless one of the basic items (as noted below) needs to be modified.

Maintain Log Level: RFgen provides an abundance of information – which generally you will not need (and will not want unless you have truly unlimited disk space). For standard usage, set to capture only Errors. No capturing should be set for Print labels unless you are having issues.



Log Scanning Versus Typing: This determines what happens when a scan takes place while inside RFgen. If this is changed from the below default, you will need to perform additional coding across your existing RFgen codebase.

Log Scanning Versus Typing	
<input type="checkbox"/>	Force Scanning
<input checked="" type="checkbox"/>	Return Values from Search
<input type="checkbox"/>	Scan Lot Number for Item

Environment: This controls many RFgen setup items for JDE functions.

Environment	
OneWorld / Enterprise E1	9.10
Proc.Opt.Release	9.10
<input checked="" type="checkbox"/> JDE OneWorld / Enterprise E1	
<input type="checkbox"/> Named User	
Days Until Passw. Exp.	90
Passw. Min. Length	6
<input type="checkbox"/> Advise User Locked	
<input type="checkbox"/> Advise User on Password Expiration	
<input type="checkbox"/> Allow Password Change	
RFgen Release	5
JDE Database Type	SQL ▾
LP Database Type	SQL ▾
<input type="checkbox"/> Use LP Commitment Control	

OneWorld / Enterprise E1. This displays the JDE version being used. Versions from World A.7.3 up to EnterpriseOne 9.2 can be selected.

Proc.Opt.Release. This displays the Processing Options version being used. In most cases when the OneWorld / Enterprise E1 selection is made, this option will automatically select the correct Processing Option version.

JDE OneWorld / Enterprise E1 checkbox. Click this ON if using any version of OneWorld.

Named User. Check this ON if RFgen is using JDE Names users.

Days Until Passw. Exp through Allow Password Change. If not using the Oracle Security Server and/or JDE Named Users, this allows the admin to setup various security options. If using Oracle Security Server and/or JDE Names users, these settings have no value.

RFgen Release: This is the release version of RFgen that is in use.

JDE Database Type and LP Database type: Because SQL syntax can be a bit different, this is a flag setup so RFgen knows which syntax to validate against. This is generally only needed for some of the RFgen optional elements.

LP Commitment Control. This flag is used only if the customer is utilizing the RFgen Licensing Plating option.

Transaction Queuing. This section is used only for RFgen versions earlier than 4.1. After 4.1, the newer transaction queuing was added to RFgen, and this section was no longer applicable.

Transaction Queuing

Notify User for TM Errors

Use TM Queueing

Load Balance the Queues

Queue names for MonitorTM (sep. by ";")
RFQUEUE;RFQUEUE1;RFQUEUE2;RFQUEUE3

Alert after x Min.: # Connector DBMoveQueue: 3

The SAVE button is clicked when saving any changes to this screen. The last user, date, time will be tracked.

Save

Last Update: ADM 09/10/2019 14:45:19 158.222.2.202

Custom User Profiles. This provides a method to build unique user profiles. This is generally no longer needed with the current RFgen versions and with today's hand-held devices.

Custom User Profiles

Use Custom User Profiles (RFUP001)

Email Setup: While most customers sending email from RFgen have multiple requirements based on user IDs, transaction name, etc., for those needing only a single email address set, the defaulted data can be setup here.

Email Setup

Mail To

CC

Email Type 3

Server Name (SMTP Only)

Offline Inventory. This is where the user would enable offline inventory and advise if using the JDE transaction server.

Offline Inventory

Offline Inv. Enabled

JDE Transaction Server

Label Printing. For label printing, RFgen needs to know the location of the label print server, as well as the location of the label template folder.



Appearance. These can be modified as needed and will control the listed item. For example, if your locations required a 20-character input, this would need to be modified. Of importance, these are defaults only, and are often over-written inside the actual codebase to meet the needs of that specific transaction and/or based on the JDE data definitions.

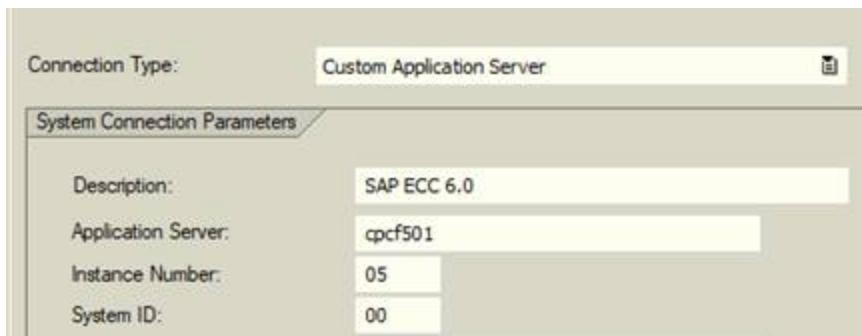
Appearance	
Min. Search Length	0
Max. Search Rows	200
Length Item Number	10
Length Location	10
Length Lot / Serial Number	12
Length Quantity	7
Length License Plate (LP)	8

You are now done with setting up standard, administrative items.

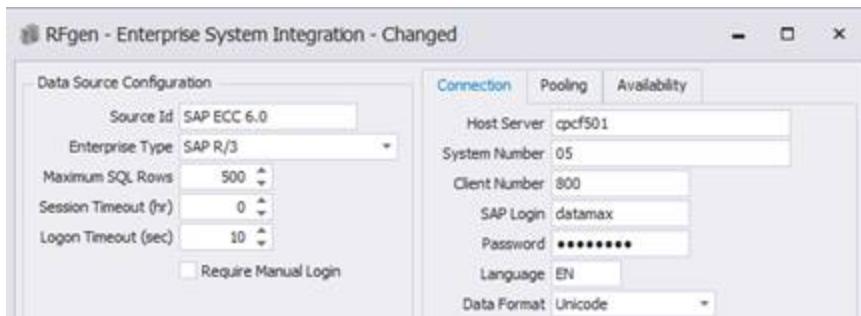
Configuring RFgen to SAP

RFgen is a zero-footprint integration to SAP, meaning there is no custom coding or additional software to install on your SAP server.

RFgen makes the connection to SAP using the SAP Connection Properties found in your SAP Logon screen.



Example of an SAP Connection Parameters Screen



Example of Corresponding RFgen Data Source Screen

From that connection, RFgen can download the schema for any remotely enabled function module/BAPI.

Setup Requirements

RFgen requires a 'User ID' to make the connection to the SAP server. A System ID user is recommended for this purpose. For example, you can enter "RFgenConnect" or even "RFgen".

Requires a non-expiring password

Must be assigned authorization objects S_RFC and S_TABU_DIS

To configure an SAP connection

Select **Enterprise Connections > Add New Enterprise Connection > Add New ERP System Connection**.

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

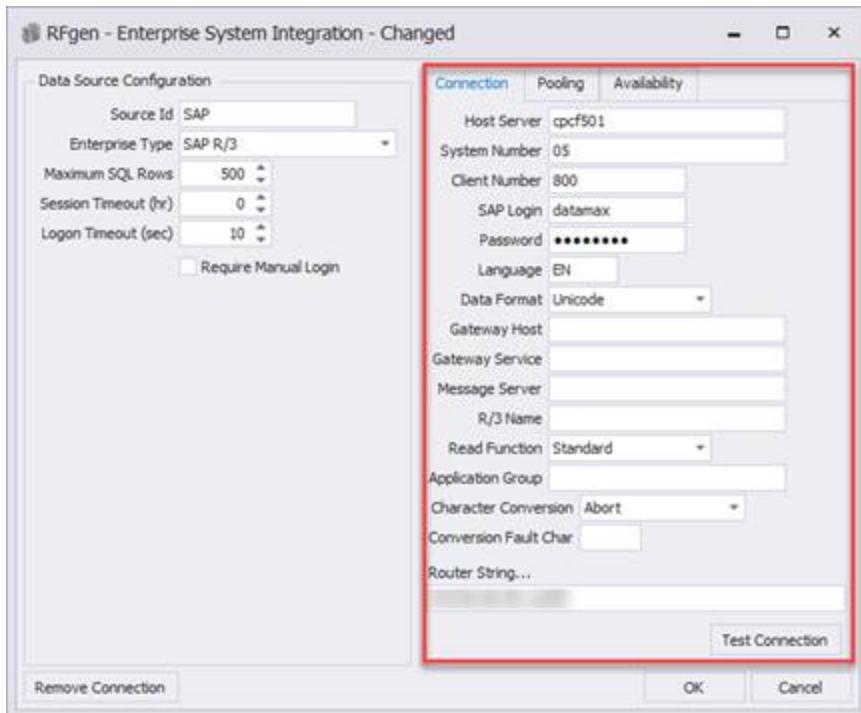
Next, enter a value for the **Max SQL Rows**. The **Max SQL Rows** prevents a 'lock-up' scenario in case a query was bringing back too many records by accident. ERP systems can typically have millions of records and this could prevent a frozen client. The command ERP.ReadData can perform SQL statements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

You can select a **Manual Login** if desired.

Connection Tab (SAP)



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

The **Data Format** 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.

Read Functions has two options: *Standard* and *BBP*. The Standard option executes SQL queries using the RFC_READ_TABLE function module; the BBP option executes SQL queries using the BBP_RFC_READ_TABLE function module. Only used with SAP HANA.

SAP Load Balancing

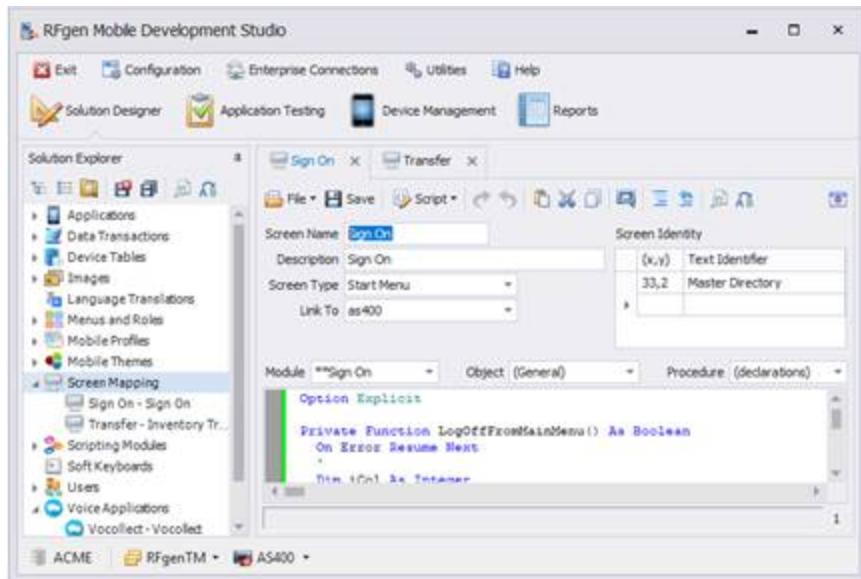
For SAP load balancing, configure the following fields: **System Number**, **Client Number**, **User ID**, **Password**, **Message Server**, **R/3 Name**, and **Application Group**.

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 **Pooling** options and **Availability** options work exactly as described in the **Add New Database Connector** section.

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.

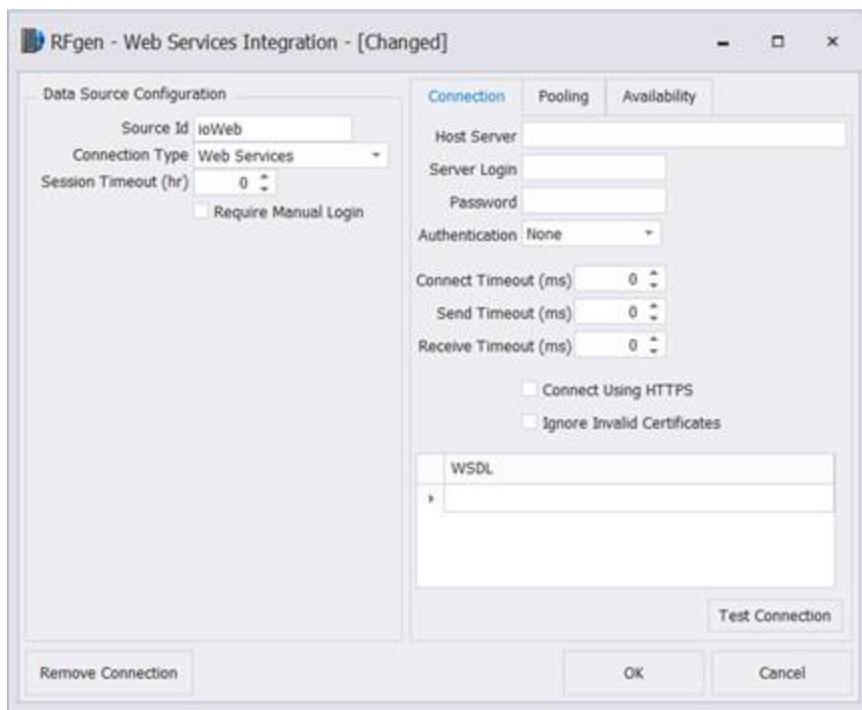
In practice, screen mapping applications use keystrokes recorded for a host screen's navigation and data entry, along with the collected data and play back of the keystrokes, while replacing the recorded data with the newly collected data. Accuracy in staging and applying keystrokes is of the utmost 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.

For more details on how to record macros, see [How to make Screen Mapping Work](#).

Adding A New Web Services Connection

To configure a Web connection, select **Enterprise Connections > Add New Enterprise Connection > Add New Web Services Connection.**



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

Select the Connection Type from the Web Services list menu.

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.

Select the type of Authentication to use when sending the User Name and Password over the Internet: *None*, *Text*, or *Digest*. *Digest* is generally an application of the MD5 cryptographic hash.

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.

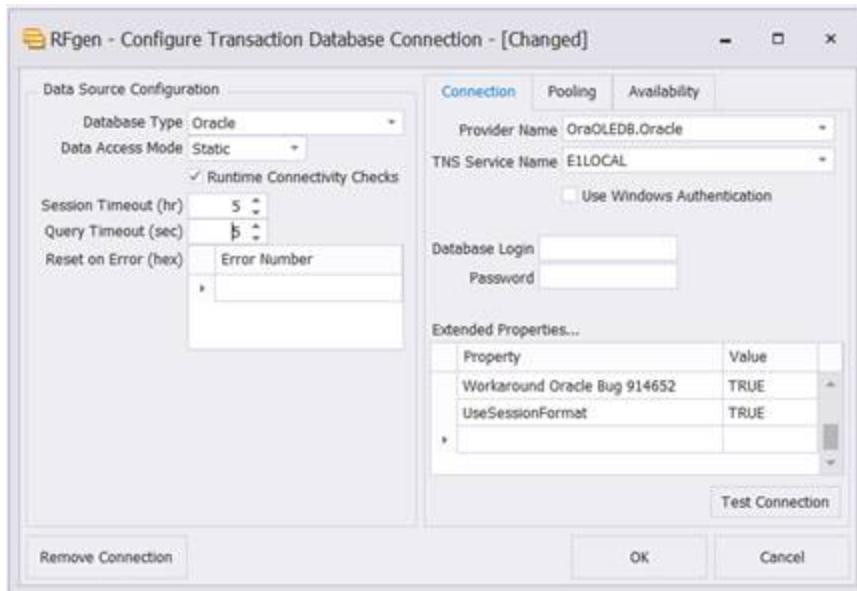
The **Ignore Invalid Certificates** option is a True or False value (only for HTTPS connections) indicating that the data connector will ignore certificate errors from the server. If this is set to False, and there is an error, it will be logged in the RFgen error log and the Web Object's Execute method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

The **WSDL** grid is for the XML-based file which describes the business logic, parameters and data structure to be used when processing/exchanging information with the web logic server.

The **Pooling** and **Availability** options work exactly as described in the database connector section.

Configuring Transaction Management Connection

To configure transaction connections, select **Enterprise Connections > Transaction Management Database**.



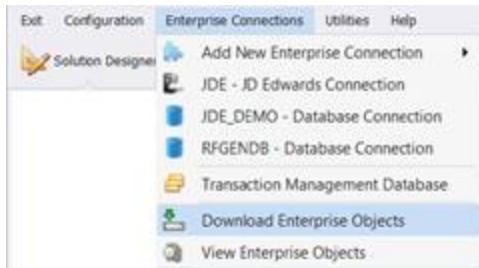
Configuring a Transaction Management database connection is the same as any other database connection. All the same fields apply. This connection is simply dedicated for the queuing process. See the section "Configuring Database Connections" for details.

There is one property that is unique: **Reset on Error (hex)** when turned on will reset the data connection under the following conditions. This should not be used unless deemed necessary.

If you submit a SQL statement to the DB, and an error code is returned, RFgen will look for the code in the *Reset on Error (hex) table*.

- If the error code is found, RFgen will try to reset the data connector. (RFgen won't resend the submitted query that produced the error code.)
- If RFgen cannot find the error code in the *Reset on Error (hex) table*, RFgen won't reset the data connector.

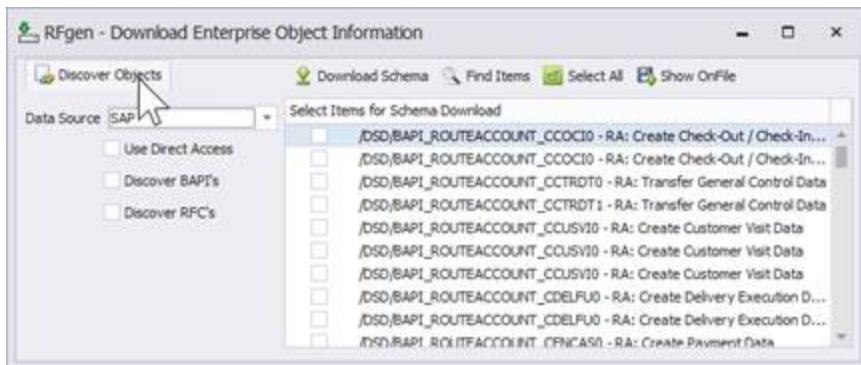
Download Enterprise Objects



If you want to access table fields directly, or to take advantage of backend functions or stored procedures, the server needs to know which Data sources to connect to and which items to download. It also needs the required tables or procedures and the object's structure in order to internally generate the proper calls and perform the reads and writes.

To access the tables, RFgen will prepopulate the display with relevant objects in the file when you select **Enterprise Connections > Download Enterprise Objects**. Once your schemas are displayed, you can select and refresh the schemas without performing a Discovery first.

If however you need to discover objects, this function is also available.



Discover Objects allows for all objects to be discovered or just selected items that may be selected.

Download Schema starts the download process

Find Items provides a text search of items using the words you enter.

Select All selects all entries in the list.

Show On-File limit the list to previously downloaded tables.

You can also check these boxes to filter items before you download: Use Direct Access, Discover BAPIs, Discover RFCs

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 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 > Download Enterprise Objects**.
2. Select the ERP connection from the Data Source drop-down and select **Discover Objects** from the menu.
3. Click the rows of functions to be downloaded (or click **Select All**), and then click **Download Schema**.

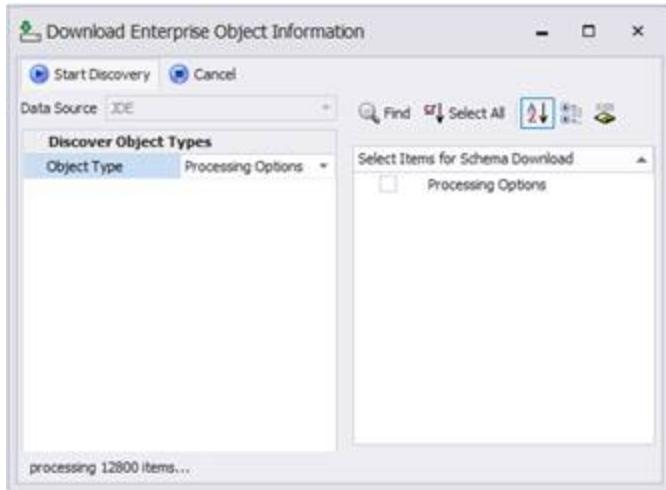
Selecting all business functions from an ERP system will save an extremely large set of data in the RFgen application database and could take a very long time. Only download the business functions that are required by the applications.

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.

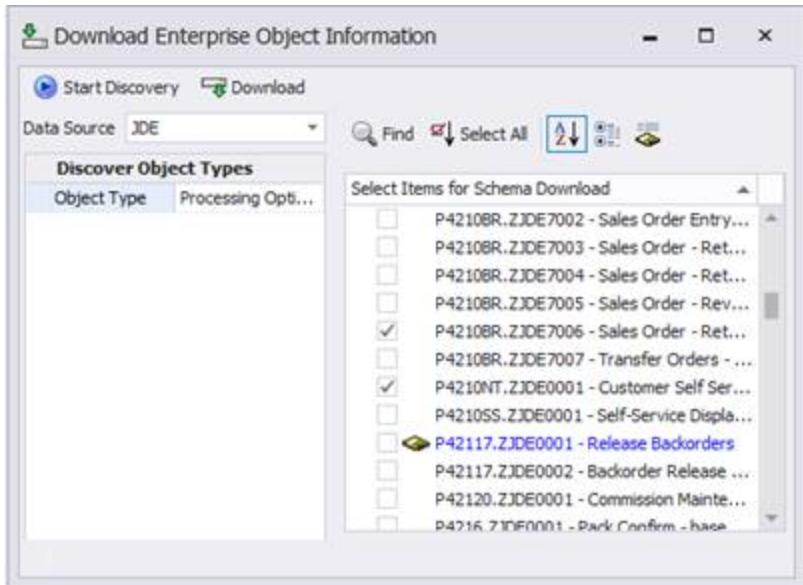
Downloading JDE Processing Options

To work with business functions from an JDE system, users must first  **Download** the desired business functions from the JDE system into RFgen.

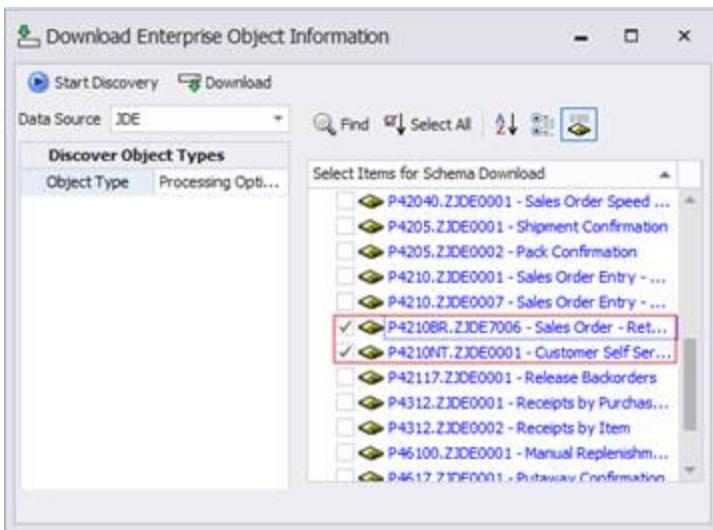
1. Click on the **Enterprise Connections > Download Enterprise Options**.
2. Select the JDE connection source from the **Data Source** menu.
3. Select *Processing Options* from **Discover Object Types >Discovery Mode** drop-down menu.
4. Click **Discover Objects** from the menu.



5. The Discovered objects display in the right pane. Select the desired processing option to be downloaded or click **Select All**. You can also use Find to located the specific item(s) to be downloaded.



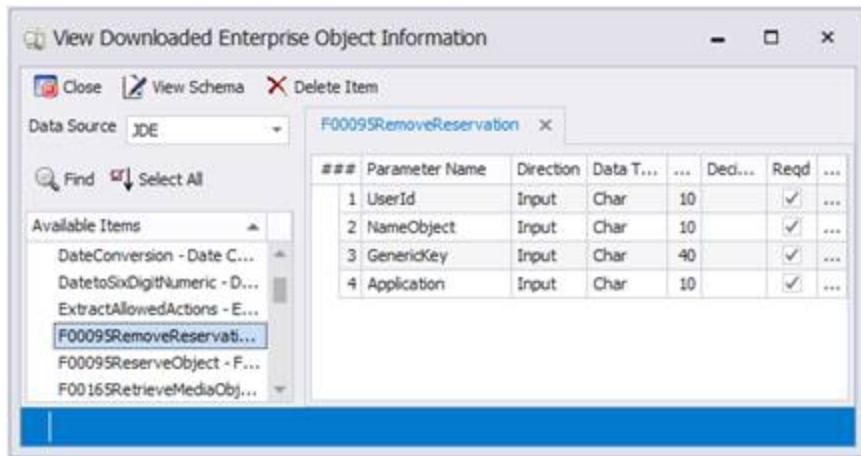
6. Items in blue indicate they have been downloaded before.
7. Click **Download**.



In this example above, the checked items are the new downloads added to the list of prior downloads.

Viewing 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 a chosen 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).

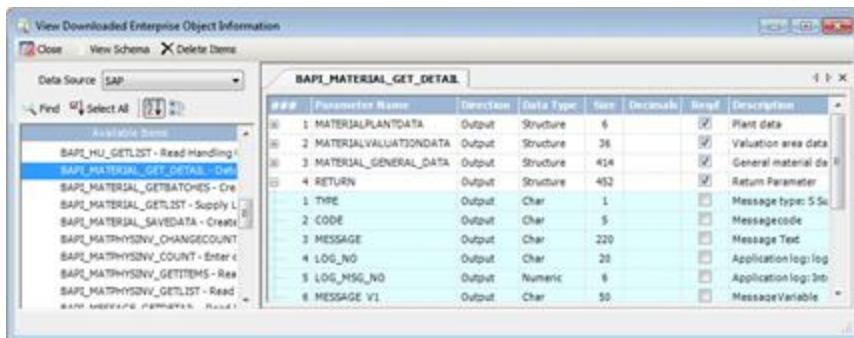
Only fields defined as updateable will be sent to the database when a transaction is completed. This is only true if RFgen is generating the SQL statements internally as opposed to user created SQL statements. Fields not defined as 'null allowed', will send a space or a 0 (zero) if no data is collected 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.

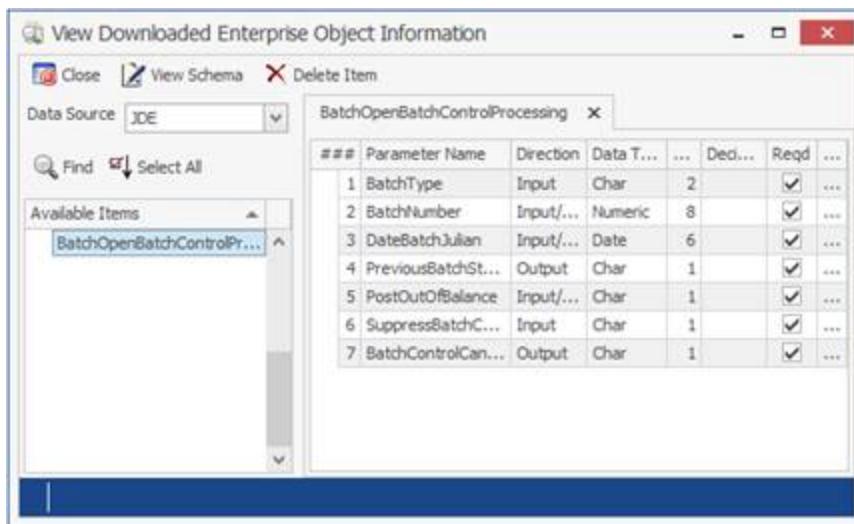


A list of downloaded business functions is displayed. If a description is stored in the database and was retrieved by RFgen, it can be displayed after viewing the parameters. Select a name and click the "View Schema" menu item or simply double-click the business function name to view its parameters.

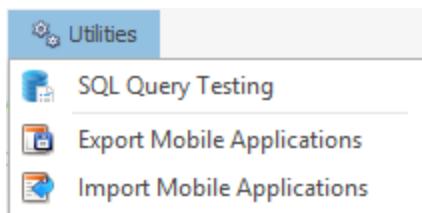
Viewing JDE Processing Options

To view JDE Process Options parameter information, follow these steps:

1. Click on Enterprise Connections - View Enterprise Objects.
2. Select the JDE connection source from Data Source.
3. The Processing Options displays in the left pane of the View Downloaded Enterprise Objects Information screen.
4. Select the Process Option then click View Schema from the menu. The Process Option's parameters and its associated details will display in the right pane. Double-clicking the Processing Option will also display the details in the right pane.



Utilities



The Utilities menu option provides tools to test SQL statements, import or export objects, and view and undo specific actions. 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.

Under SQL Query Testing is the option to enable or disable your ERP Connections when executing a SQL query.

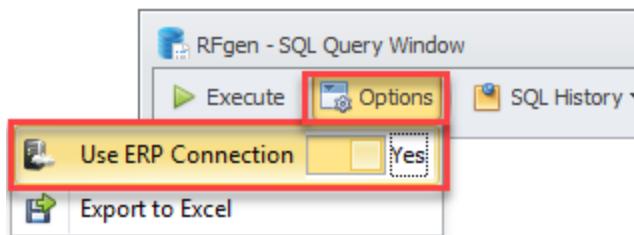
SQL Query Testing

You can test SQL statements to see the results before executing them in the code, and preserve and recall the statements from SQL History. 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.

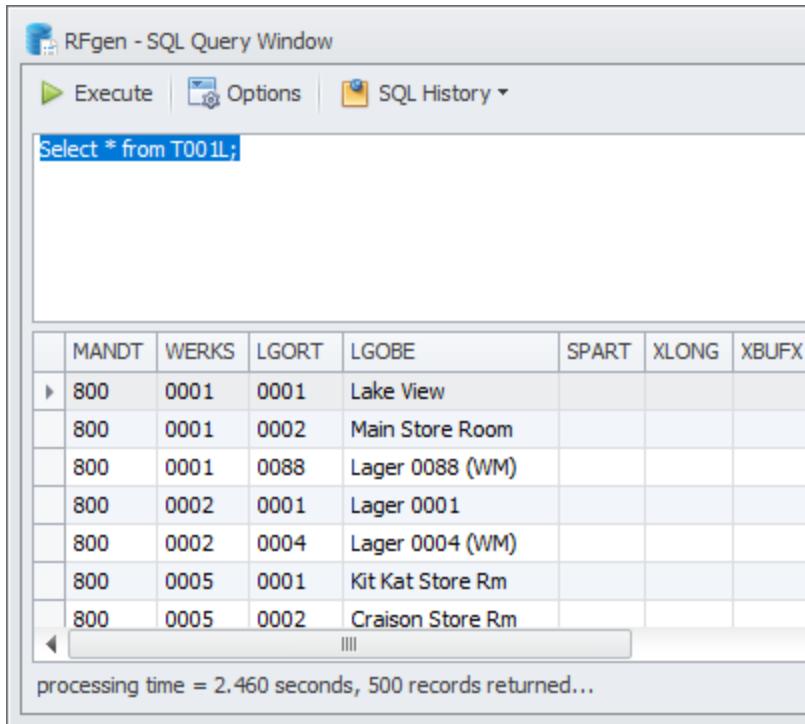
To access this function click on **Utilities > SQL Query Testing > Options**.

The **Options > ERP Mode** menu allows you to enable a connection to a specific ERP which simplifies the process of executing a SQL statement against an ERP connection such as SAP, Oracle JDE, Oracle EBS, Deltek CostPoint, or Microsoft Dynamics. You slide the box over to enable or disable the connection.

To test the query, enable the ERP mode (slide box to the right), enter your statement in the query box and press Execute. The execution will display results in the second box.



The **Export to Excel** 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.



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;

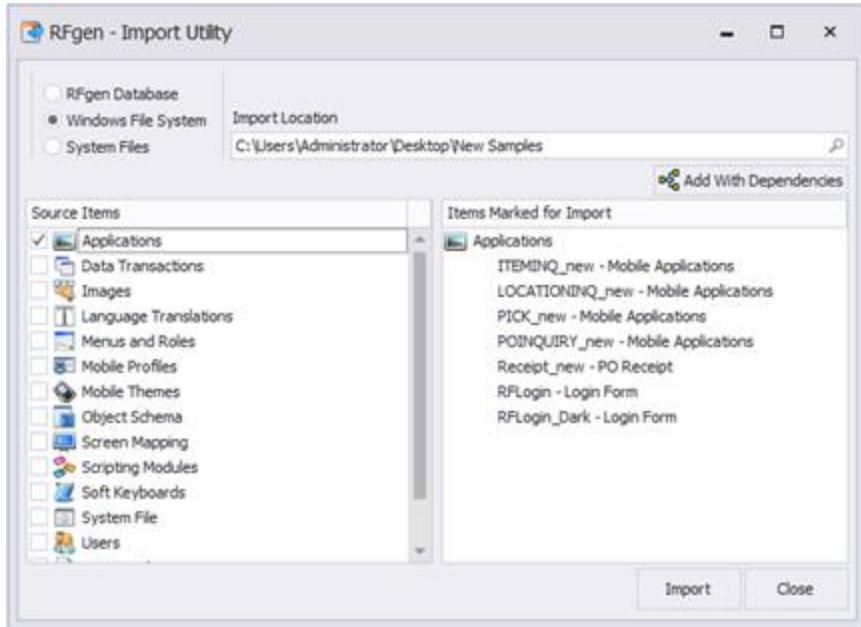
Select * from itemmaster;

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 **SQL History** option allows you 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.

Export or Import Mobile Applications



If your applications have already been developed and you merely wish to use them on another system, you'll need to transfer the Mobile Development Studio objects (Applications, Menus, Users and VBA code and macros) from your current system to the destination system.

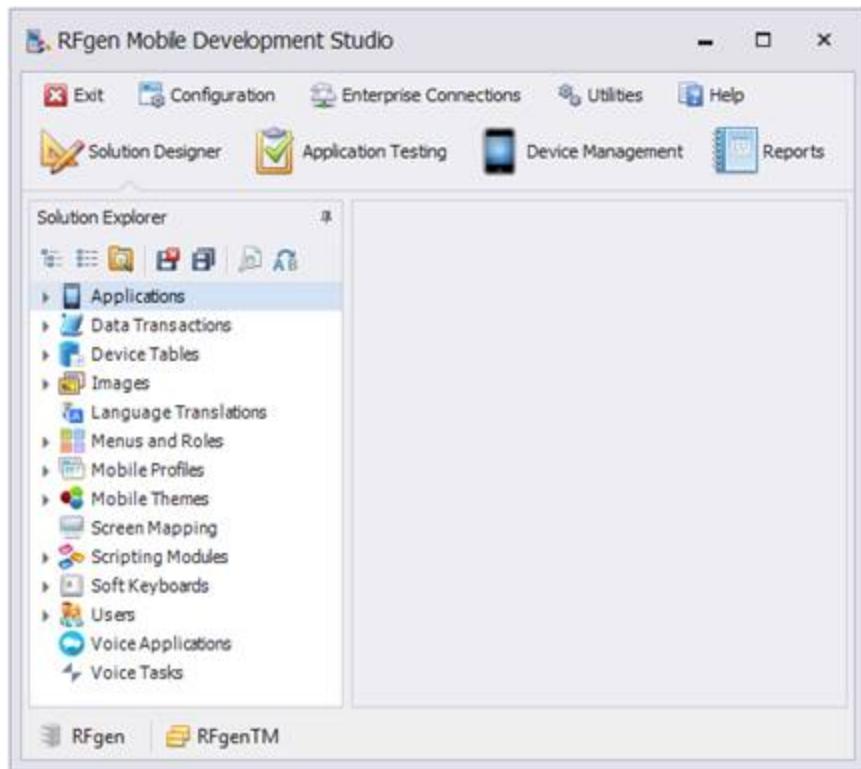
Objects are freely transferable from/to other solution databases, or other external files, for the following purposes: (1) production usage, (2) ongoing development, and (3) backup / retrieval.

To transfer an entire set of (same release) applications, and 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.

Note that you can choose to: (1) import/export to a standard *Windows File System* and/or (2) add a configuration file from the *RFgen Database* or (3) Import a file from RFgen System Files.



Mobile Development Studio



The **Mobile Development Studio** provides all the tools you need to create a mobile solution, test your solution, and manage the client after the RFgen client software has been installed.

The top menu enable you to: a) Configure the RFgen software; b) Setup Data Source connections; c) Configure as applicable ERP or host or web object connections; d) Configure your Transaction Database connections; e) test your apps and sessions; f) manage devices (i.e. connection sessions to the RFgen server); and g) report on performance.

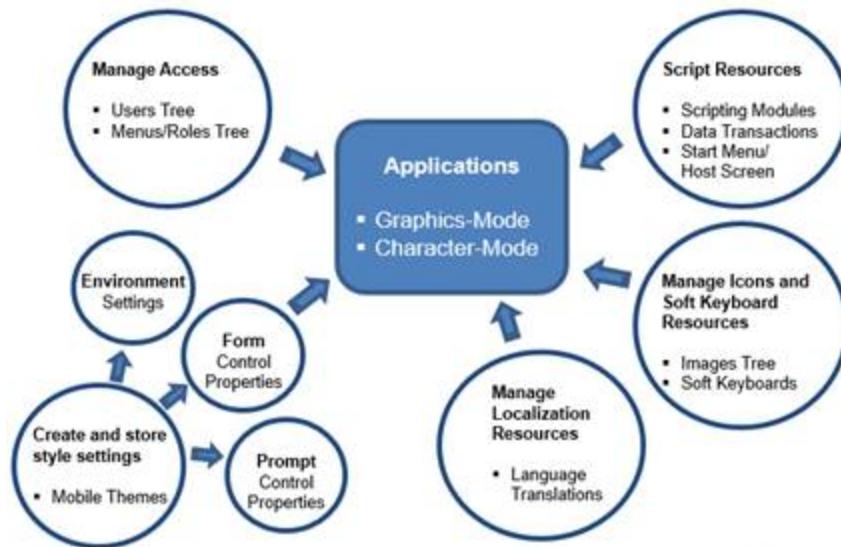
The **Solution Designer** panel provides the tools to:

- Design and create mobile applications for different methods of data collection.
- Script the application.
- Define users and menus to access the application.
- Build installation files for specific client platforms.
- Store and maintain objects for reuse or reference when building applications. (i.e. Globally accessible scripting modules, data transactions, translated terms/phrases, images, and soft keyboards).

If you have special voice-driven hardware equipment, the Solution Designer can be used to create voice applications that are executed via verbal commands.

Resources for Solution Development

The illustration below shows the various components and functions that can be used to create applications and organize how its accessed.



The **Applications** tree is used to design the visual aspects of your mobile application and basic validation for the data collection transaction. >

The **Data Transactions** tree 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 **Device Tables** tree allows you to setup various tables for storage on mobile devices (iOS, Android and Windows CE). These are used as a resource in Mobile Profile. This allows the user of the device to make changes to the table that is installed to the device.

The **Image** tree is a collection of images that can be used with your mobile application, menus or any other interface that may require an icon or picture

The **Language Translations** tree lists your translated/customizable strings that may be referenced from the code and enables you create new text translations on display forms and in the menu headers.

The **Menu and Roles** tree is used to organize applications and link a menu item to an application. If desired the RFgen Menu, can be imported from the RFgen system, and used to customize the appearance of your menus.

The **Mobile Profiles** tree is used to prepare a mobile device for use in the wireless / wired environment or the mobile environment.

The **Mobile Themes** tree lists customizable Android, Apple and Standard theme configuration settings for the mobile devices. It also serves as a centralized location for managing the defaults for common styling of controls. (i.e. BackColor, Fonts, Bezels etc.)

The **Screen Mapping** tree is used to add new hosts and capture screen (navigation) macros.

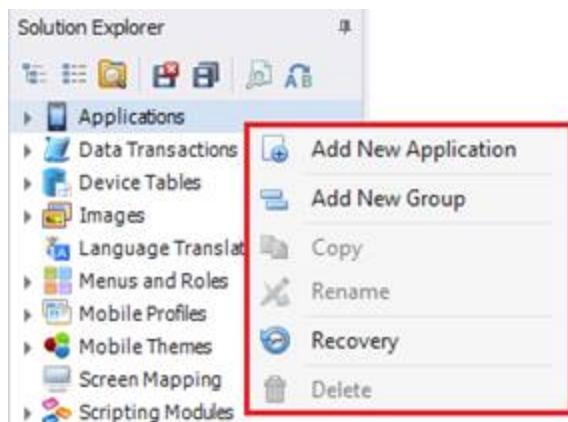
The **Scripting Modules** tree is used to manage global scripts that can be used by any application, timed event, transaction macro, etc.

The **Soft Keyboards** tree is used to create new soft keyboards for mobile applications as a supplemental means of entering data.

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

The **Voice Application/VoiceTasks** tree contains pre-defined scripts that execute on Vocollect hardware and interact with RFgen to provide the voice solution together with the backend data solution. Voice applications are unique in that they can only be developed with the scripts designed to execute tasks. The use of other resources such as the prompts/controls from the Tools tab, images, soft keyboards, etc. cannot be used to develop a voice application.

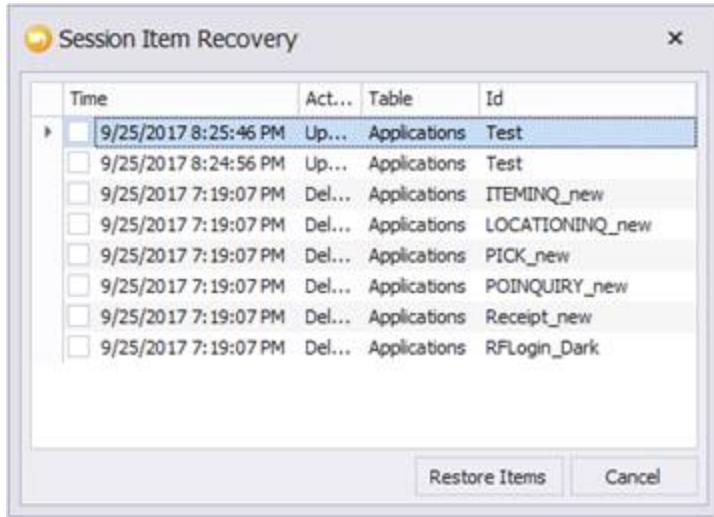
Solution Explorer Right-Click Menu



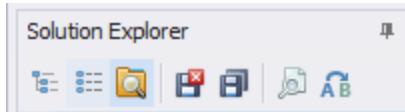
Use the right-click menu (or Context menu) to **add new nodes** (i.e. User, Role/Menu, Application, VBA Module etc.) or **add new groups** for any of the item in the Solution Explorer tree. Once you create and name your group, simply drag the selected item to the new folder.

You can also **open** specific items such as a user, an application, or application script as well as perform other standard operations such as **copy**, **rename**, or **delete** from this menu.

Recovery is similar to the undo feature except that it retains a history of all your changes since your last design session (or when you last closed the Mobile Development Studio). You can select the items you wish to restore and then click **Restore Items**.



Solution Designer/Explorer Menu Bar



Expands all nodes in the tree

Collapse all nodes in the tree

Closes all open objects

Saves all unsaved objects

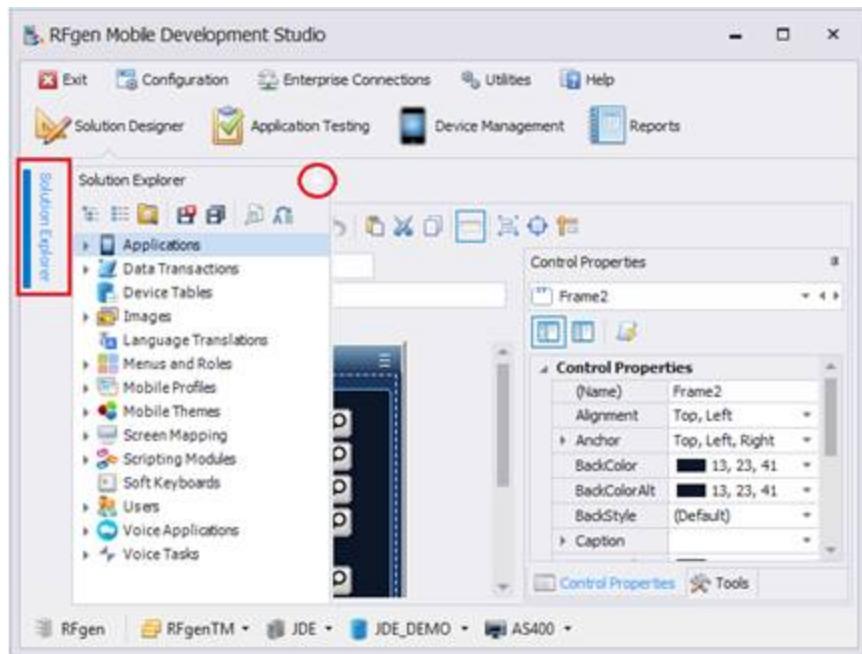
Finds or performs a match on text or formatted text in all files

Searches and replaces content in files

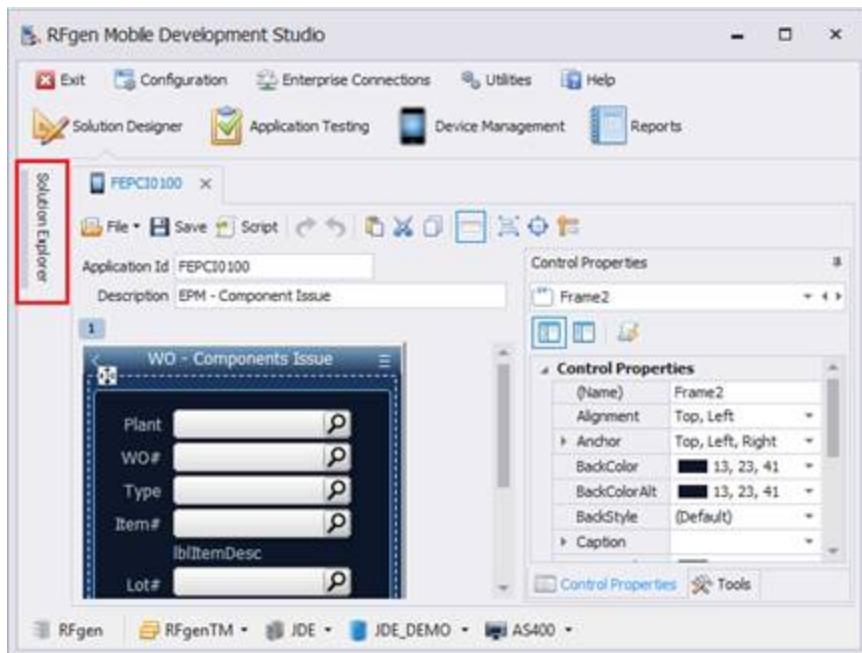
Replaces content in a file/application's script

Note: Double-clicking on a parent such as "Applications" automatically opens all child items in the group. Use the "Close All" folder icon to close child items.

Allows you to dock the Solution Explorer panel to the left side of your Mobile Development Studio screen and toggle the hide or show bar which then gives you more space in the Studio. When the panel is docked, you can click on the blue bar and hide the panel. To unhide the panel, click the blue bar. To redock a panel, click the pin again.

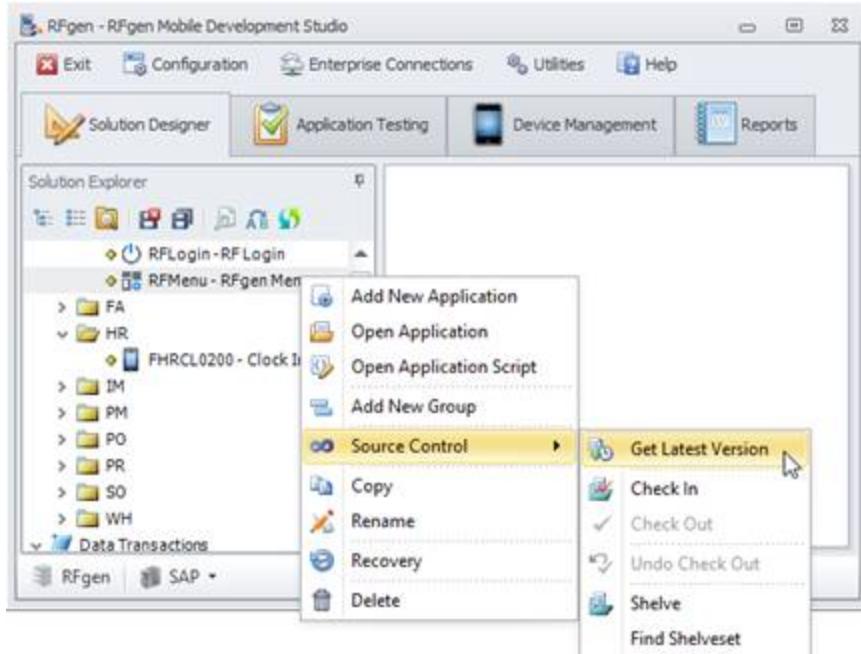


Example of the Solution Explorer tab in a docked, but viewable position.

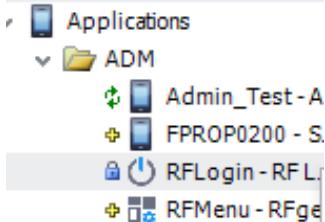


Example of the Solution Explorer tab in a docked, but hidden position.

Source Control Options in the Solution Designer



When a source control system such as Microsoft Team Foundation Server (TFS) is configured to manage objects in RFgen, a secondary icon will display next to each Solution Explorer object icon.



For example, a yellow + sign, green arrows, red checkmark or a blue lock will display to the left of the RFgen Login icon.

The Source Control Options manages Applications, Data Transactions, Device Tables, Images, Mobile Profiles, Mobile Themes, Screen Mapping Scripts, Scripting Modules or Soft Keyboard Codes. It does not manage Menus and Roles or Users.

Source Control Menu and Object States

The menu selection options are:

- +** The yellow plus sign means the object (i.e. application) has not been checked into the source control system. This also indicates RFgen is connected to a source control system.



The Check In menu option places new objects (objects that have not been checked in before) under source control. If the object was previously checked out, the source control system registers the check in as another version of this object. Notations of what has changed is usually required.



The Get Latest Version menu option retrieves the latest version (last modified) object that has been checked in. When you get the latest version, you are getting a copy of it, but leaving it "open" for others to also get a copy. To prevent others from working on the object, you must check the object out.

✓ The Check Out menu option retrieves a copy of the object and locks the file so others cannot modify it. This is known as a "check out."

✓ The red check mark means YOU checked out this object and are the owner. When the object is checked out, no one else can modify the object.

🔒 The blue lock means the object has been checked out by someone else. Your object will be a read-only copy. For example, the tools and script will be greyed' out until the object is checked back into the source control system.

⌚ The green cycle icon means the object's version status is unclear. This can occur if a user checked out the object, made code changes on an RFgen system that is not connected/configured to the Source Control system, then reloaded the object to the RFgen connected to the Source Control system. The unclear version status can also occur if a second user loads an object with the same name as another object already under Source Control system, but the object was not from the Source Control system (i.e. The object came from a system that was not under Source Control).



The Undo Check Out menu option reverses the process. The object reverts to the version it was at before it was checked out and unlocks the object so its available to other team members.

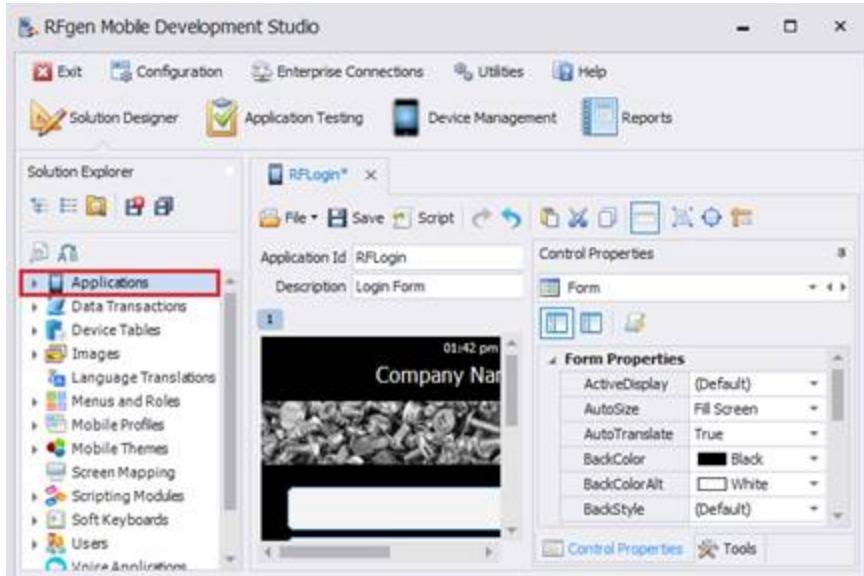
The Find Shelveset menu option helps you find shelved objects, view any comments entered, and check out the object.

Find ShelveSet

When you select **Find Shelve Set** from the menu Source Control, the **RFgen – Source Control Get Shelf** screen displays.

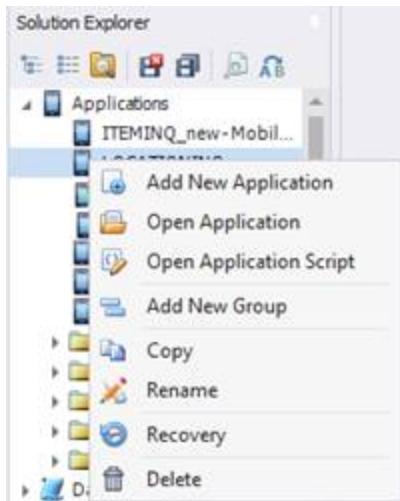
This allows you to check out the shelved version of the object and get a copy of the object and the changes. After you have reviewed it, you can then perform a Check In which will save the changes permanently.

Applications Tree



The Application tree is where you access the tools to create and customize applications (apps) for your own workflows.

Applications Right-Click Menu



The Applications' Right-Click menu options are similar to the Solution Explorer's Right-Click menu.

Applications Menu Bar



File and Save Menu

Save saves the current script to the solution database.

Opens the VBA scripting center for the application.

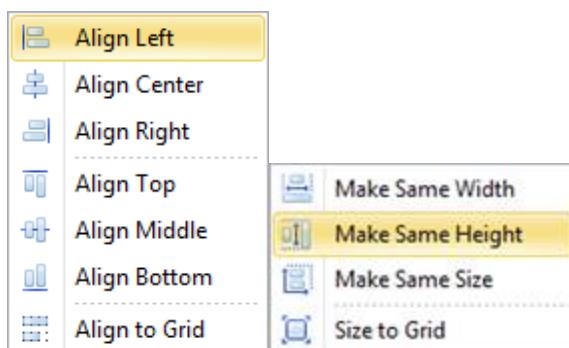
Undo and **Redo** the last task/action.

Removes the object selected.

Copies and **Paste** pastes the last item copied.

Display Invisible Prompts button toggles the designer screen to show the prompts that have their **Visible** property set to False.

Alignment icons help place a graphical prompt relative to the snap points which are configured under **Configuration > Desktop Preferences**. Here are the variety of placement options:

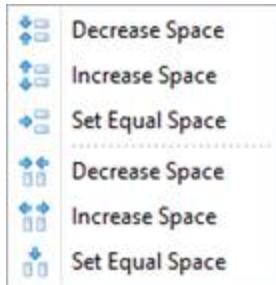


Control Alignment icons offers multiple alignment options when more than one control is selected. Using the selected prompt with the smallest prompt number, all alignments are aligned with that prompt.

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.

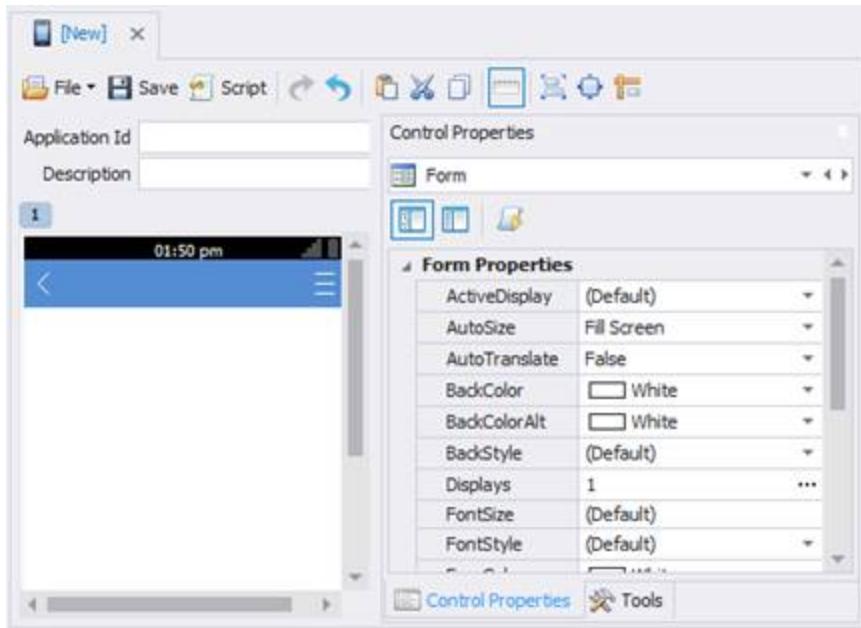


Control Sizing icons offer sizing options when more than one control is selected. Using the selected prompt with the smallest prompt number, all sizing adjustments are aligned with that prompt. The Size to Grid option stretches a prompt to the right most edge of the form size.



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.

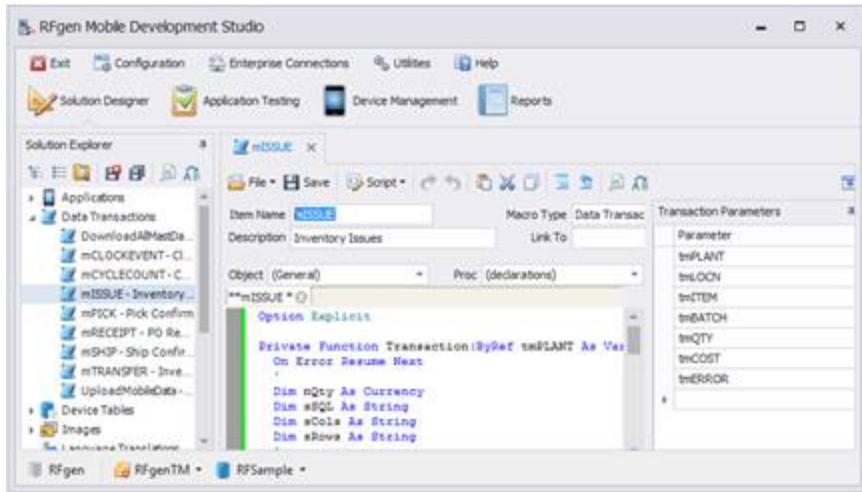
Applications



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.

Data Transactions

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

1. Right-click on **Solution Explorer > Data Transactions** and select **Add New Transaction** from the menu. If desired, you can also create a new folder to group your transactions by selecting **Add New Group** from the menu and moving your new transaction into this folder.
2. Enter an **Item Name, Description**. Set the **Macro Type** to Data Transaction.
3. 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.
4. 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.

Data Transactions Edit Menu



The **File > New Item** option clears all fields and provides a blank window for creating a new transaction macro.

The **File > Save** and **Save** options save the current script to the solution database where as **File > Save As** and **File > Rename** allows you to rename an existing file before saving it.

The **File > Delete** allows you to delete an item.

The **File > Close** closes the file.

The **Script > Syntax Check** option performs a syntax check of the script.

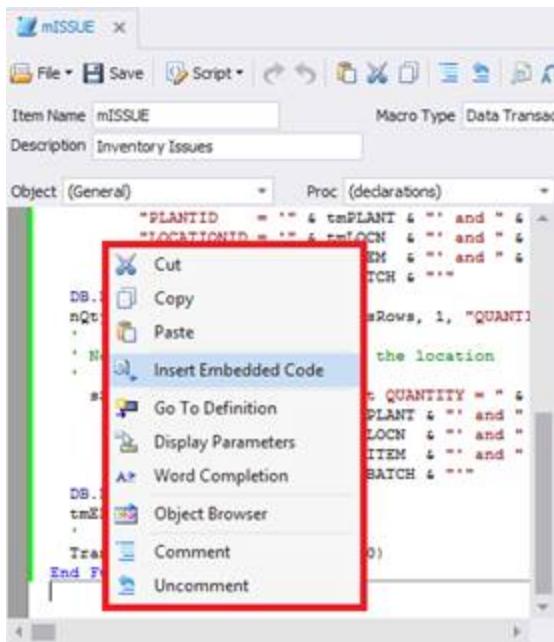
The **Script > Test Transaction** option will test the macro. (Requires it be linked to an active source.) It can also be used to test the Screen Mapping Host Transaction macros to be sure the keyboard recording is correct.

Also provided are **Undo**, **Redo**, **Cut**, **Copy** and **Paste**.

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

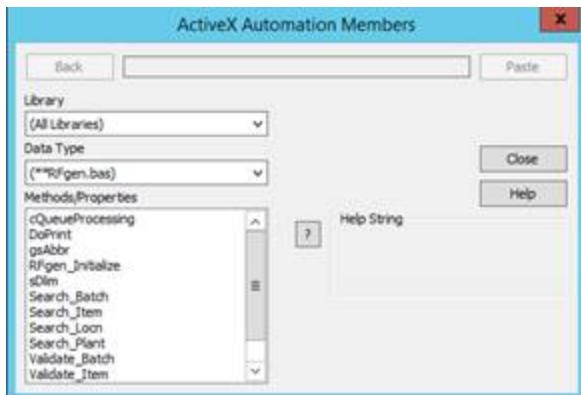
The **Find** and **Replace** icons allows the user to find text or replace text within the current application.

To Allow References to Global Scripts



To allow references to global scripts that are in a VBA module but are not globally available, follow these steps:

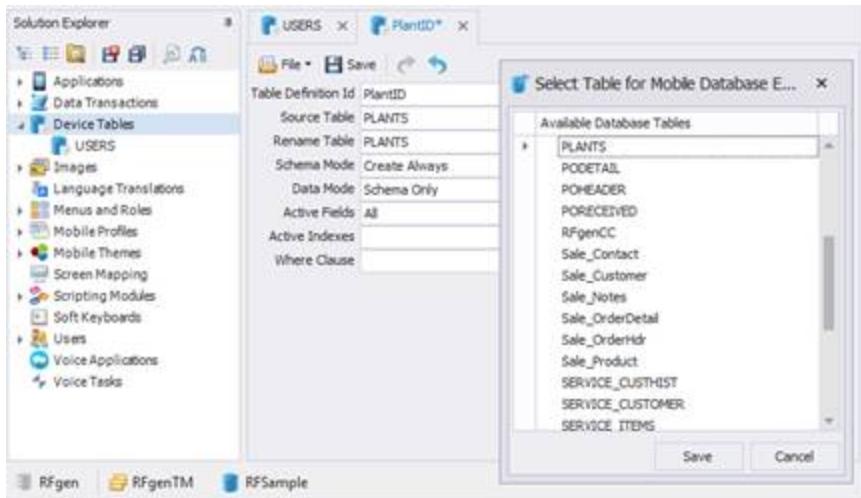
1. Right-click on any space in your scripting form to bring up the Script Edit menu shown above.
2. Select the **Object Browser**.
3. The **Active X Automation Members** window displays.



This lists existing VBA modules from **Solution Explorer >Scripting Modules**.

4. Check those modules you wish to include with the current macro. Modules designated Win32.bas and RFgen.-bas are automatically included for each macro.

Device Tables



The Device Tables allows you to have preconfigured tables for processing data on a device when the device is in batch or offline mode. The settings for each Table are then applied and provided as a mobile database source in the **Solution Explorer > Mobile Profiles – Mobile Database** category.

To add a new table or group, use the Right-click menu and fill in the Table form to configure the table and how its modified on the device.

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. Note that SQLite naming follows much stricter rules than other databases, so renaming a table might be necessary for Android or iOS platforms.

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 if the table already exists, it will be deleted and recreated on the mobile database.
- *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.

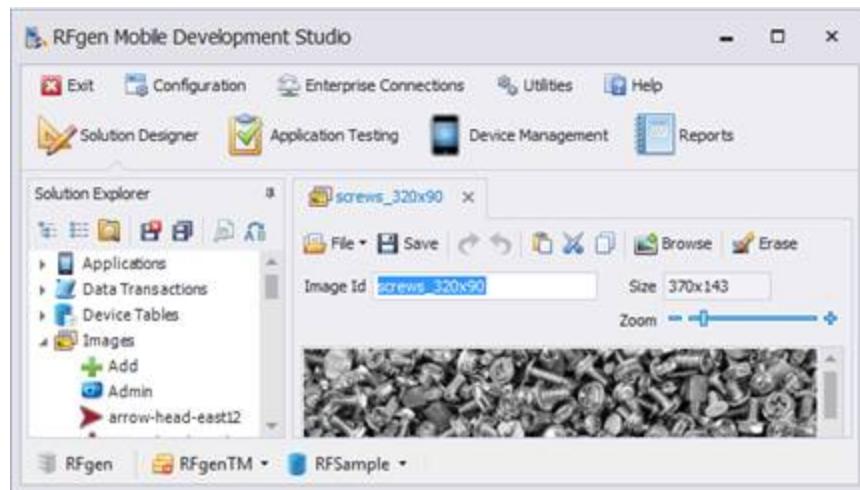
- *Clear and Copy* will delete the contents of the table and re-populate it with the new data from the server.
- *Copy Only* will place the data from the server on top of the existing data. Where it is the same, there is no change. Where it is different and depending on the keys, either it will overwrite 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.

Images



The Image Tree 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.

Size shows the image size in pixels but does not change the image size when its rendered on the application screens.

Zoom magnifies or miniaturizes the image but this does not change how the image will be rendered on the application screen.

Image Menu



The **File > New Item** option clears all fields and provides a blank window for creating a new transaction macro.

The **File > Save** and **Save** options save the current script to the solution database where as **File > Save As** and **File > Rename** allows you to rename an existing file before saving it.

The **File > Delete** allows you to delete an item.

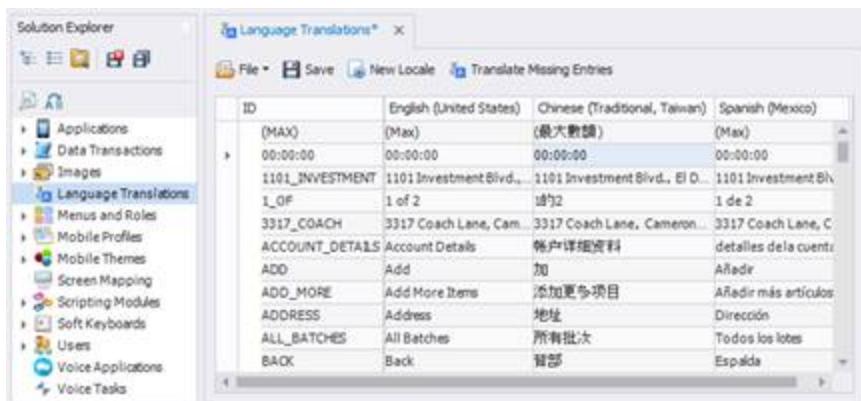
The **File > Close** closes the file.

Also provided are **Undo, Redo, Cut, Copy and Paste**.

The **Browse** icon is used to import images to the Images tree. Supported formats are jpg, gif, bmp, png and jpeg.

The **Erase** button will erase the image in the form but leave the Image ID in the Image tree.

Language Translations



The **Language Translations** feature automatically translates lists of customizable strings into the local of your choice.

In order for a translated string to appear in a mobile app, the language name (i.e. English (United States), Chinese (Traditional, Taiwan), Spanish (Mexico) etc.) must be:

- 1) Defined in the Language Translations Locale grid (shown above).

AND

- 2) The language's corresponding locale code must be set in the application's Form: Displays: Manage Alternative Form Displays: Locale property.

For example, if translating to Spanish, the language name Spanish (Mexico), the Locale code is "es". Numeric values (e.g., 1033 for English) also called Locale IDs (LCIDs), are **not** supported when specifying Display locales.

To view a list of supported languages, refer to the first table (the "phrase-based model") at <https://cloud.google.com/translate/docs/languages>

Language Translations

ID	English (United States)	Chinese (Traditional, Taiwan)	Spanish (Mexico)
(MAX)	(Max)	(最大數額)	(Max)
00:00:00	00:00:00	00:00:00	00:00:00
1101_INVESTMENT	1101 Investment Blvd., El D...	1101 Investment Blvd., El D...	1101 Investment Blv...
1_OF	1 of 2	1的2	1 de 2
3317_COACH	3317 Coach Lane, Cam...	3317 Coach Lane, Cameron...	3317 Coach Lane, C...
ACCOUNT_DETAILS	Account Details	帐户详细资料	detalles de la cuenta
ADD	Add	加	Añadir
ADD_MORE	Add More Items	添加更多项目	Añadir más artículos
ADDRESS	Address	地址	Dirección
ALL_BATCHES	All Batches	所有批次	Todos los lotes
BACK	Back	背部	Espalda

The **Language Translations** feature automatically translates lists of customizable strings into the local of your choice.

In order for a translated string to appear in a mobile app, the language name (i.e. English (United States), Chinese (Traditional, Taiwan), Spanish (Mexico) etc.) must be:

- 1) Defined in the Language Translations Locale grid (shown above).

AND

- 2) The language's corresponding locale code must be set in the application's Form: Displays: Manage Alternative Form Displays: Locale property.

For example, if translating to Spanish, the language name Spanish (Mexico), the Locale code is "es". Numeric values (e.g., 1033 for English) also called Locale IDs (LCIDs), are **not** supported when specifying Display locales.

To view a list of supported languages, refer to the first table (the "phrase-based model") at <https://cloud.google.com/translate/docs/languages>

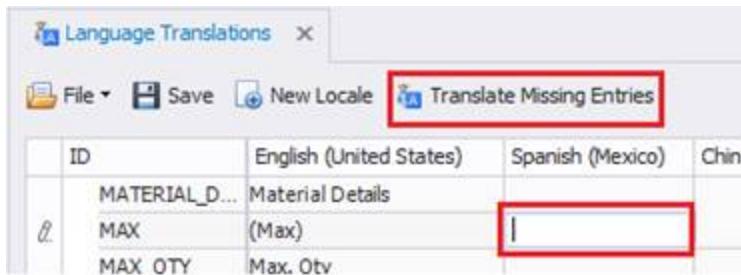
To Translate Strings into Multiple Locales

ID	English (United St...)	Spanish (Mexico)	Chinese (Traditi...
(MAX)	(Max)	(Max)	(最大數額)
00:00:00	00:00:00	00:00:00	00:00:00
1101_INVESTMENT	1101 Investment Blvd., El D...	1101 Investment Blvd., El D...	1101 Investment Blv...
1_OF	1 of 2	1的2	1 de 2
3317_COACH	3317 Coach Lane, Cam...	3317 Coach Lane, Cameron...	3317 Coach Lane, C...
ACCOUNT_DETAILS	Account Details	帐户详细资料	detalles de la cuenta
ADD	Add	加	Añadir
ADD_MORE	Add More Items	添加更多项目	Añadir más artículos
ADDRESS	Address	地址	Dirección
ALL_BATCHES	All Batches	所有批次	Todos los lotes
BACK	Back	背部	Espalda

1. Click on **Solution Explorer Tree > Language Translations**. A blank list of IDs and text displays.

Note: If you performed an upgrade from 5.0 and your database included translated strings, they would appear in the list below. If you want to import strings, refer to **Utilities > Import**.

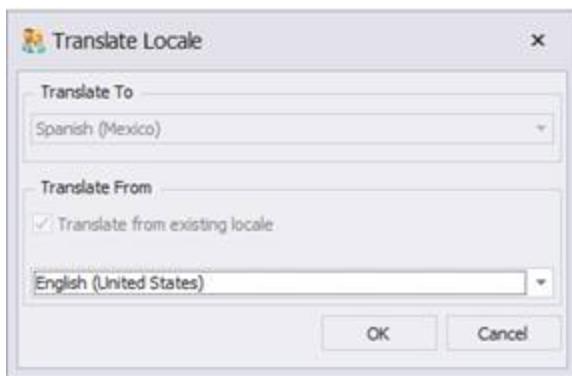
2. Enter the text ID and text to be translated. In the example above the resource strings are in English.
Note: The user can reorder columns as desired.
3. Click on **New Locale** to add new locales. Spanish and Chinese were added in the example above. You can also type the locale to access the specific language faster.
4. To translate an entire list, or to fill in any missing translations for a locale, click in the empty cell of the desired locale (i.e. the empty cell for "(Max)" under the Spanish column).



The screenshot shows a software window titled "Language Translations". At the top, there are menu options: File, Save, New Locale, and Translate Missing Entries (which is highlighted with a red box). Below the menu is a table with four columns: ID, English (United States), Spanish (Mexico), and Chinese. There are three rows of data:

ID	English (United States)	Spanish (Mexico)	Chinese
MATERIAL_D...	Material Details		
MAX	(Max)		
MAX_ OTY	Max. Oty		

5. The **Translate Missing Entries** button is enabled.
Click on **Translate Missing Entries**. The Translate Locate windows displays.



6. Select the locales you want to **Translate From** and **To** and click **OK**. The entire list will translate automatically to the selected Locale if the translation exists for that text string. (Sometimes an equivalent does not exist in the locale, in which case the same English text string is used.)

7. This example shows the steps for Spanish and Chinese translations.

ID	English (United States)	Chinese (Traditional, Taiwan)	Spanish (Mexico)
(MAX)	(Max)	(最大數值)	(Max)
00:00:00	00:00:00	00:00:00	00:00:00
1101_INVESTMENT	1101_Investment Blvd., El D...	1101 Investment Blvd., El D...	1101 Investment Blv...
1_OF	1 of 2	1 of 2	1 de 2
3317_COACH	3317 Coach Lane, Cam...	3317 Coach Lane, Cameron...	3317 Coach Lane, C...
ACCOUNT_DETAILS	Account Details	账户详细资料	detalles de la cuenta
ADD	Add	加	Añadir
ADD_MORE	Add More Items	添加更多项目	Añadir más artículos
ADDRESS	Address	地址	Dirección
ALL_BATCHES	All Batches	所有批次	Todos los lotes
BACK	Back	背部	Espalda

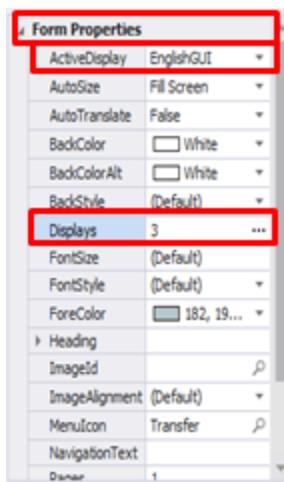
8. You are now ready to link these resource IDs to Captions in your Application forms.

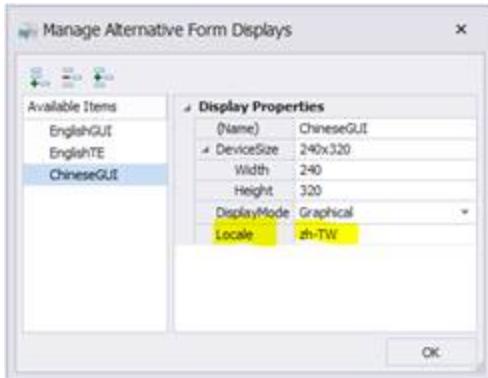
To Localize Text in Graphical Displays



Before you localize strings associated with a control, ensure the string to be localized has been entered in the **Solution Explorer > Language Translations** table.

1. From **Solution Explorer > Applications > [your application]**, open your application and select the **Form**. From the Form select the **Displays** property. Click on the dots ...



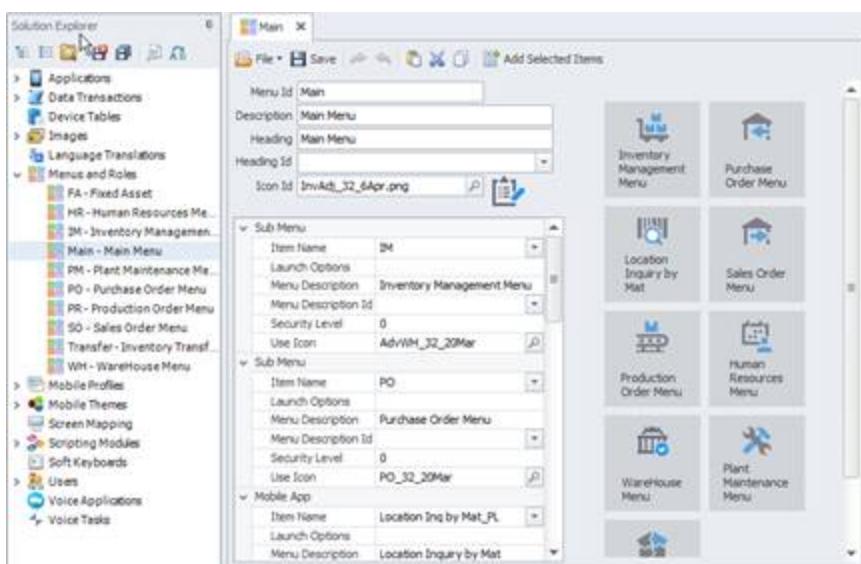


2. Create a new Display GUI by clicking on the + icon on the far left under the **Manage Alternative Form Displays** window.

Note: The Locale value here needs to match the Locale set in Language Translations.

3. To translate a caption, click on the property that contains the caption to be translated. If the drop down menu is blank, you many need to add the text string ID to the **Solution Explore > Language Translation** table.
4. In the Caption property, expand the drop-down list.
An ID list displays. Select the text ID from this list. Click **Save**.
5. In the Form, change the **Active Display** property to the Display which is set the desire Locale.
6. The translated text should appear in the prompt on the form.

Menus and Roles

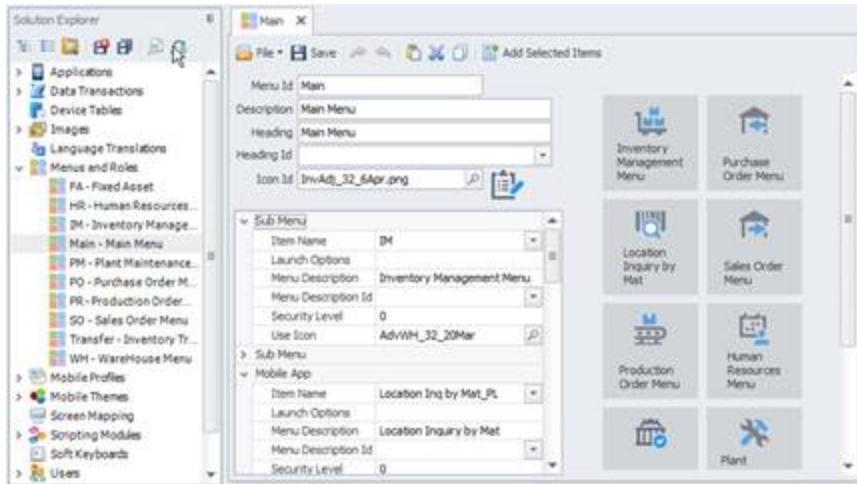


Roles and Menus are designations you create and assign to groups of existing applications for organization purposes. The *Main Menu* is an example of a parent menu which can be setup as the first menu that displays after a user logs in on their mobile device. The Sub Menus such as the *Inventory*, *Purchase Order*, *Sales Order*, and *Human Resources Menus* display when the user selects the *Main Menu*.

Once the menus, submenus and applications are setup, you can assign the menus to user accounts in the **Solution Explorer > Users** tree. This helps control which Mobile Apps a user will can access on their mobile device or Windows desktop system.

NOTE: The *User Management Console* also provides features that allows Console users to setup Menus and Users without providing Console users full access to everything on the server. This enables database updates from the server or the console by different users. For more details, refer to the *User Management Console* section in this guide.

Menus and Sub Menus



Menu Fields

The **Menu Id** is the name of the menu that is used when assigning the menu to users.

The **Description** is required and only shows in the Menu list.

The **Heading** value will be placed at the top of the menu and is optional.

The **Heading ID** allows you to localize the Heading when the source ID and text strings have been setup in the **Solution Explorer > Language Translations** sections of *Mobile Development Studio*, and the locale codes have been set in the **Configuration > Application Preferences** menu.

The **Icon Id** references an Image resource to represent this menu in case the menu is presented to the user with graphical elements.

Sub Menu Fields

These define the child menus to the parent menu; they are used to categorize and group mobile applications. For example, the *Inventory*, *Purchase Order*, *Sales Order*, and *Human Resources Menus* are the “Sub Menus” that display when a user clicks the *Main Menu*.

The **Item Name** is the sub menu name.

The **Launch Options** can be used to define how the sub menu is launched. See *Launch Option Details* below.

The **Menu Description** appears under the Sub menu icon.

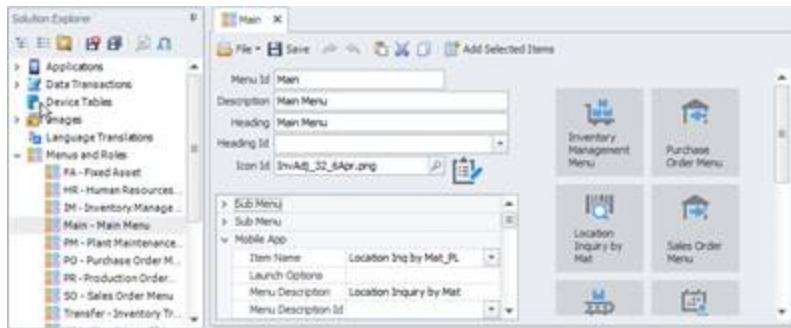
The **Menu Description ID** field is optional. Its used if you want the Menu Description translated or localized.

The **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.

The **Use Icon** allows you to select the image for this menu.

Mobile App Fields

These define the application icons that display under a Menu (Main Menu) or Sub Menu. The icons and descriptions (items in the green boxes) are examples of what the user would see on his or her Mobile Device after they selected the menu (i.e. Inventory Menu).



The **Item Name** contains the name of the application that will be launched when its icon is selected.

The **Launch Options** can be used to define how the application is launched. See *Launch Option Details* below.

The **Menu Description** appears under the Sub menu icon.

The **Menu Description ID** field is optional. Its used if you want the Menu Description translated or localized.

The **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.

Launch Option Details

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.

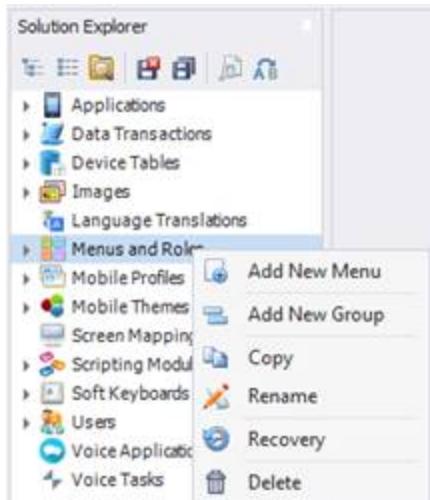
In a case where you **want the user to select from two variations** of the same application, add the application to the menu twice, give them different descriptions and then use two different command **Options**. Using the format –VARNAME=VALUE will create variables with values that can be referenced 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.

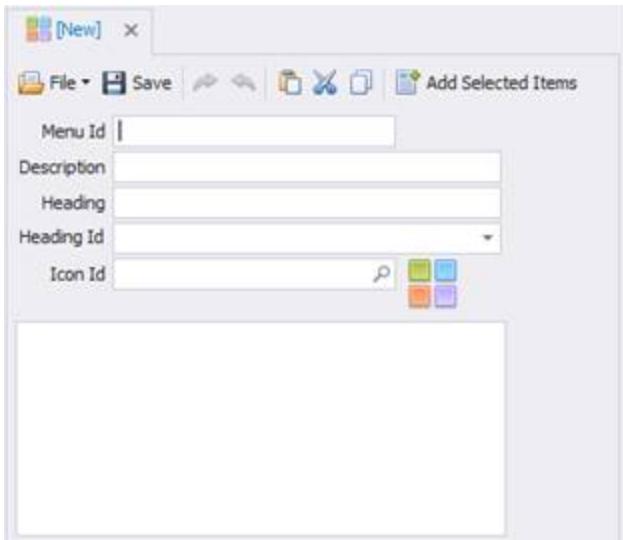
Making the application name a dash, supports comment lines within a menu.

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.

To Add Menus



1. From the **Solution Designer > Solution Explorer**, right-click on **Menus and Roles** and select **Add New Menu**. A blank form displays.



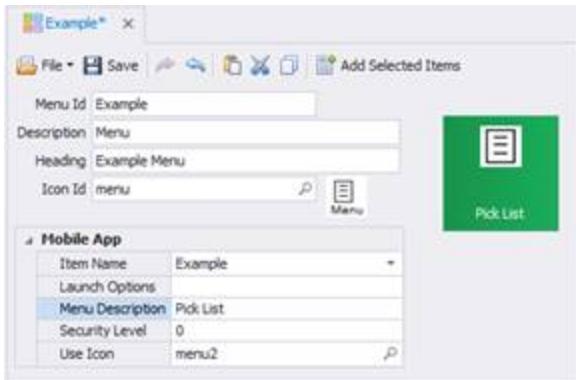
2. Complete the Menu Id, Description and Heading. The Icon ID and Heading are optional.

The **Heading ID** is used for localizing the Heading. Localization of Heading strings requires the Heading be setup as a string **ID** in the **Solution Explorer > Language Translation > Locale** grid and the **Solution Explorer > Configuration > Application Preferences: Language Set** Locale Name be set to the same Local as the one used in the Language Translation grid.

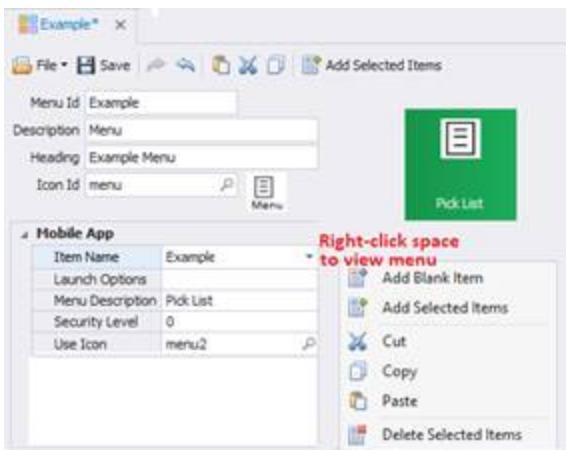
3. Click on **Add Select Item(s)**. A window similar to the one below displays.



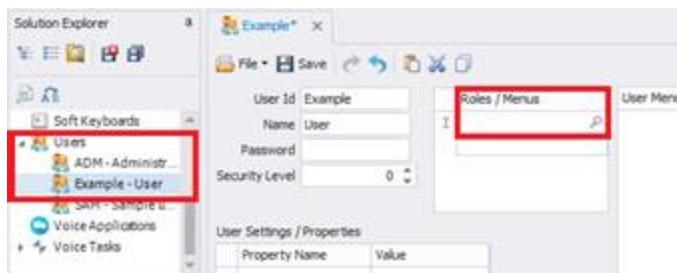
4. Expand the tree to view the item to be selected. Check the application or another menu from the list and click **OK**.
5. A **Mobile App** grid displays with the new entry.



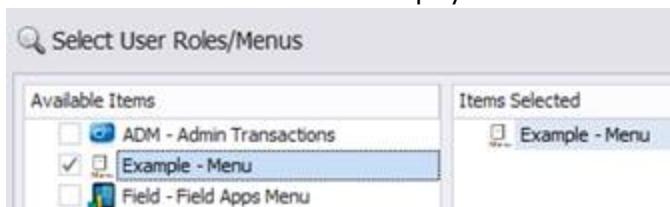
- From this point, you can also add new menus by right-clicking on a blank space.



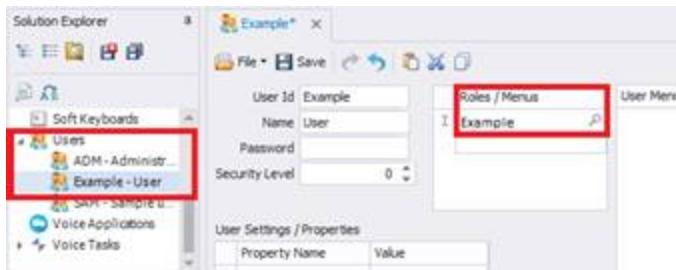
- Assign a User(s) to this menu by selecting an existing user or adding a new user from **Solution Explorer > Users** list.



A screen similar to the one below displays.

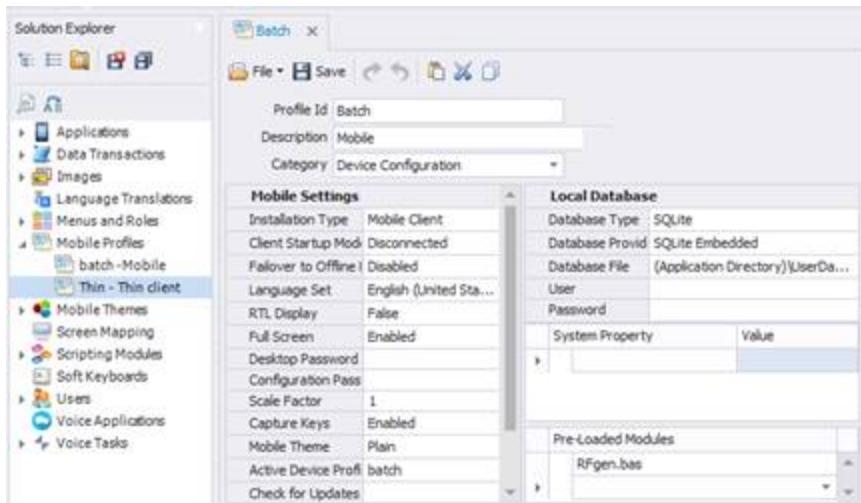


Below is an Example User assigned to an Example Menu



8. Click **Save**.

Mobile Profiles



The **Mobile Profile** feature is used to prepare a mobile device for installation to a client device. There are two profile types: Batch or Thin.

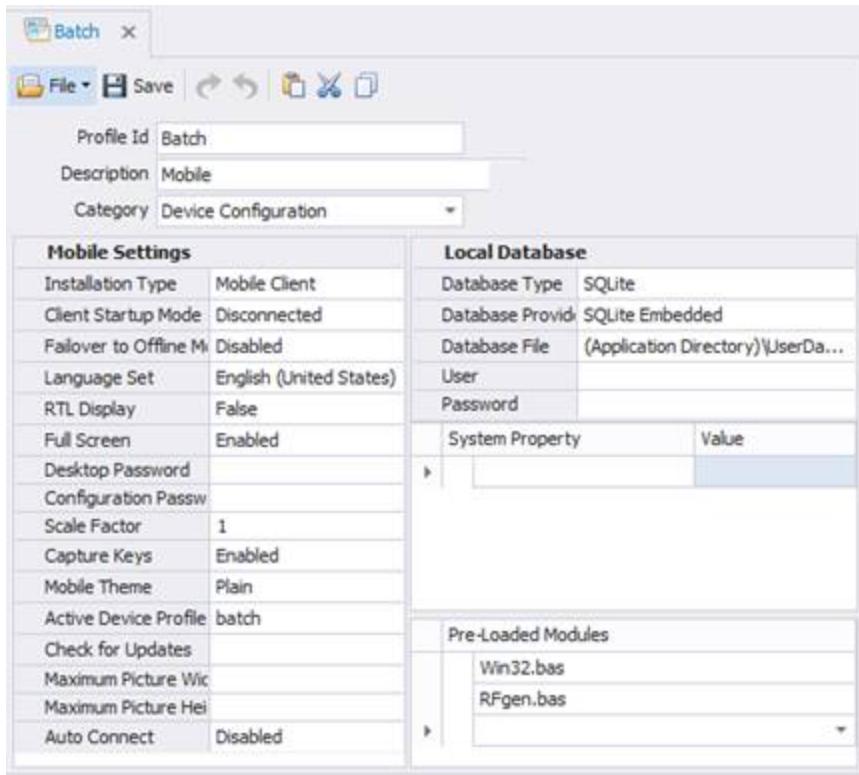
Thin Client The traditional wireless real-time interface to the server where the mobile device is restricted to the RF environment.

Batch 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.

Note that the Mobile Settings for Thin and Batch are different. For example, Thin Clients do not require a Local Database setup because the application and data are “projected” from the server and is not saved locally. However, if a Batch Client disconnects from the network, collection may continue as data is saved to the device’s local database.

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



(For Thin and Batch Clients)

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 client choices are **Thin** or **Mobile**.

Client Startup Mode refers to which state the mobile client will use when started, connected, disconnected. 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.

Category: Device Configuration

Local Database

(For Batch Clients only)

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

Database 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: Server Connections

(For Thin and Batch Clients)

The **Server Connections** category records which servers the mobile device can access and what type of connection will be used.

The **Enterprise Application Servers** table 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

(For Batch Clients Only)

The Mobile Applications category applies only to Batch (Mobile) mode clients. 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.

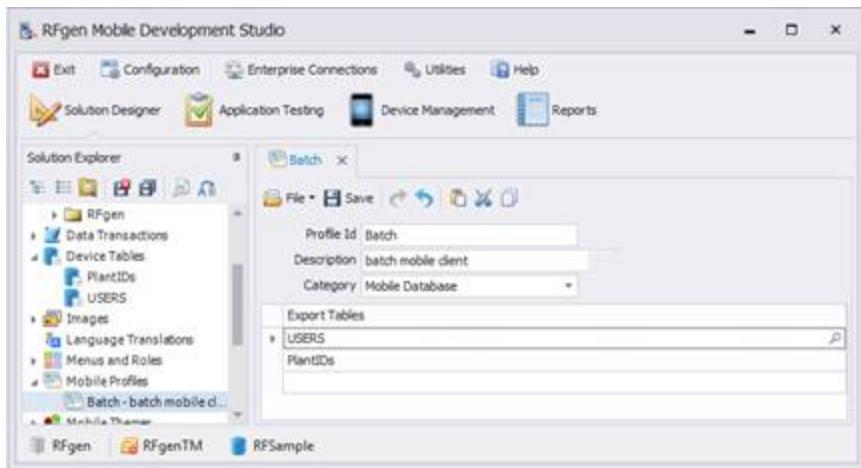
Select the required items or use the toolbar to help in the selection.



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

(For Batch Clients Only)



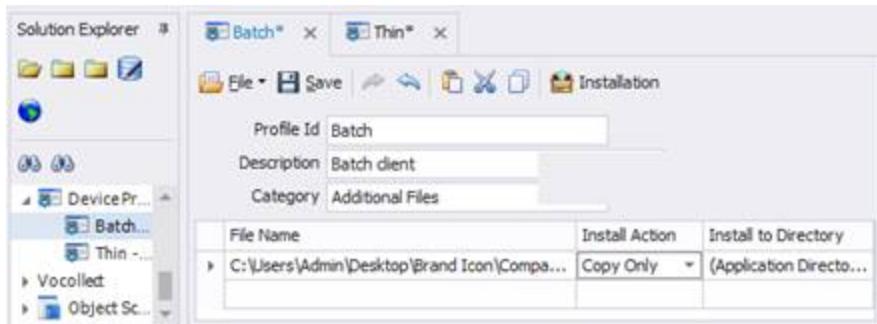
The **Mobile Database** category is only applicable to Mobile-mode clients. The sources come from the **Solution Explorer > Device Tables** tree.

For details on configuring the sources, see *Device Tables*.

You may select multiple tables for faster entry. Click on Select Table to return to the previous screen.

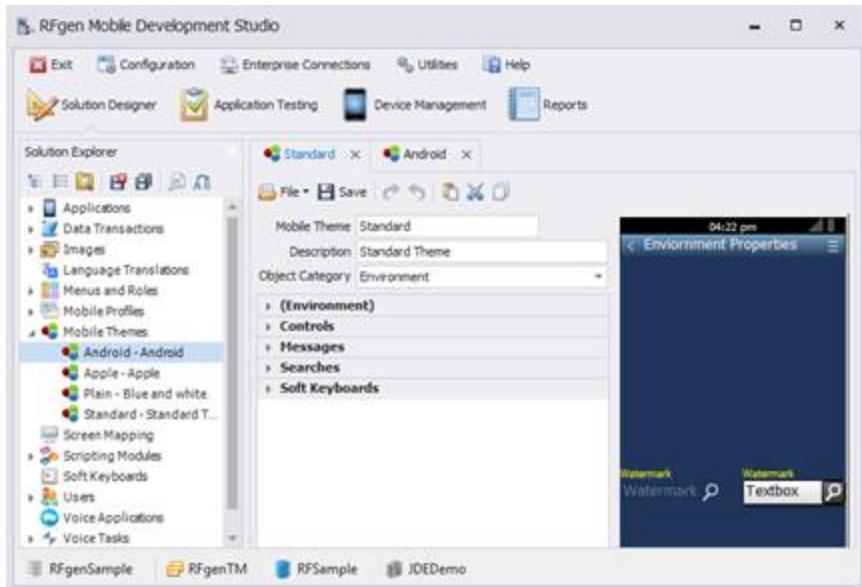
Category: Additional Files

(For Batch and Thin Clients)



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.

Mobile Themes



Mobile Themes is a collection of graphical properties that can be assigned to new or existing applications so the individual controls will have a consistent look and feel when they are assigned a specific theme. You access this feature from the **Mobile Solution Development Studio > Solution Explorer > Mobile Application** node.

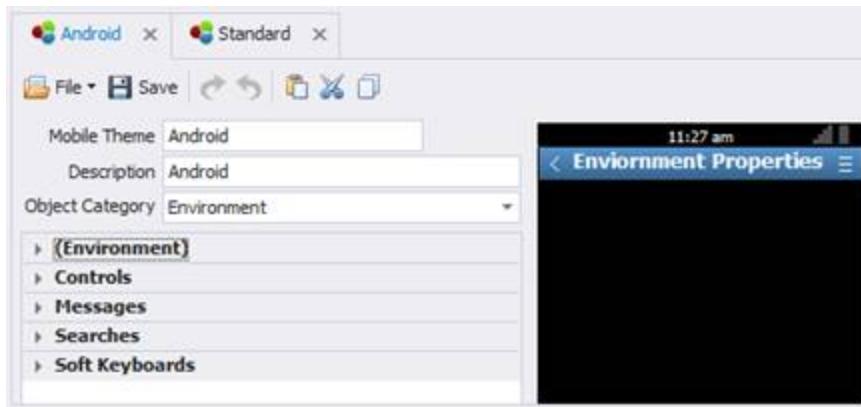
The property values set here are automatically applied to the Prompt/Control properties of a new application at the design time and/or at run time when:

- Configure > Application Preferences is set the specified theme and
- The matching Prompt/Control's property in an application is set to Default or Automatic.

You can override the property setting inherited from the theme when you change that value of the property in the application.

Note that whenever the Mobile Theme is changed in **Desktop Preferences – Mobile Theme** (i.e. from "Standard" to "Android"), the changes to open applications is immediate. Clicking on a selected node in the Mobile Theme tree does not change the active theme for an open or new application.

Mobile Theme Overview



The **Mobile Theme** is the file name of your theme and is also the name that displays under Configuration > Application Preferences. Themes are used to consolidate your look and feel settings for all elements of an application so that they can be efficiently applied to your applications.

The **Description** is your description of the Mobile Theme

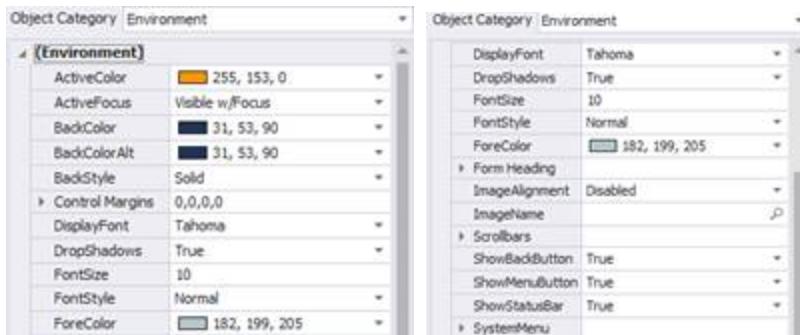
The **Object Category** includes the list of all the categories of objects used to create your theme.

Mobile Themes Right-Click Menu

You can add new themes, copy, modify, or delete existing themes using the right-click menu from the Menu tree. You can also rename an existing theme or copy an existing theme then rename it.

Mobile Themes: Environment

This category sets the themes for elements common to all graphical application. For example, all graphical applications are comprised of a form, a heading, text, and some navigation elements.



ActiveColor – the color of the border of the prompt with focus if the ActiveBorders property is set to Active Border.

ActiveFocus – 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.

BackColor – the primary color of the background of an application screen depending on the BackGradient value

BackColorAlt – 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.

BackStyle – set to Solid or a direction for the fade effect.

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

DisplayFont – the default font for all graphical applications using this theme.

DropShadows – True will display a drop shadow when the prompt is displayed in the Standard, Active Border or Visible w/Focus modes. False will not display the drop shadow.

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.

Form Heading



The form heading is comprised of a black strip which is permanent (no theme properties), the Back button (see ShowBackButton), the Menu Strip button (see SystemMenu).

Align - Aligns the form header according to the selected setting (i.e. Top, Left; Top, Center etc.)

BackColor, BackColorAlt, BackStyle, FontSize, FontStyle and ForeColor are applied the same as in other examples.

Bevel – sets the degree of roundness of the corners of rectangular objects such as a buttons, panels, tab controls, textboxes. “0” will no bevel edges; “2” is the average setting for a rounded corner; “4” is typically used on objects in Android. Its currently not used in Form Heading.

BorderColor – sets the border color of an object.

Height – sets the form’s heading height in pixels

Visible – turn the form’s heading on or off. True is always visible; False is hidden.

ImageAlignment – If the application has a background image, this positions the image relative to the edge of the form heading. If an image is present, it can also be disabled, stretched, or tiled.

ImageName – loads a specified image resource to the form.

ScrollBars

If the scrollbar property is used on the Application, the environment theme setting for the scrollbar will be applied.

Arrows - set the coloring scheme for the end points of the slider in the scrollbar. BackColor, BackColorAlt, BackStyle and BorderColor are applied the same as in other examples.

Arrows Pressed – This sets the color values when the scrollbar arrow button is selected.

BackColor, BackColorAlt, BackStyle and BorderColor are applied the same as in other examples.

Buttons - set the coloring scheme for the buttons in the scrollbar. BackColor, BackColorAlt, BackStyle and BorderColor are applied the same as in other examples.

Button Pressed – sets the color values when the scrollbar button is selected.

BackColor, BackColorAlt, BackStyle and BorderColor are applied the same as in other examples.

Button Style – set the style and placement of the scrollbar. Values are None, Separate, Top, Right, Bottom, Left

Size – sets the thickness of the slider in pixels. 16 is the default.

Slider – sets the color and styling theme for the scrollbar’s slider.

BackColor, BackColorAlt, BackStyle and BorderColor are applied the same as in other examples.

BorderStyle – sets the scrollbar slider appearance. Values are None, Flat, Sunken or Raised.

Margin – controls the spacing between the slider and inside edges of the scrollbar. Values are Left, Top, Right and Bottom.

Slider Pressed – sets the appearance of the slider when its selected for the Scrollbar.

BackColor, BackColorAlt, BackStyle and BorderColor are applied the same as in other examples.

ShowBackButton – displays (True) or hides (False) the back button on the Form Heading. See image under Form Heading.

ShowMenuButton – displays (True) or hides (False) the menu button on the form heading. See image under Form Heading. This menu button is hard coded to launch the menu strip (also called the system menu) on a device.

MenuButtonStyle – toggles the Form header menu to display as horizontal lines or vertical dots.

ShowStatusBar – If set to True, the system menu will always display; if set to False it will only display when launched from the menu.

System Menu

The system menu is a menu launched from the menu icon (located on the form's header.) This sets the themes for the system menu which allows the user to work with functions that interact with the device (i.e. Exit an application, bring up a soft keyboard.)

AutoHide – True will hide the system menu (also called a menu strip) at runtime, until its launched by the menu button typically located in the upper right corner of the form. False will display the system menu at runtime.

System Menu BackColor, BackColorAlt, BackStyle, FontSize, FontStyle, and ForeColor are applied the same as in other examples.

Dock – Sets the position of the system menu relative to the device screen. The values are: bottom, top, left or right.

IconMode – Selects the style for how the menu items display in the system menu. The values are Image + Text, Image Only or Text Only.

IconSize – Sets the height and width of the image used in the SystemMenu's IconMode.

ItemSize – Sets the overall size of the objects in the System Menu.

MenuState – Sets the visibility of the System Menu when an application or login screen is launched. The values are Visible, Hidden or Locked. Hidden will not display the menu until the user clicks on the menu icon. Visible will display the System Menu when the application launches if AutoHide is set to True. Locked will display the System Menu on top of the application or display the System Menu by pushing the application screen aside.

Pressed – Sets how the System Menu option looks when an option is selected.

The BackColor, BackColorAlt, BackStyle and ForeColor are applied the same as in other examples.

Mobile Themes: Controls

This sets the theme for prompts or controls when they are displayed, are pressed, or when a field is edited in an application.

Mobile Themes: Buttons

This sets the themes for the Button control.

BackColor – the primary background color in a control.

BackColorAlt – the secondary background color in a control.

BackStyle – sets the background colors as blended/solid if BackColor and BackColorAlt are used.

Bevel – sets roundness of the corners of a square or rectangular-shaped object such as a button, panel, tab control, or textbox. “0” will no bevel edges; “2” is the average setting for a rounded corner; “4” is typically used on objects in Android.

BorderColor – sets the color of a control’s border.

BorderStyle – sets the style as None, Flat, Sunken, or Raised.

ForeColor – sets the color of the font if a caption is used.

Button Pressed - sets the color values for this control when its selected.

BackColor, BackColorAlt, BackStyle and ForeColor are applied the same as above.

Mobile Themes – Labels

This sets the themes for the Label control.

BackColor – the primary background color in a control.

BackColorAlt – the secondary background color in a control.

BackStyle – sets background colors as blended/solid if BackColor and BackColorAlt are used.

Bevel – sets roundness of the corners of a square or rectangular-shaped object such as a button, panel, tab control, or textbox. “0” will no bevel edges; “2” is the average setting for a rounded corner; “4” is typically used on objects in Android.

BorderColor – sets the color of a control’s border.

BorderStyle – sets the style as None, Flat, Sunken, or Raised.

ForeColor – sets the prompt label fore color (color of the font).

Mobile Themes – List Boxes

This sets the themes for List Boxes.

BackColor, BackColorAlt, BackStyle, Bevel, BorderColor, and ForeColor are applied the same as in other examples.

Heading – sets the theme for list box header.

Align – sets the heading’s alignment for the.

BackColor, BackColorAlt, BackStyle, Bevel, BorderColor, and BorderStyle are applied the same as in other examples.

FontSize, FontStyle, and ForeColor are applied the same as in other examples.

Height – sets the ListBox Heading’s height in pixels.

Visible –True display the heading; False hides it.

LineColor – the color of the lines within the grid if they are shown.

ListStyle – will hide or show the lines between rows and columns. None hides all the lines; Horizontal displays the lines between rows; Vertical displays the lines between columns; and Both displays lines between rows and columns

Row Selector – sets the coloring and styling of a selected row.

BackColor, BackColorAlt, BackStyle, Bevel, and ForeColor are applied the same as in other examples.

Transparency – sets the degree of transparency in a selected row.

Visibility – enables row selection (True); disables it (False)

Row Style – sets the theme for row elements

AltBackColor – sets the primary background color of a row when UseAltColors is True.

AltForeColor – sets the primary foreground color of a row when UseAltColors is True.

Padding – sets the row height in pixels for the List Box.

UseAltColors – Enables rows to use alternating colors (True); disables it (False).

ScrollBars – set to None, Horizontal, Vertical or Both to have the List Box and MenuList controls show or hide the default scrollbar. This is only the default and can be overwritten in the form design.

Mobile Themes: Menus

This sets the theme for application menus and submenus. Note that System Menus (Strip Menus) are different from application or function menus. To set System Menu (Strip Menu) see Environment.

AutoEntries are items that are automatically entered into the Menu. For example, you can list "Logoff" as a menu item that automatically will list with any application's menu and when launched, will log the user out of the application.

AddItems – select if the menu should have additional options like a Logout, Backup menu option or both. Choose "None" if you don't want a either.

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

Backup ID – lists the ID and its associated text in a grid.

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

Logout ID – lists the ID and its associated text in a grid.

BackColor, BackColorAlt, BackStyle, Bevel, BorderColor, and BorderStyle are applied the same as in other examples.

Button – sets the button properties for the Menu Theme. If the menu style is set to Images & Text, the button will appear as a panel.

BackColor, BackColorAlt, BackStyle, Bevel, BorderColor, and BorderStyle are applied the same as in other examples.

Pressed sets the color values for the Menu button when its selected.

BackColor, BackColorAlt, BackStyle, and ForeColor are applied the same as in other examples.

Size – sets the button's width and height in pixels.

Width – the width of the space that contains the image and the text. This applies to all the modes.

Height – the height of the space that contains the image and the text. This applies to Button and Desktop modes.

ForeColor – the text color of the items listed in the Menu.

Icon Size – sets the icon's size in pixels.

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.

Margins – sets the menu's distance from the form's edge. For example, if the menu style = Images & Text, then the margins form the spacing between each panel.

Left, Top, Right, Bottom – the padding in pixels between the buttons or rows in the menu. This applies to all menu styles except Standard.

MenuStyle – sets the appearance of the menu. The options are: Buttons, Column, Desktop, Image List, and Standard.

Buttons – displays the menu icons as buttons (icon on a button control) with the application name under it.

Desktop – displays the menu item's icon on the desktop of the platform with the menu items name under the icon.

Images + Text– displays each menu item in a panel, where the first column contains the icon of the menu item, and the second column contains the menu item's name.

Text – displays the menu items' name.

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 spacing between the rows when the menu's ListStyle property is set to Standard. This setting does not affect the other menu styles.

ScaleDownText – reduces the size of the caption text by the factor supplied and only applies to the Buttons and Desktop line list styles. The menu item's font size is based on the Environment Fontsize.

ScrollBars – set to None, Horizontal, Vertical or Both to have the menu show or hide the default scrollbar.

BackColor, BackColorAlt, BackStyle, Bevel, BorderColor, and BorderStyle, ForeColor and Transparency are applied the same as in other examples.

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).

Mobile Themes: Panels

Panels are parent controls which are capable of containing child objects and can host multiple layers of objects.

BackColor – the primary background color in a control.

BackColorAlt – the secondary background color in a control.

BackStyle – sets background colors as blended/solid if BackColor and BackColorAlt are used.

Bevel – sets roundness of the corners of a square or rectangular-shaped object such as a button, panel, tab control, or textbox. "0" will no bevel edges; "2" is the average setting for a rounded corner; "4" is typically used on objects in Android.

BorderColor – sets the color of a control's border.

BorderStyle – sets the style as None, Flat, Sunken, or Raised.

Mobile Themes: TabControl

TabControl are parent controls which are capable of containing child objects and as such, can host multiple layers of objects. This control is used to toggle tabbed views of an application(s).

Tab Control

Panels are parent controls which are capable of containing child objects and can host multiple layers of objects.

BackColor – the primary background color for the body of the tab.

BackColorAlt – the secondary background color for the body of the tab.

BackStyle – sets background colors as blended/solid if BackColor and BackColorAlt are used in the body of the tab.

Bevel – sets roundness of the corners of a square or rectangular-shaped object such as a button, panel, tab control, or textbox. "0" will no bevel edges; "2" is the average setting for a rounded corner; "4" is typically used on objects in Android.

BorderColor – sets the color of a control's border.

BorderStyle – sets the style as None, Flat, Sunken, or Raised.

ButtonStyle – sets the style for the tab(s) buttons that do not have focus.

BackColor, BackColorAlt, BackStyle, Bevel, BorderColor, and BorderStyle, and ForeColor are applied the same as in other examples.

HotStyle – sets the style for the tab button that has focus.

BackColor, BackColorAlt, BackStyle, Bevel, BorderColor, and BorderStyle, and ForeColor are applied the same as in other examples.

Mobile Themes: TextBoxes

This sets the theme for TextBoxes. If you decide to use the WaterMark feature, note that the WaterMark styles is affected by the BackColor, BackStyle and ForeColors listed below.

BackColor – the primary background color in the Textbox.

BackColorAlt – the secondary background color in the Textbox.

BackStyle – sets background colors as blended/solid if BackColor and BackColorAlt are used in the textbox.

Bevel – sets roundness of the corners of a square or rectangular-shaped object such as a button, panel, tab control, or textbox. "0" will no bevel edges; "2" is the average setting for a rounded corner; "4" is typically used on objects in Android.

BorderColor – sets the color of a control's border.

BorderStyle – sets the style as None, Flat, Sunken, or Raised.

ForeColor – sets the font coloring in the textbox. Note that if the WaterMark property is used, the ForeColor here will control the color of the WaterMark, but the WaterMark Transparency value will also control the appearance of the text in the TextBox.

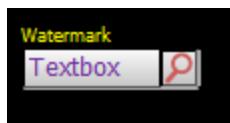
SearchButton

BackColor, BackColorAlt, BackStyle, BorderColor, and BorderStyle, and ForeColor are applied the same as in other examples.

Pressed – sets the theme for the Search button when its selected.

BackColor, BackColorAlt, BackStyle, BorderColor, BorderStyle, and ForeColor are applied the same as in other examples.

Watermarks – sets the theme for the label (i.e. the yellow label which appears outside the textbox) when the Watermark is used. The WaterMark label is different from the TextBox Label.



ForeColor – sets the color of the label for the Watermark when ShowLabel is enabled and the Watermark is enabled.

FontSize – sets the color of the label for the Watermark when ShowLabel is enabled and the Watermark is enabled.

Font Style - sets the style of the label for the Watermark when ShowLabel is enabled and the Watermark is enabled.

Transparency - sets the degree of transparency for the Watermark when the Watermark is enabled. 0 is not transparent, 100 will make the Watermark invisible.

Note: There are no VBA extensions for the WaterMark – all text must be entered in the WaterMark’s Text property.

Mobile Themes: Messages

This sets the theme for Messages (Message box) pop-up window.

BackColor – the primary background color in the body of the Message box.

BackColorAlt – the secondary background color in the body of the Message box.

BackStyle – sets background colors as blended/solid if BackColor and BackColorAlt are used.

Buttons

BackColor, BackColorAlt, BackStyle, and ForeColor are applied the same as in other examples.

Bevel – sets roundness of the corners of a square or rectangular-shaped object such as a button, panel, tab control, textbox, and message. “0” will no bevel edges; “2” is the average setting for a rounded corner; “4” is typically used on objects in Android.

BorderColor – sets the color of a control’s border.

BorderStyle – sets the style as None, Flat, Sunken, or Raised.

ForeColor – sets the font coloring for the button’s caption.

Pressed – sets the theme for the button when its selected.

BackColor, BackColorAlt, BackStyle, and ForeColor are applied the same as in other examples.

Panel

BackColor, BackColorAlt, and BackStyle are applied the same as in other examples.

Mobile Themes: Searches

This sets the default styles for search screens reached from the form.

BackColor – the primary background color in a control.

BackColorAlt – the secondary background color in a control.

BackStyle – sets background colors as blended/solid if BackColor and BackColorAlt are used.

FontSize, FontStyle, and ForeColor are applied the same as in other examples

Heading – sets the theme for Searches.

Align - Aligns the form header according to the selected setting (i.e. Top, Left; Top, Center etc.)

BackColor, BackColorAlt, BackStyle, Bevel, BorderColor, BorderStyle, FontSize, FontStyle and ForeColor are applied the same as in other examples.

Height – sets the heading's height in pixels.

LineColor – the color of the lines within the grid if they are shown.

ListStyle – will hide or show the lines between rows and columns. None hides all the lines; Horizontal displays the lines between rows; Vertical displays the lines between columns; and Both displays lines between rows and columns

RowSelector – sets the coloring and styling of a selected row.

BackColor, BackColorAlt, BackStyle, Bevel, ForeColor and Transparency are applied the same as in other examples.

RowStyle – sets the style for rows that alternate their colors.

AltBackColor – sets the primary background color of a row when UseAltColors is True.

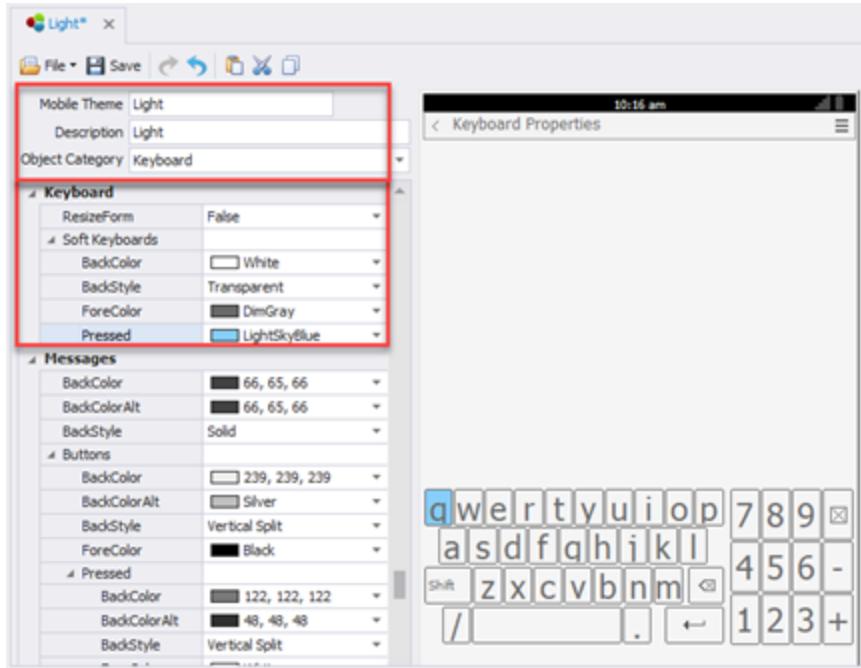
AltForeColor – sets the primary foreground color of a row when UseAltColors is True.

Padding – sets the row height in pixels.

UseAltColors – Enables rows to use alternating colors (True); disables it (False).

ScrollBars – sets the orientation of the scrollbars. Values are: Automatic, None, Horizontal, Vertical, and Both.

Mobile Themes: Soft Keyboard



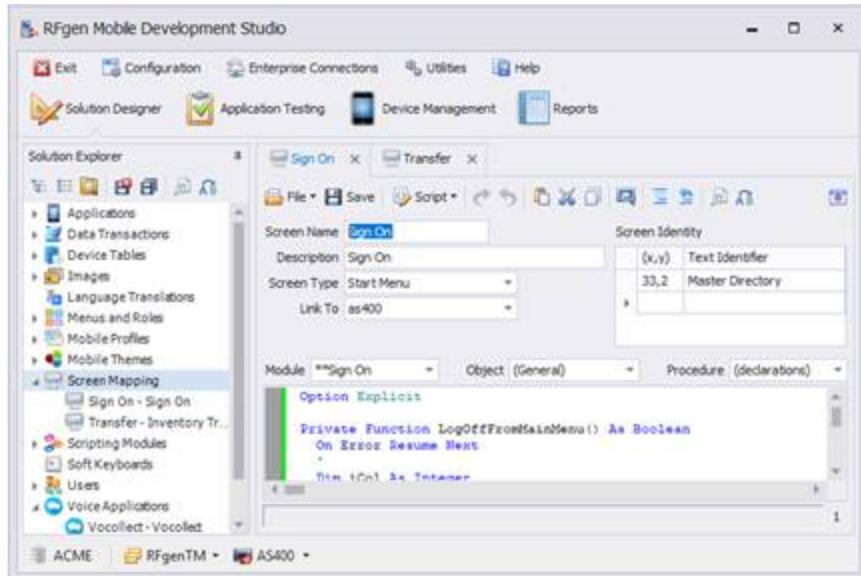
This sets the style for the Soft Keyboards in the Solution Explorer.

ResizeForms – True will cause the form to shrink to a smaller size when the soft keyboard displays on a mobile device. False will allow the soft keyboard to push the form upwards when the keyboard displays on the mobile device.

BackColor, **BackColorAlt**, and **BackStyle** are applied the same as in other examples.

PressColor – Sets the color of a pressed virtual keyboard

Screen Mapping



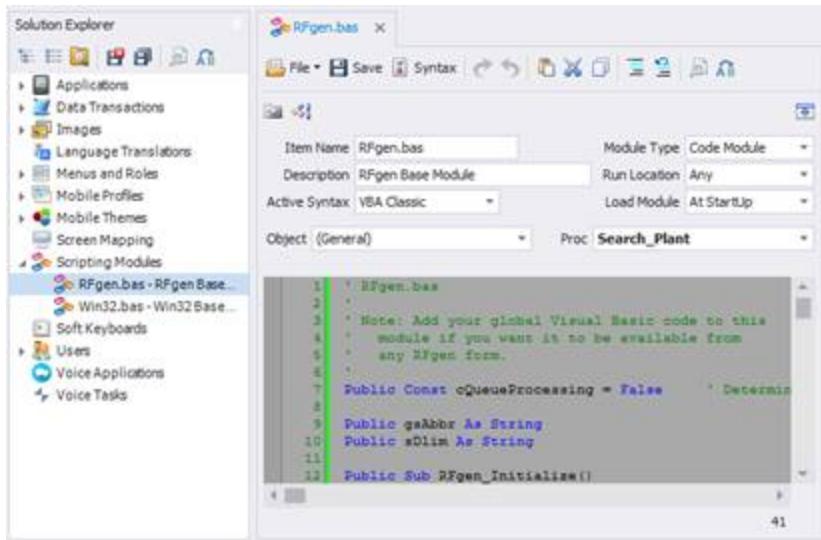
The Screen Mapping module enables mobile applications to be mapped against multiple host systems like the AS/400, IBM Mainframe, UNIX systems and other character-based 'legacy' applications.

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.

For more details on how to record macros, see [How to make Screen Mapping Work.](#)

Scripting Modules

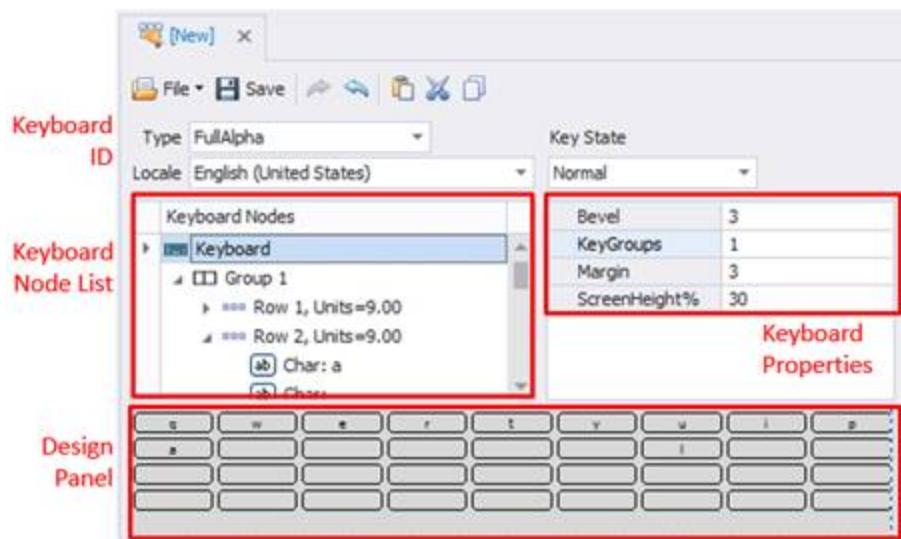


These modules are used to manage global scripts that can be used by any application, timed event, transaction macro, etc.

These **Scripting Modules** files are installed by default:

- RFgen.bas
- Win32.bas
- Both contain global definitions of variables, functions, and procedures that can be referenced by any application or macro script.
- If you want these definitions to be available from any form in your application, add your Visual Basic code one of these base modules and customize it.
- Note: In RFgen version 5.0, "Scripting Modules" was named "VBA Modules".

Soft Keyboards



The Soft Keyboards feature enables you to add a soft keyboard interface to your mobile application for entries that cannot be scanned and may need to be localized (translated) into another language.

Bevel – Shapes the outer edges of keyboard.

Caption – Enables you to enter text for buttons that may have a function other than a character or numeric key value. (i.e. Tab, Shift, Delete.) **Caption** displays if **Key State** = **CapsLock** and (**Style**) = **Character + Text** or **KeyDown + Text**.

KeyGroups. Groups the sections of the keyboard. For example, the left section is alphabetical keys and the right section, numeric keys.

Keyboard ID – Enter a text or numeric identifier.

Key State – Associates character or function of the key based on one of these three states: Normal, CapsLock, and FuncLock

For blank space buttons, set the (Style) to *BlankSpace*.

If the **Key State** is *Normal*, you can assign a button two values: the lower-case letter (Char) and its upper case (CharShift).

If **Key State** is *FuncLock*, you can only assign one value, the Fkey to the button. Leave *Char* and *CarShift* to 0.

Type – Allows presets settings for all alphabetical characters, alpha-numeric characters, or all numeric characters.

Locale – The Local/language that the characters/labels of each key in the keyboard.

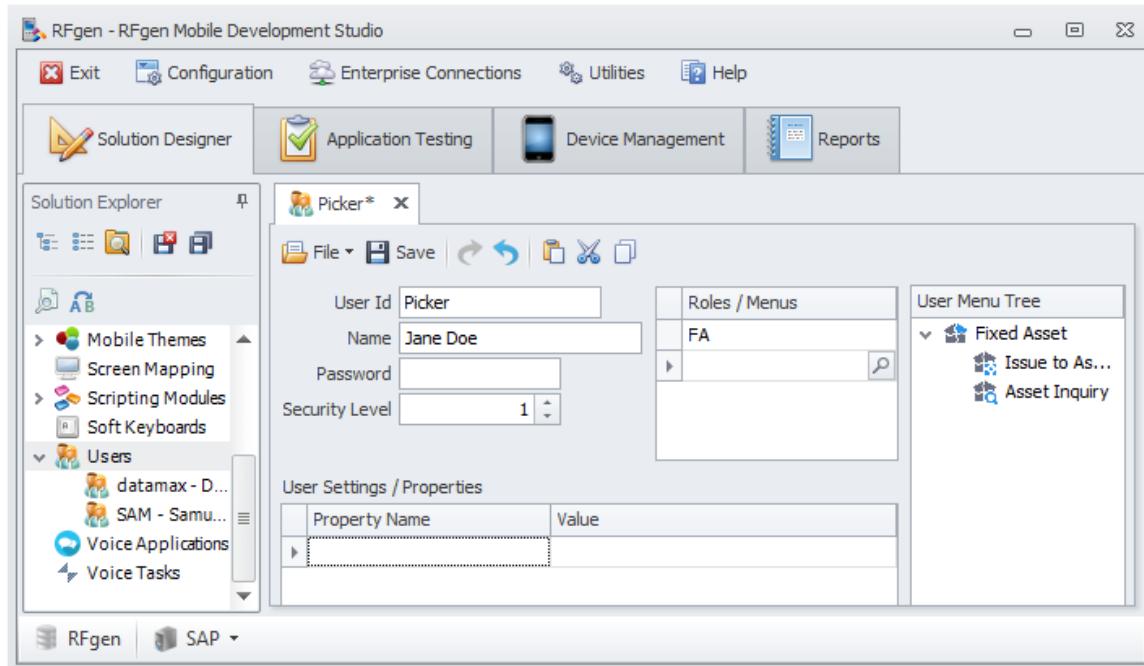
Margin – Sets the spacing between buttons.

RowSpan – Sets the button height based on the number of rows there are above and/or below the button.

ScreenHeights% – Is a percentage of the screen size.

Units – The number of units a row is divided into. (I.e. number of equally sized buttons in a row.) The most granular number of units will be the base set of units for all rows. This way, a button can be allowed to span 3 or more units.

Adding or Removing Users



If you want to assign specific applications (i.e. Inventory Management or Purchase Order Receiving) to specific users (or user roles), you can do this by:

1. Setting up unique Menus in the **Solution Designer > Menus and Roles** tree.
2. Assigning specific applications to each unique Menus and Roles menu.
3. Adding a user to the **Solution Designer > Users tree**.
4. Assigning the user to the unique Menu or Role.

When the user logs in, they are only given access to the menu that was assigned to them.

You can also prevent unauthorized access to your mobile apps, by configuring user identifications and passwords and by setting the security access levels in the **Users** tree.

To add a user

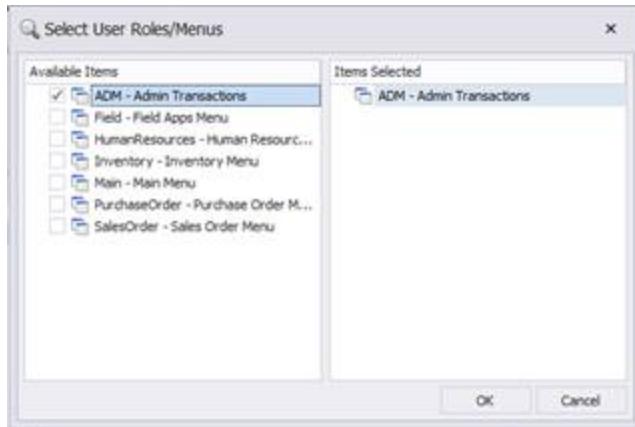
TIP: Create your menus (or roles) and assign applications to the menus or roles before you create your users.

1. Navigate to the **Solution Designer > Users tree**.
2. Right-click on an existing user (or in the blank space) to add a new user, or right-click on the "Users" object and select **Add New User** from the menu.
3. The [New] user tab displays. Enter the user's information.
4. The **User Id** is required, but the **Password** is optional for a user account. SAM's startup menu is 'Main Menu'.

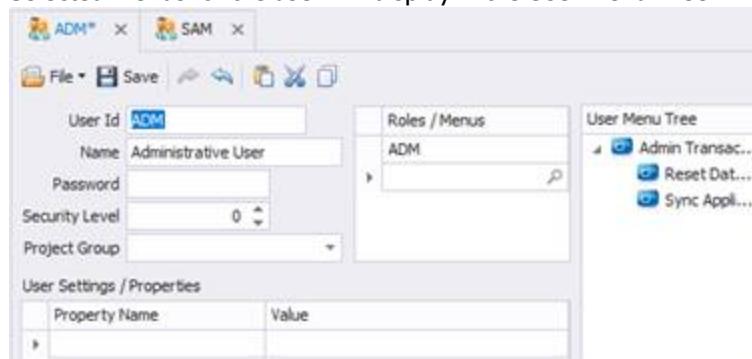
- The **Security Level** is a numeric value between 0 – 100 that will be compared to the menu's required security level before allowing that user access to the following menus or forms.

Assign Menus or Roles to the User

- In the **Roles/Menus** table are menus that come from the Roles/Menus tree. To assign a menu, click on the search icon and check the menu to be added, then click **OK**.



- Selected menus for the user will display in the User Menu Tree.

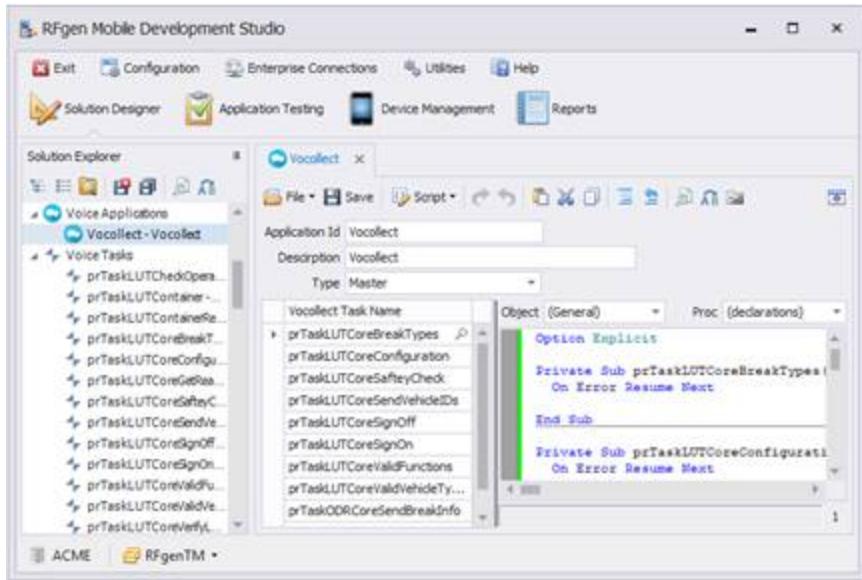


- The **User Settings / Properties** allows the administrator to include any property and value for the selected user for the purposes of retrieving that data at run time. This has no effect on the user logging in.
- When done, click the **Save**.

To Remove a User

You can remove a user by right-clicking on the user and selecting Delete.

Voice Applications



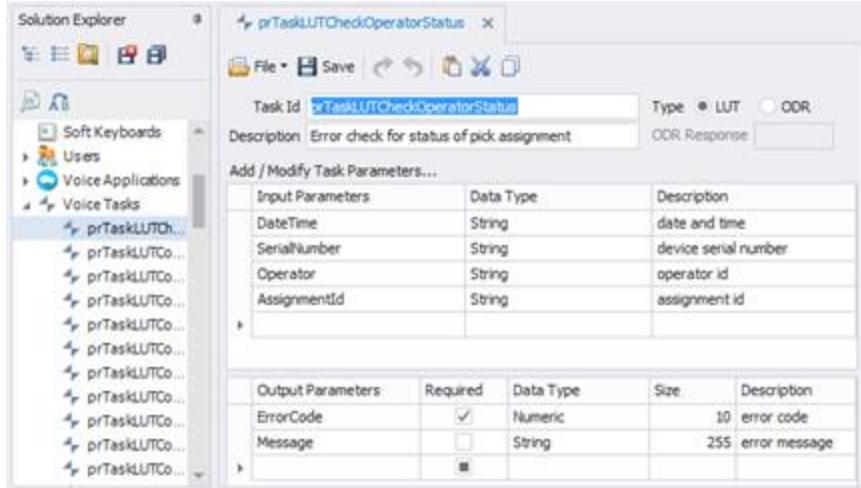
Voice Applications are comprised of one or more voice tasks to be executed when the voice application is installed to a Vocollect device.

To create a new voice application

Before you start, you must first have the desired voice task. See **Voice Tasks** and **To Import Voice Tasks** for details.

1. Right click on the Voice Application icon to add a new application.
2. Enter your **Application ID** and **Description**.
3. Select the **Type**. Master refers to the master database from which the task (script) is resourced.
4. Under Vocollect Task Name, in a blank row, click on the search icon. The Select Tasks list will display.
5. Click on the desired Task(s) to be added then click OK.
6. The selected tasks should display in the Vocollect Task Name field.
7. In the Proc (Process) listbox, select the task prTask to be added to your application.
8. When done, click Save.

Voice Tasks



Vocollect Tasks are pre-defined scripts that execute on Vocollect supported platforms and interact with RFgen to provide the voice solution together with the backend data connection solution.

RFgen provides pre-entered tasks which can be added to your Vocollect Tasks Tree by importing them from the Utilities / Import Mobile Applications menu. See below for details.

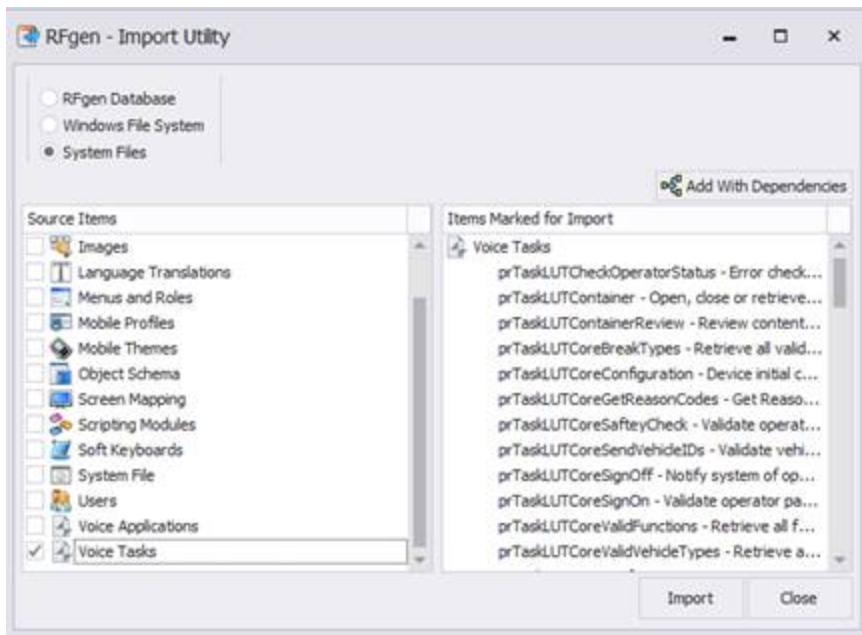
The **Task ID** is RFgen's identifier but in the case of the pre-defined tasks Vocollect provided the names.

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 **Add/Modify Task Parameters** are defined by Vocollect and are configured here.

To Import Voice Tasks



1. Click on **Utilities > Import Mobile Applications**.
2. Click on System Files (at the top).
3. In the Source Items pane, scroll to the bottom and select Voice Tasks.
4. In the Items Marked for Import (right pane) select the tasks to be imported.
5. Click on Import. A message on the number of items imported displays.

Application Tools and Controls

This section describes the components of the Control Properties, Design panel and Tools panel which are used to create the user interface part of an application.

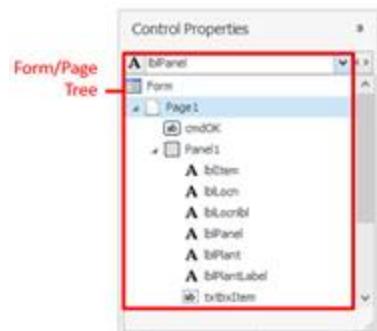
To create a new user interface (form) and application, click on **Solutions Explorer > Applications** > right-click on **Applications** > select **Add New Applications**.

A blank form we'll call the **Design** panel, and **Control Properties** displays.



New applications default to the **Form** and the **Control Properties** panel/tab. The form provides multiple functions. It allows you to visualize the layout of the objects on your form, provide style elements that are consistent across multiple pages of an application, and contains the various controls which are dragged to the form.

Once you enter your **Application ID** and **Description** and click on the **Save** icon, your application to the Application tree.



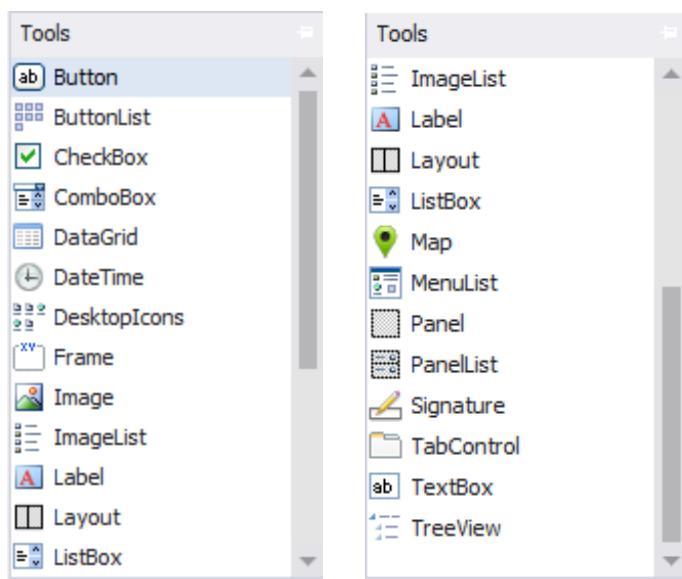
Forms and Pages. All pages by default inherit the controls and the property values (look and feel) from the Form. For example, if you placed your company logo or banner and a message on a form (the owner), all pages would instantly inherit these objects exactly as they were placed and stylized on the form. Note that when viewing a page's owner.

Page and Controls. By default, controls are owned by the page they were dragged to, but can be changed to be owned by the Form. Controls may also be owned by the form.

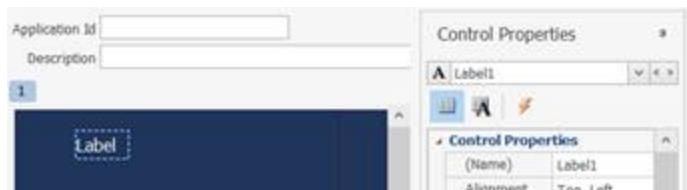
Tip: Regardless of which page you are viewing, always check the who is the object's owner to verify if the changes to a control will be replicated to other pages.

Nested Child Controls. Some controls have the ability to host other controls so that you have nested child controls. For example, if a Textbox object called txtInventory can have Panel2 as its parent, Panel2 can have Panel1 as its parent, and Panel 1 can have Page1 as its parent.

To access a control, also called "Prompts", click on the **Tools** Tab. Note there are many controls/prompts in this list. Some are graphical and will display, others will not show up until runtime.



You select and drag a control onto the Form to create your user interface. For example, if you dragged the Label property, the following would display.



Prompts/Controls Overview

This section describes the features and value of the various objects in the **Tools** tab.

TIPS for Placing and Sizing Objects on a Form

Before you start, plan which objects you will need in your data collection application, what the workflow prompt order will be, how many pages you'll need. Create a diagram of the various graphics/boxes so you can get an idea of how you want them laid out.

When you are ready, create the app and set the number of pages to be used on the Forms control. Drag-and-drop the desired prompts from Tools into the display area (the page of the form) in the order in which prompts will appear.

You can also double-click on the objects and they will transfer automatically to the display). Note that a data field and its associated Caption (prompt) or label are addressable separately.

The upper left-hand corner of the form/display area is "0,0". The location (Left, Top) of the prompt is relative to the left-hand corner of the form. 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. The Move icon  displays.



Once the control is selected as indicated by the Move icon, you can use the arrow keys or your mouse to move the control to a new location. If working with nested controls, grabbing the outside edge of the control will select the parent and child controls.

To resize a control first select it and grab the edge of the control and drag in a direction. Holding down the Shift key and clicking on the arrow keys will move the prompt 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.

The Button Control



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 Control



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 Control



Final Confirmation?

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 Control

Reason Code

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 DateTime Control



The DateTime control displays the time in 12-hour format. The time is based on the device supplying the application. For example, if the device is in thin mode, the time is supplied from the system that is hosting the application on the device. If the device is running in Batch mode, the time is based on the device itself.

The prompt Format property defaults to "hh:mm TT". For details on supported formats to display the date and time in other formats, refer to *VBA Language Extension – Format*.

The DataGrid Control

Grid Example						
Order: 521						
Line	Item ID	Locn ID	Batch ID	Qty	Customer	Ship Date
1	714409	0002	15962	40	SARA	6/11/2013
2	100620	0004	98745	30	3M	6/9/2013
3	733483	0003	54278	10	3M	6/9/2013
4	896470	0001		1	3M	6/9/2013

The DataGrid object supports binding large sets of data and displaying it in columns and rows so that users can view, select and edit the data.

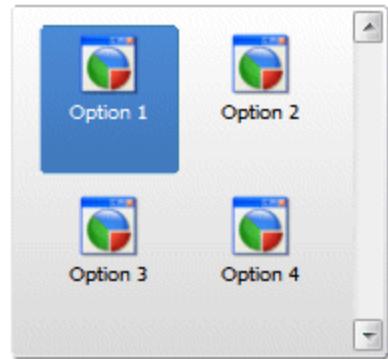
Data edits (adding, modifying and deleting text in a cell) are supported in the DataGridView -- scripting for data edits is not required.

To add, populate, and manage the contents of the DataGridView, use VBA scripts.

Refer to *VBA Language Extensions, Prompt-Specific Extensions* such as **List** for details.

The DesktopIcons Control

Select an Option

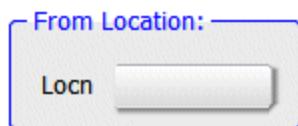


The DesktopIcons object is a control used to display a list of options in a tiled icon format.

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.

The Frame Control



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 Control

This control was removed in version 5.1 of RFgen.

The Image Control



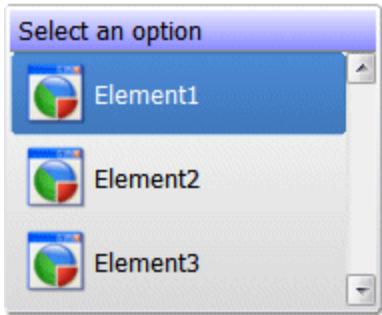
The Image object is used to display a picture on the application. This object supports a large variety of image formats in a Thin Client environment. They are BMP, DIB, GIF, JPG, WMF, EMF and ICO. This object can only be seen in the graphical version of the client. When using the Windows CE / Mobile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

To add an Image, browse for the desired image using the **Image** property and select the file from the **Solution Explorer > Images** tree. 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 Control



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 Control

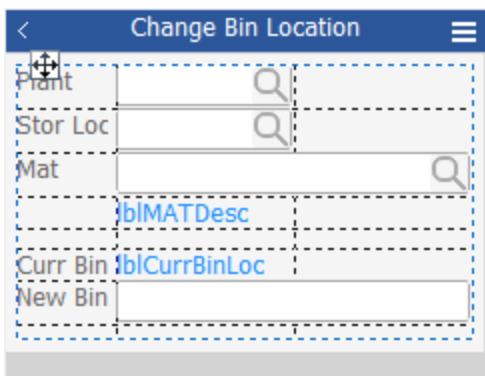


This object is used to add additional text to the screen not necessarily associated with one prompt. All of the default objects come with their own label. For example, if the customer desired the text "F4=Exit" at the bottom of the screen, a label would be used. If a prompt's label required 2 lines, 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 Layout Control



The **Layout** object is a parent control that helps keep child objects (i.e. labels, textboxes etc.) aligned when a prompt changes its size at run time. It uses columns and rows to maintain layout alignment. Each column and row can expand or shrink to accommodate the contents of a cell within the parameters of the Layout control. For example, if your application was translated from English into German and the text expanded 25%, the Layout control would automatically adjust the text and its adjacent controls' position so they remain aligned to each other.

This auto-alignment feature saves development time creating separate layouts of an application that will be localized into different languages. It also saves the developer's time positioning individual controls when the application is used on devices with small or large display areas.

How the Layout Control Works

The easiest way to design for automated alignment and resizing is through the graphical control for prompts.

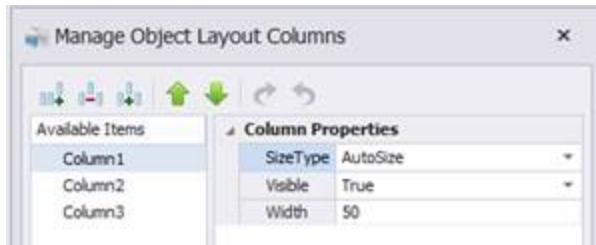
In a Layout control, RFgen adjusts the widths of columns and heights of rows using the space available inside the control. RFgen allocates the available space using these factors:

- The number of columns and rows present
- The contents of each cell
- The property (or methods) assigned to each column or row

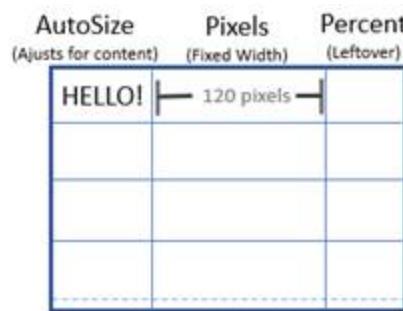
When assigning values, note that the Layout control's overall width and height cannot be made larger or smaller by the Column and/or Row's width/height values.

See "How to use the Layout control," for more information.

Layout Column Properties



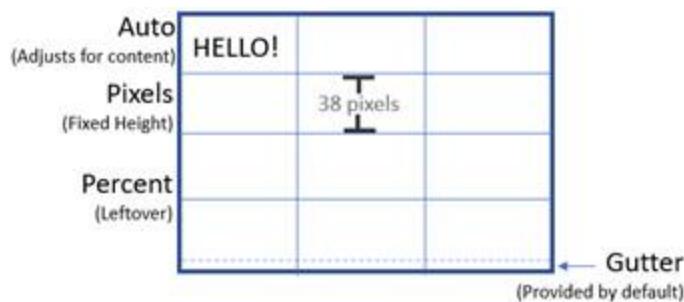
RFgen adjusts the widths using the Layout Column **SizeType** and **Width** properties.



- *AutoSize* – Column width is based on the largest item (Label, image etc) inside the column; the Width value is ignored. If the column is empty, the column will collapse (i.e. become 1 pixel wide)
- *Pixels* – Uses the Width value to set the column width.
- *Percent* – Uses the space left over from other columns or uses the Width value to calculate the allocation as a percentage of the Layout control. If you have columns where each column has at least one object (i.e. Label, Caption, button, TextBox etc), and want to use AutoSize, its best to have at least one column set to Percent – preferably one that contains TextBoxes.

Note: Layout Controls do not have TabNos as they are passive "containers."

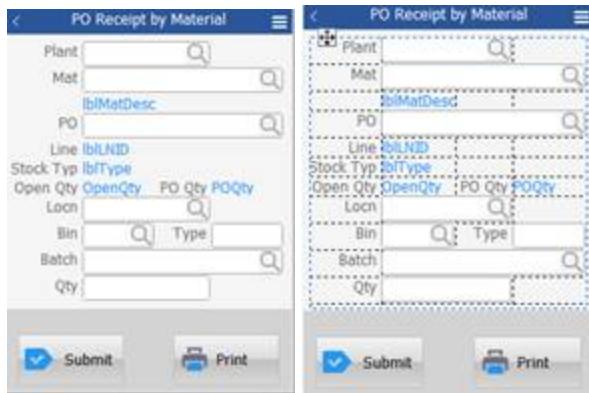
Layout Row Properties



- *AutoSize* – Row height is based on the largest item (Label, image etc) inside the column; the Width value is ignored.
- *Pixels* – Uses the Height value to set the column width.
- *Percent* – *Uses the space left over from other rows or uses the Height value to calculate the allocation as a percentage of the Layout control.*

Note: RFgen automatically places an extra row at the bottom of the Layout control in order to "soak" up leftover space that not used up by the other rows. While the property on this bottom row can be changed or deleted, we recommend you leave its SizeType as Percent.

How to Use the Layout Control

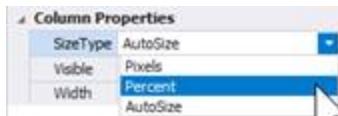
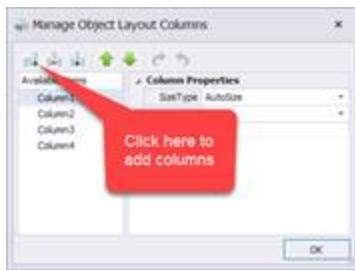


Preparation

Use a Panel control to contain the Layout control. This makes it easier to apply themes to all the objects in your panel. Note that there are no Themes for the Frame property.

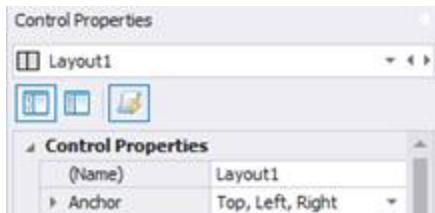
To Add Your Controls

1. Create your application form. For example, make your form 240 pixels wide and 320 pixels high. If you like having extra space for moving objects around before adding them, consider making the size larger then resetting it to your final size when done.
2. Drag the Panel onto the form, and make it the same width as your form but $\frac{3}{4}$ of the Form's height. Set your Panel's anchor (Top, Left) and location (i.e. 10, 10). Note: The purpose of the panel is that it allows all child objects such as the Layout control and its objects to inherit common property values from the Panel's Theme. You can add the Panel now, or add it after you've created your Layout control, then drag the Layout control onto the Panel.
3. Drag your Layout control onto the Panel.
4. Add the required columns and rows and set their SizeTypes to Percent. For this example, add 5 columns and 12 rows, set the SizeType to Percent, leave Visible to True, and ignore the Width.



TIP: We added an extra column and row because it's a good practice to have extras in case you need it. Its easier to delete a column and row rather than add it after everything else is set up.

5. Set the height of your Layout (grid) control
6. You now have a squishy grid. Calculate your desired overall grid height by multiplying the estimated row height by the number of rows. For example, our app requires 11 rows. (The 12th is an extra row.) We want each row to be about 22 pixels high. Therefore: $11 \times 22 = 222$ pixels. We entered 223.
7. Set the Layout (grid) Size Height to 222 pixels.



8. Anchor the Panel and Layout to use Left, Top and Right.
9. Decide if any columns need to serve as "spacers". Drag all the prompts into their cells.
10. Set your Layout Column properties.
11. Since the first column has Labels, set the Column SizeType to **AutoSize** on columns that contain Labels and captions.



For example:

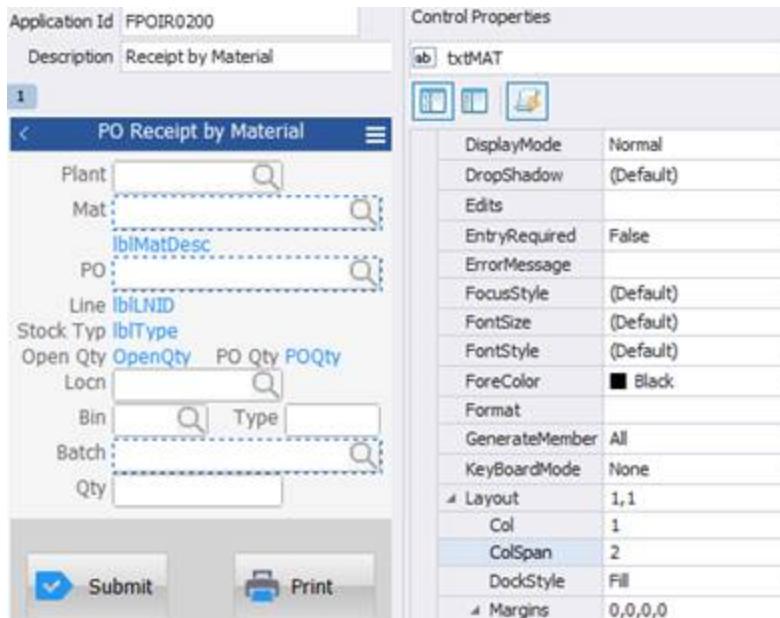
lblPlant > Alignment = Top, Right

lblPlant > Layout > DockStyle = Fill

lblPlant > Layout > Margins: Right = 3

12. To make your TextBoxes span across columns, select the TextBox and set the ColSpan value to the number of columns you want it to span. If you have TextBoxes that will all span the same number of columns you can use Control + Click to select them and then set their values. Remember to set the prompt's DockStyle to Fill.
13. Finalize Your Layouts
14. Stretch the Layout control to the size of the Panel and make sure both the Panel and Layout controls are anchored to Top, Left and Right.

TIP: Anchoring the Layout control to the Bottom will cause it to grow taller if the form changes. If you want it to remain a static height, do not anchor to the Bottom.



For example:

txtMat > Layout > ColSpan = 2

txtMat > Layout > DockStyle = Fill

txtMat > Layout > Margins = 3

If needed, align your labels so they won't overlap your TextBoxes by changing the label's Alignment properties.

TIP: If you have any columns that are empty, set the Layout control Column's SizeType to Percent; if its empty and is set to AutoSize, it will collapse (1 pixel wide). This way, RFgen will automatically allocate remaining space to the empty column and make it easier to add any other prompts to that space.

15. Finish positioning the Layout with its location and anchoring if this has not been done already.

Using the Layout Control for Buttons

For Layout controls containing a button, you can setup the Layout Column properties to automatically resize the Width of a Column, but not the Row Height. To accommodate a button with a multi-line caption, first set the button's height, then set the Layout Control Row Height to a fixed height similar to the Button's height value.

The ListBox Control

Entre	Prep	Weight
<input type="checkbox"/> Roast Beef	rare	2oz
<input type="checkbox"/> Salmon	smoked	5oz
<input type="checkbox"/> Chicken	BBQ	5oz

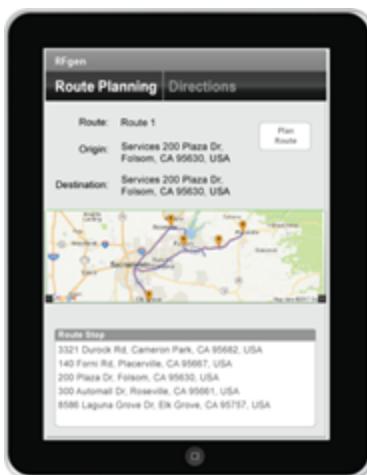
The Listbox object is an application prompt that allows items to be displayed in an area of the application. Items may be sorted and/or selected as required by the needs of the application. A Listbox can have multiple columns, selection styles (spinners, checkboxes, or none).

To use a ListBox, select the desired layout and selection styles from the Columns property and use VBA script to populate and manage items displayed in the box. The ListBox prompt can contain the values itself by entering them in the 'ListData' property and separating each column using the pipe character (|). Or, you can enter data using scripts.

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.

Note: If you would like to convert the Listbox to a PanelList, you can right-click on the object and select "Convert to PanelList." This action will literally convert your ListBox control into a PanelList Control.

The Map Control

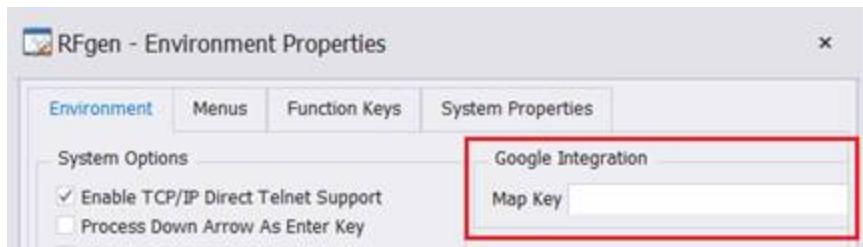


The Map object is a control that provides a subset of the features available from Google maps and global positioning station (GPS) APIs. You can use it to create applications that:

- Calculate the most efficient route from a set of addresses or GPS coordinates
- Display a map of a route plan
- Convert GPS coordinates into an address or converts an address to GPS coordinates
- To use the Map object requires VBA scripting, an API key provided by Google maps, and registration of your app and API with Google API Console. To learn more about obtaining an activation license key, go to the following URL and click on "Paid".

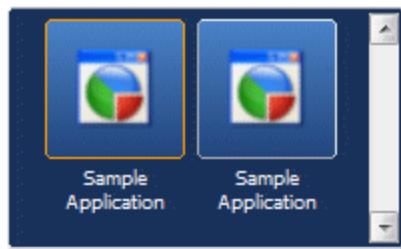
<https://developers.google.com/maps/pricing-and-plans/>

Once you have obtained your activation license, enter it under **Configuration > Environment Properties > Google Integration Map Key**



Note: The Map object cannot hold focus and therefore does not have a TabNo (tab number).

The MenuList Control



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 Control

This control was removed in version 5.1 of RFgen.

The Panel Control



The Panel Object is a parent control that enables you to easily perform group actions such as deleting, hiding or moving all child controls at once by deleting, hiding or moving the parent Panel control. It also makes it easier to apply common grouping values such as the background.

When placing a child object on the panel, make sure its owner is listed as the **Panel**.

Note: Since the Panel is a container, it does not hold Focus and therefore does not have a **TabNo** (tab

number).

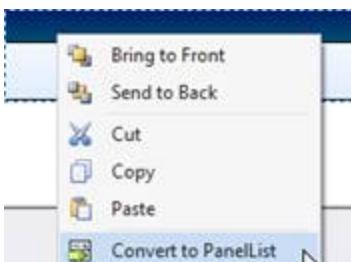
The PanelList Object



The PanelList control creates rows of panels much like a ListBox except each panel may contain a variety of controls arranged in any order on a panel and, if the control is data-centric, it can be bound to a column. Multiple objects bound to columns on a panel form a set of columns (called a ColSet) that can be populated with data.

Using the PanelList Control

Simply set the number of panels you will need, arrange your controls and information on each panel that has a unique layout, bind your data-centric controls to specific columns, and script your data source. At runtime, RFgen replicates the panels populated by your data. You can accommodate the volume of objects by changing your panel's height in a PanelList. For more details on how to use the PanelList Control refer to to Technical Bulletin



You can also convert a **ListBox** to a **PanelList** by right-clicking on the PanelList object and selecting the **Convert to PanelList** function. This action will allow you to reuse your ListBox object as a PanelList, and enable you to rearrange your ListBox column headers as Labels on the panel. Since the PanelList is a container for child objects, you can add TextBoxes to display your contents for each Label. To display the contents sourced from your VBA script, use the TextBox **BindToColumn** value. The **BindToColumn** property is only enabled for the CheckBox, ComboBox, Image, Labels and TextBoxes on the PanelList control.

Scrollbar Control Property

In the design view, the Scrollbar property is available to graphical controls where content can be expanded horizontally or vertically. For example this property is available in the ComboBox, DataGrid, Desktop Icons, Image List, ListBox, MenuList, Panel, and PanelList.

The properties for Scrollbars are:

(Default) - The scrollbar's appearance such as background coloring and line thickness is set by the properties under Mobile Themes > Scrollbars. The type (Horizontal or Vertical, automatic, both or neither) is obtained from the control under Mobile Themes > [control name] > Scrollbars.

Automatic - A vertical and/or horizontal scrollbar will display once the control exceeds the limits of the screen size.

None - Scrollbar display is disabled.

Horizontal - The horizontal scrollbar is enabled.

Vertical - The horizontal Vertical is enabled.

Both - The horizontal and vertical scrollbars are enabled.

If you would like all your scrollbars to have a similar appearance, you can set the characteristics of the scrollbar appearance via Mobile Themes > [Your theme name] > Scrollbars.

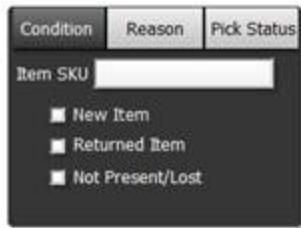
The Signature Control



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.

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 TabControl Control

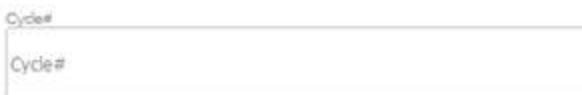


The **TabControl** enables users to toggle between pages or prompt groups at runtime. No coding is required to enable switching between tabs. To use this object: 1) Drag the control to the page;

- 2) Set the number of required tabs in Control Properties > Size > Tab Count; 3) Enter the captions under TabInfo > Caption; and 4) Drag the appropriate control to each tab. Note that the total width of *each* tab is the total width of the TabControl divided by the number of tabs. At run time, the control will determine which tab was selected and switch to the appropriate view.

Note: The **TabControl** cannot hold focus and therefore cannot have a **TabNo** (Tab Number).

The TextBox Control



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. The WaterMark property allows you to prepopulate the textbox with data and make it appear semi-transparent so that it looks like a water mark. You can also force the watermark text to appear as a label for the textbox after the user has entered his/her data in the text box. (See the TextBox WaterMark property for more details.)

You can also use the data display area to print (translate) data from your database using the '@T' default Property. (See the Data Entry Default Properties section). Translated data is typically sourced from your database of localized strings; For example, 'CA' is entered at a prompt, 'California' is retrieved from your database and displayed in the unlinked textbox field.

You can also display the search icon in the textbox to enable searches of data the textbox is linked to. To enable the search icon, in the application's Script tab, select the OnSearch event from Procedures.



The TreeView Control



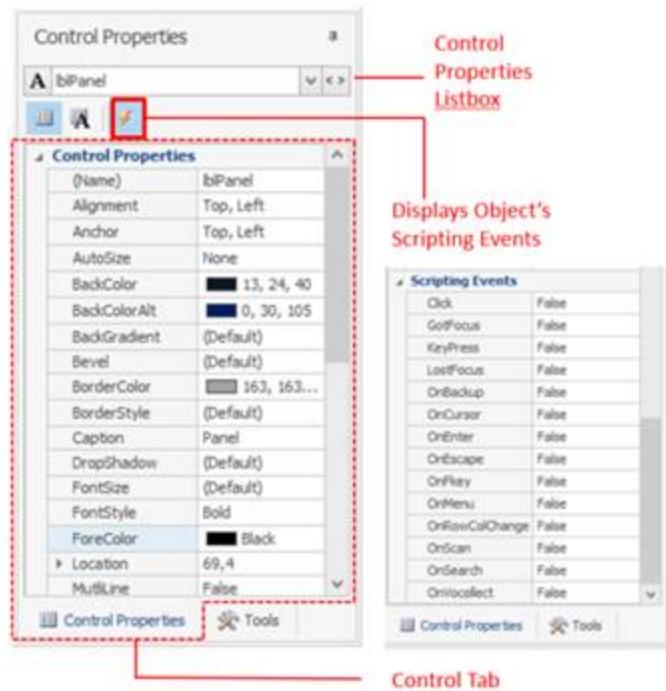
The TreeView object enables developers to present data in an outline format where information is sorted hierarchically. Each group (also called a parent) may contain child items and parent "containers" can be expanded or collapsed for ease of navigation to other parent items.

This object is very similar to the ListBox object except that the Tree supports multi-level, indented lists that can be expanded or collapsed at run time. For example to add data and manage the contents of the Tree, use VBA prompt extensions such as List.AddItem.

Control 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. Note that not all control properties have a label.

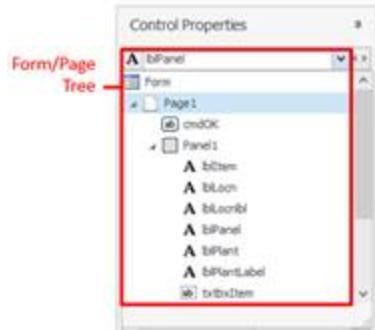
An object's properties are modified by setting the property values for that control via the **Properties Control** panel, and/or by scripting the methods that set the properties such as how the control will be populated, positioned at run time, stylized, formatted, sized etc.



Note that **Scripting Events** in the Control Properties panel remains hidden unless you click on the Thunderbolt icon. Each object has "Scripting Events" which will be discussed later.

It's also important to know that objects/controls have relationships to other objects in the form. For example, there are parent-child relationships between forms and pages, and between controls and other controls. This enables you to change or move multiple objects instantly through a parent "Owner" object.

These relationships are viewed when you click on the **Control Properties Listbox**. Below shows an example of the relationship between a form and its pages, and the child controls belonging to specific pages.



Forms and Pages. All pages by default inherit the controls and the property values (look and feel) from the Form. For example, if you placed your company logo or banner and a message on a form (the owner), all pages would instantly inherit these objects exactly as they were placed and stylized on the form. Note that when viewing a page's owner

Page and Controls. By default, controls are owned by the page they were dragged to. The parent can be

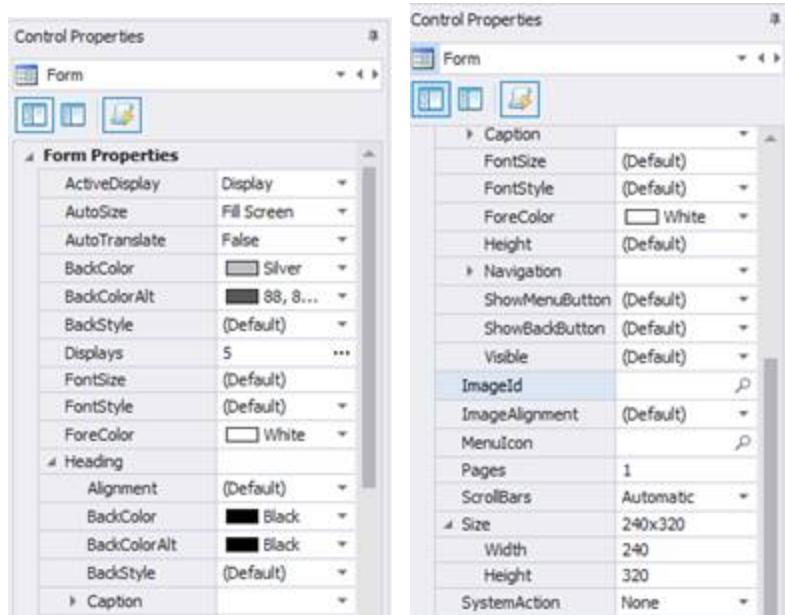
changed by dragging the object to the form header and then moving it back to the target container.
Tip: Regardless of which page you are viewing, always check the who is the object's owner to verify if the changes to a control will be replicated to other pages.

Nested Child Controls. Some controls have the ability to host other controls so that you have nested child controls. For example, if a Textbox object called txtInventory can have Panel2 as its parent, Panel2 can have Panel1 as its parent, and Panel 1 can have Page1 as its parent.

Form Control

The Form acts like a template to other application pages, its properties are both global and unique, and is also referred to as "Form Group Control".

The form controls how the screen is: sized, navigated, interacted with (i.e. lock or allow swipe), and localized. The screen's sizing, automatic translation, number of displays (display types), and look and feel are only controlled by the properties in the form. The form's heading, display of the menu icon = and\or back button < and the menu's appearance and interactive behaviors are controlled in Mobile Themes and by the properties in the Form.



Form Properties

ActiveDisplay lists which of the Display is used if more than one display format was created in the **Display** property. For example, if the application needs to be localized, you can create a unique display for each language, or for other purposes. The default is "(Default)" from the Display property.

AutoSize will stretch the form's background based on the options of None, Contents, Horizontal, Vertical and Fill Screen. How the screen appears in the device's display will also depend on Form and Menu properties. For example, if AutoSize = None, and the Form size is taller than an Android's screen, then the application screen could be scrolled (swiped up or down) to see more of the screen. But if the AutoSize = Fill Screen, RFgen

shrinks the screen so there is no additional extra screen to scroll (swipe up or down).

AutoTranslate will display localized strings if the localization values are setup properly in the Form Display, in the Configuration > Application Preferences and in the source of the translated string ID.

BackColor and **BackColorAlt** and **BackStyle** create either solid or gradient backgrounds depending on the option chosen in the BackStyle property.

The **Displays** property group allows you to save one or more sets of property values, such as the device display size, display mode and specify the Locale if you want to create localized apps. The display values are sourced from the **Solution Explorer > Target Devices node**.

The **FontSize** and **FontStyle** 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.

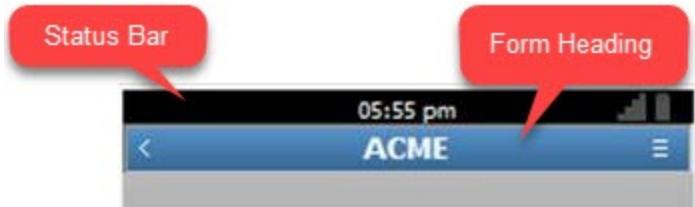
The **Heading** property changes the appearance of the heading at the top of the form if Heading: Visible =True. The Heading subproperties are: Alignment (of content inside header), BackColor, BackColorAlt, BackStyle, Caption, FontSize, FontStyle, ForeColor, Height, Navigation (label for Navigation icon), ShowMenuButton, ShowBackButton, and Visible.

The **Heading ShowBackButton** is used for the carrot < button.

The **Heading ShowMenuButton** is used to display the system menu =which launches the system menu icon.

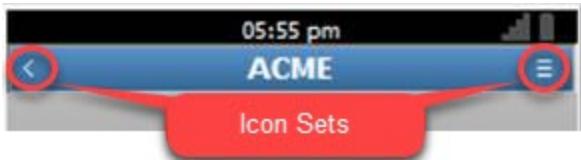
If the value is set to True, the button is visible on the form header. It returns the user to the previous screen. If set to False, the button will not display. The (Default) value inherits the values provided in the **Mobile Themes > Form Heading**.

The **Visible** property hides or displays the form's heading. True will display the heading; False hides it. (Default) will use the values from **Mobile Themes > Form Heading**.



Note: The height of the black status bar that appears above the Form Heading is set by the Target Device. If set to "True", the Status Bar will display the device's time, signal strength and battery strength. For details on the StatusBar property, see **Mobile Themes > Environment: ShowStatusBar**.

The **Icons** property allows you to upload and add icons to the form header (not the form Status Bar). Use Icon Sets to add an icon, set its location in the form header, and associate an event with the icon when its tapped by the user.



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 **Pages** property is an editable field that sets the number of pages associated with the application. Any newly added page will inherit the control value from the parent. For example if you change the Form Page Property value from 1 to 3, and two more pages are added to the Page control and form after you press Enter. See the Page# control for more details on page properties.

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

The **Size** property sets the height and width in pixels for the form.

The **SystemAction** property sets how the application is launched – from the login or another menu.

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.



icon will launch 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

Page Control Properties

You set unique properties for each page through the Page Control. The values set here will override values set in the Form or from Mobile Themes. But if the value is set to "(Default)" then its values are inherited from the Form (if the Form's corresponding property value is set) or from Mobile Themes, (if the Form's corresponding property value is set to (Default)).

Page Control Properties

Anchor sets the coordinates of where a page is docked/anchored for screen sizing purposes.

BackColor and **BackColorAlt** are used to create either solid backgrounds or gradients depending on the option chosen in the **BackStyle** property. If these are set to (Default) they use the values from Mobile Themes.> Environment.

FontSize and **FontStyle** properties are used to set the font size and style as specified. If (Default) is set, the

values come from Mobile Themes.> Environment.

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

GenerateMember helps improve an application's performance by telling RFgen what it should or shouldn't generate. The values are: *None*, *Member Only*, *Event Only* and *All* (which is Member + Event).

None will NOT generate a member variable; *Member Only* generates a member variables with no events; *Events Only* generates object events only, but no member variables; and *All* generates both member variables and object events. The default for prompts like the Textbox and Labels is "Member Only," whereas prompts like Buttons, DataGrid, CheckBoxes, ComboBoxes, Maps, MenuList and Signature will default to "All."

Heading can be used to enter your header (text string) or link to a text string from the drop down list if the list exists.

Location sets the location of the page (number of pixels to the left of an anchor and number of pixels from the top.)

Navigation is not used.

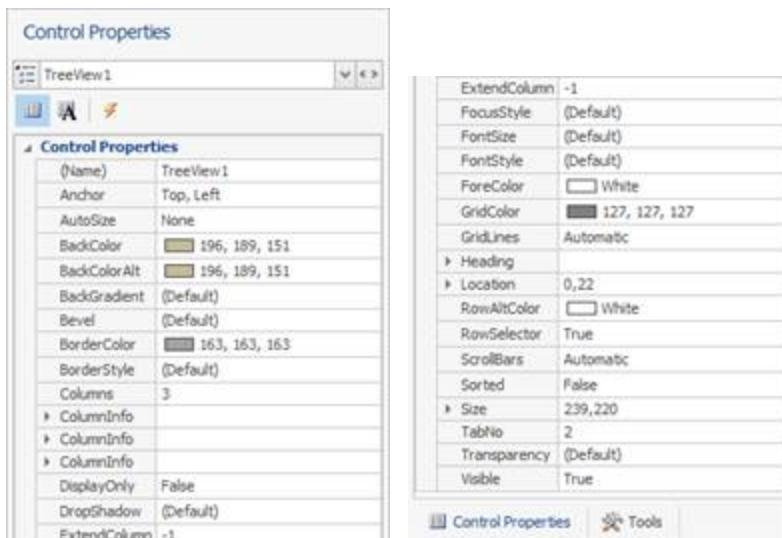
PageNo is an inherited value from the Form Page property.

Type sets how the page is used by the application. The options are Standard and Resource. Standard (the default) contains the graphical controls that display in the mobile application. A Resource page contains other objects or controls referenced by a Standard page and is typically not designed for user interaction.

Page Advance controls the flow of the prompt based on the option selected.

Graphical Control Properties

The Control Properties tab contains the properties specific for the object/control selected in the design area.



Not all properties appear for all control types. Image controls, check box controls and others will have unique properties. This example uses the **TreeView** which has more properties than most controls.

If the value of the property is set to "Default" then it will use the settings inherited from the Mobile Themes.

The **(Name)** is the internal name used to identify the control/prompt in the script.

Tip: As a best practice, follow 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 has been removed in version 5.1 of RFgen. See Parent-Child relationships.

The **Prompt** property has been removed in version 5.1 of RFgen and replaced by **TabNo**.

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. If you anchor a control to the right or bottom, the anchor property in the designer will become a group and the right or bottom values (depending on active anchors) will display in the Control Property Anchor settings in pixels. For example, if the Alignment is "Bottom, Right", and the Anchor is "Right" and its 10 pixels from the right, then the value of "Right" "10" displays in as a child property in the Control Panel. If you change the size of the Form, the prompt anchored 10 pixels from the right remains the same.

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 **BackColor** and **BackColorAlt** (previously called **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 color can be set from using Custom Color tab or by a 6-character hex value (which gives you 16 million colors to choose from).

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 **Bevel** (graphical mode only) property sets the curvature of a square's and rectangle's corner edge.

The **BindToColumn** is only used with data-centric controls on Panel(s) in the PanelList control. Its used to bind a data-centric controls such as a Textbox or label to a specific column. For example, if your first column is 1, then the **BindToColumn** value should be "1". The list values can be ordered to start with 0 or 1; You can force the ordered list to start with "1" by checking the box under **Configuration > Environment Properties > Environment > List Items Collection is One Based**.

The **BorderColor** (graphical mode only) property controls sets the color of an object's boarder. The (Default) is Flat. Options are: None, Flat, Sunken, and Raised. Its only functional if Visible and Transparent are True.

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 **ButtonSize** property was removed in version 5.1 of RFgen. See the **Size** property.

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

The **Caption** property (graphical mode only) 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 **Columns** property group has two functions: a) It lists how many columns there are in the control, and b) provides a graphical interface for creating and saving column themes called **ColSets**. The Columns property group is only functional for the ListBox, DataGrid and TreeView controls. (It appears in the ComboBox and Layout controls but is not fully functional for these controls.)

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 **DisplayMode** (previously called DisplayOnly) property changes how the prompt is used. It has three modes: Normal, ReadOnly, or Disabled. *Normal* allows the end user to enter data into the prompt. *ReadOnly* makes the prompt into a display-only prompt where no data can be entered. Disable will make the text appear as if its ghosted (grey'ed out or disabled). For example, if the prompt's forecolor is set to (default), and in Mobile Themes, the forecolor for the prompt is set to black, then the font color should be a diluted version of that color so it appears grey or ghosted. The user should also NOT be able to enter data at runtime too.

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.

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 **FocusStyle** property specifies how an object shows it is the active/selected object at run time. The default is Stannard. Other options are: Active Border or Visible w/Focus.

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 **GenerateMember** property helps improve an application's performance by telling RFgen what it should or shouldn't generate. The values are: *None*, *Member Only*, *Event Only* and *All* (which is Member + Event).

None will NOT generate a member variable; *Member Only* generates a member variables with no events; *Events Only* generates object events only, but no member variables; and *All* generates both member variables and object events. The default for prompts like the Textbox and Labels is "Member Only," whereas prompts like Buttons, DataGridView, CheckBoxes, ComboBoxes, Maps, MenuList and Signature will default to "All."

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 **Heading** property group consists of additional properties that are used to stylize the header of a control.

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 is now called "TextOptions". For details, see "TextOptions."

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 **RowAltColor** and **RowSelector** (TreeView Control only) properties sets every other row to the color selected and enables users to select the row (True) as opposed to just viewing the content in each row.

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. [For more details click here.](#)

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 **Sorted** (DataGridView and TreeView only) enables the ability to sort contents in the first column of a DataGridView or TreeView control.

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 **TabNo** (previously called a prompt) is the sequence value assigned a prompt so that the tabbing order of a control is trackable in an application that has many controls. This can also be used for troubleshooting purposes.

The **TextOptions** property has three states for specifying how the prompt should display text: SingleLine, MultiLine, or Vertical. These features also help the text stay aligned when used with AutoSize mode.

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

The **Transparency** property sets whether a graphical object's background color is transparent. It enables the objects parent (i.e. Page1) color to show through. "Default" uses the color selected in **BackGroundColor** and **BackColorAlt** properties. To force an object to be transparent, select "Transparent" from the **Back-
GroundColor** and **BackColorAlt** properties.

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 WaterMark property is a property under the TextBox control. The WaterMark's Style property enables a caption to appear inside or outside a Textbox, or, display inside the TextBox if the value is blank, then outside if the value is entered into the textbox. WaterMark captions can be aligned and localized and its font color, size and style changed using the FontColor, FontSize, and FontStyle properties. This feature is only available in graphical form.

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.

Note:

The **Displays** tab was removed in version 5.1 of RFgen.

"Display" settings are now located under the **Form** property.

The **Design Mode** (which used to be under Display tab) has to **Configuration > Desktop Preferences** in version 5.1.

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 ';'0' (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 ';'0' expression; i.e., AllowChange = True, is the same as placing a ';'0' 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(;0)

1000;0 means default equals '1000', allow '0'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(;0)

@DATE;0 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(;0)

@TIME(;0) 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)(;0)

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

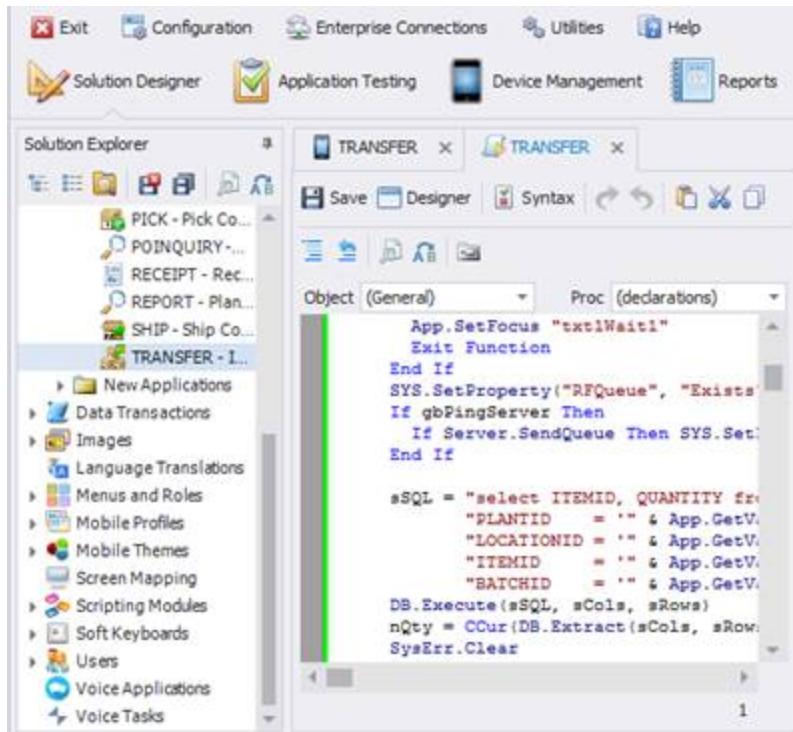
@STRIP,S,str to strip data suffix 'str' if it occurs

@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.

Scripting Your Application

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



Menu Bar

When the script window is opened, the menu bar at the top of the window provides some standard and 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** and **Replace** icons allow the user to find or replace text within the current application or all applications and macros at the same time.

The **References** icon allows non-globally loaded **Scripting Modules** to be associated to the application. This requires the setup of the specific file under the Scripting Modules Tree and the file's Load Module should be set to "When Referenced."

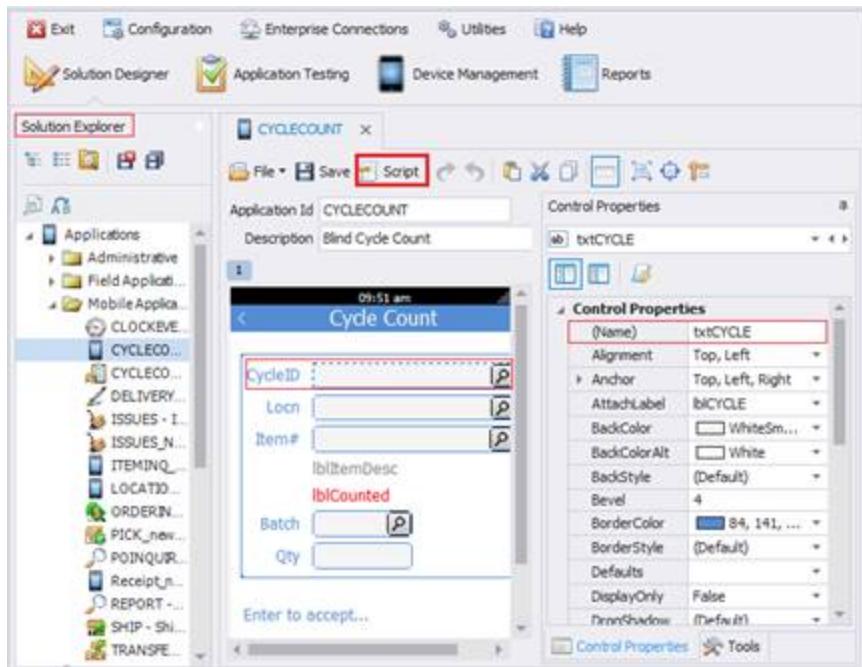
(See the **Scripting Modules – Load Modules** tree for more details.) Check those modules you wish to include with the current application. Modules designated **Win32.bas** and **RFgen.bas** are automatically included for each application.

Shortcut Keys

The 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 icon, a scripting window similar to the one below displays. Below shows the changes to the Object and Proc lists when a textbox prompt is selected.



```

1  ' (C)opyright 2003-2014 The DataMAX Software Group, Inc.
2
3  Option Explicit
4
5  Private msCycle As String
6  Private msLocn As String
7  Private msItem As String
8  Private msBatch As String
9  Private mnQty As Currency
10
11 Private Const mbUpdateInv = False ' Changing this form-level
12 ' cycle count transaction
13 Private msCCColumns As String
14 Private msCCData As String
15
16 Private Sub Form_Load()
17  On Error Resume Next
18
19  ' Initialize special controls
20
21  lblCounted.Caption = ""

```

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 described in the section "VBA Events" near the end of this book.

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

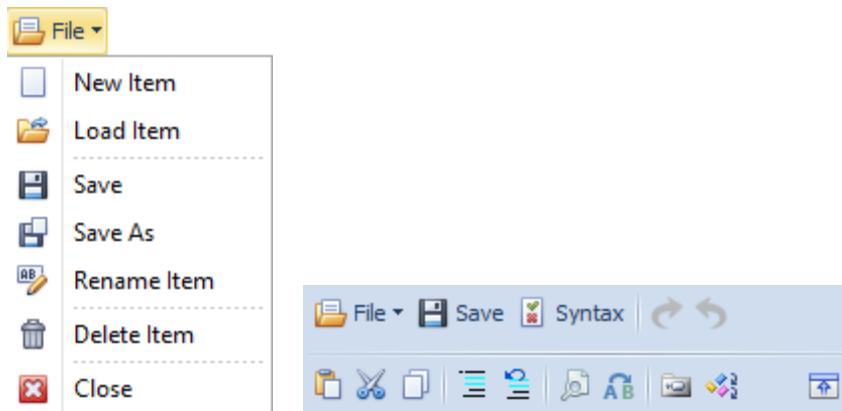
Note: VBA code is very sensitive to variable typing. Most errors result from using a String variable when an Integer or Long data type is required.

Scripting Modules Menu Bar

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.

This tree also supports files containing non-globally-loaded modules which can be used with a specific application and, is loaded when the Reference icon is clicked.



The **New Item** option clears all fields and provides a blank window for creating a new bas module.

The **Load Item**, depending on the changes made, can be used to save a change (i.e. a value change to a property) or simply load an item.

Save saves the current script to the solution database and **Save As** will allow you to copy and save the file to the solution database. **Rename** will retain the same file but allow you to change the file name without creating a new copy in the solution database.

Delete will delete the selected node and **Close** will close the node.

The **Syntax** option performs a syntax check of the script. Also provided are **Copy**, **Cut**, and **Paste**, and **Undo** or **Redo** a prior action.

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

The **Find in File** searches all applications and macros at the same time and **Replace in Files** icon allows the user to find text or replace text within the current application. The **References** icon opens the Available References window.

The **Loading Sequence** 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.

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 Application tree for clarification of the modules purpose.

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.

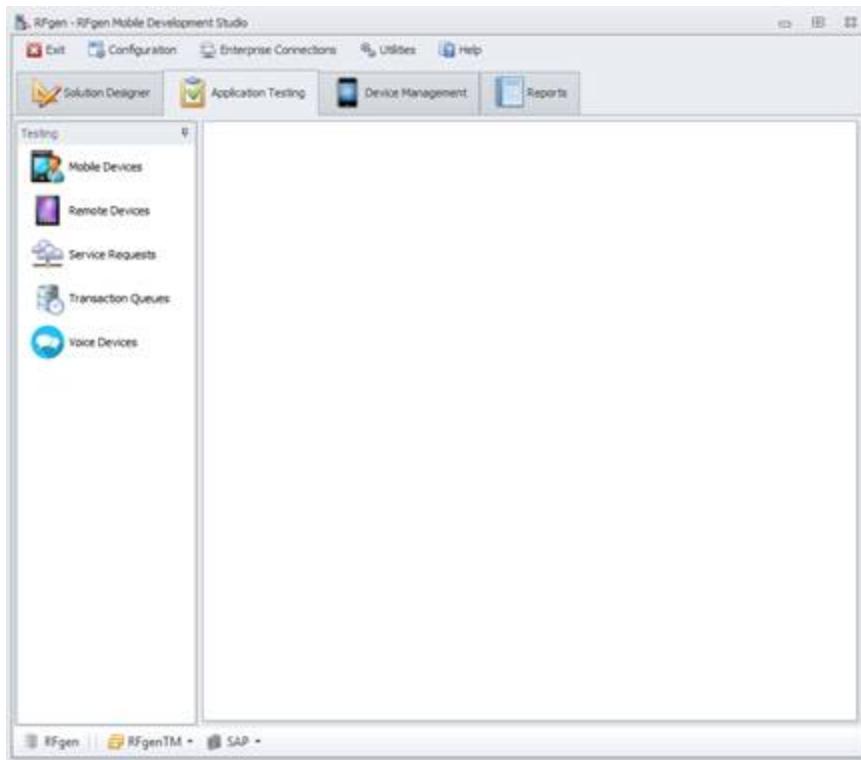
The **Load Module** option *At Startup* loads the BAS file or a user-created module globally when the application is started up, and is available (globally accessible to other functions) all the time.

The option *When Referenced* only loads the BAS file or a user-created module for the application that needs to use its functions. For example, if you created special modules for an application, and had set the *Load Module* value to "When Referenced," when you open the application, the module will not load at startup. To access your special modules, you can click on the Reference icon  from an application menu bar while in the scripting view, and the modules will load as a backend process (one you don't see).

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.

Application Testing



The Application Testing panel allows you to test your solutions, applications and transactions using various methods.

The **Application Testing panel** contains the tools to:

1. Test and debug developed objects (i.e. applications, screen maps, services) hosted in a variety of configuration modes or device types.
2. Test devices with profiles in Mobile Mode (also called Batch mode) Note: This is supported for Windows Desktop and CE devices.
3. Test the data steam portions of other third-party data collection solution that are voice driven and use unique hardware and software.
4. Test Service Requests (i.e. for integrations with cloud based solutions).
5. Test the time and processing of Transaction Queues.

Connection Limitation

When you test device connections with the Mobile Development Studio, the Mobile Development Studio limits the number of simultaneous connections between itself and remote devices. For example, you can only have a total of 3 connected sessions.

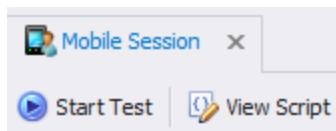
Mobile Device Testing

The following steps describe how to test an application once its been developed.

Before you begin testing, make sure your application has been added to the Main Menu.

Refer to **Menus and Roles** for more details on how to create menus and link them to applications. The following steps start with a Login app.

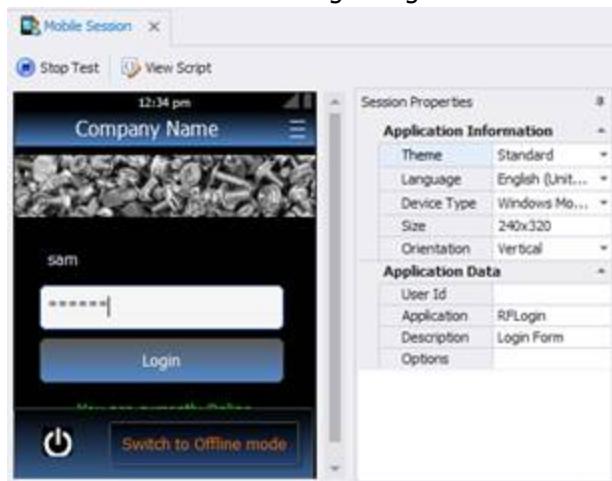
Mobile Session Menu Bar



The **Start Test** button toggles the starting and stopping of a session.

The **View Script** button switches your view to the script where other debug functions are available.

1. Click on the **Application Testing** tab.
2. Click on **Mobile Devices**.
3. A blank application with the option to Start Testing displays.
4. Click on **Start Test**. The RFgen Login Screen and Session Properties displays.



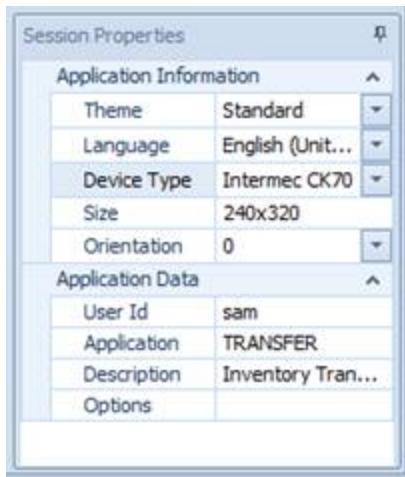
5. Enter "SAM" as the user Login.
6. Press **Enter** for Login (password is blank).
7. Click on "Sign In" or press **Enter** to login.
8. Scroll to towards the middle or bottom of the screen. If successful, the Main Menu displays.
9. Click on the icon to access a submenu or application and begin testing.

Mobile Application Testing Without the Login Screen

Follow these steps if you want to skip to the application menu and bypass the Login screen.

1. In the Solution Designer, select Applications > RFgen Login > RFgen Script tab
2. In the RFgen Script area, enter:
3. `App.CallForm("<the name of your application")`
4. Save the RFgen Login app. Use it to launch your app.
5. Click on Application Testing > Mobile Application > RFgen Login > RFgen Script tab > Start Test
6. The application listed in the RFgen Login App will launch along with a Testing Environment panel.

Session Properties



Theme – Select the style theme for testing the overall look and feel of the app. It 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.

Language – Changes 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/

Device Type – Changes the emulator's mobile device type.

Size - Change the emulator's size to test different shaped devices such as forklift displays, tablets, smart phones, or PCs.

Orientation – Changes the display between a vertical or horizontal view.

Application Data – displays the data entries used in the application

View Script option

This 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:

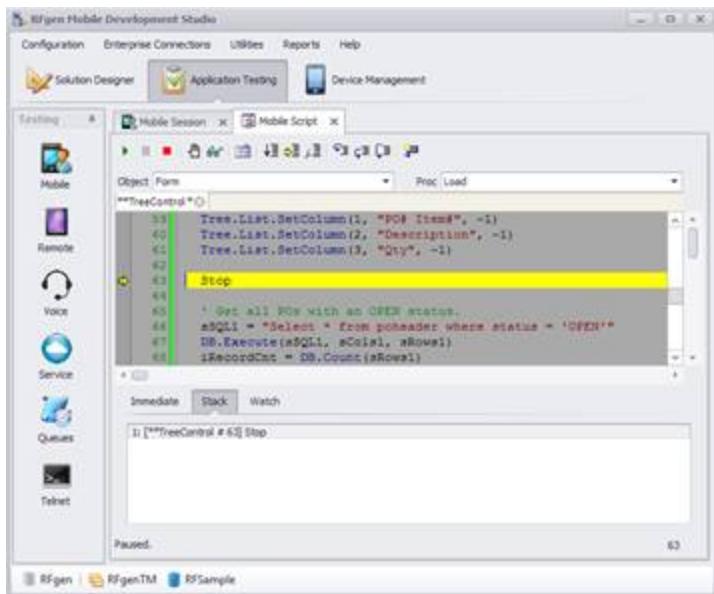
F5 = Play (green arrow)

F8 = Step (first step button)

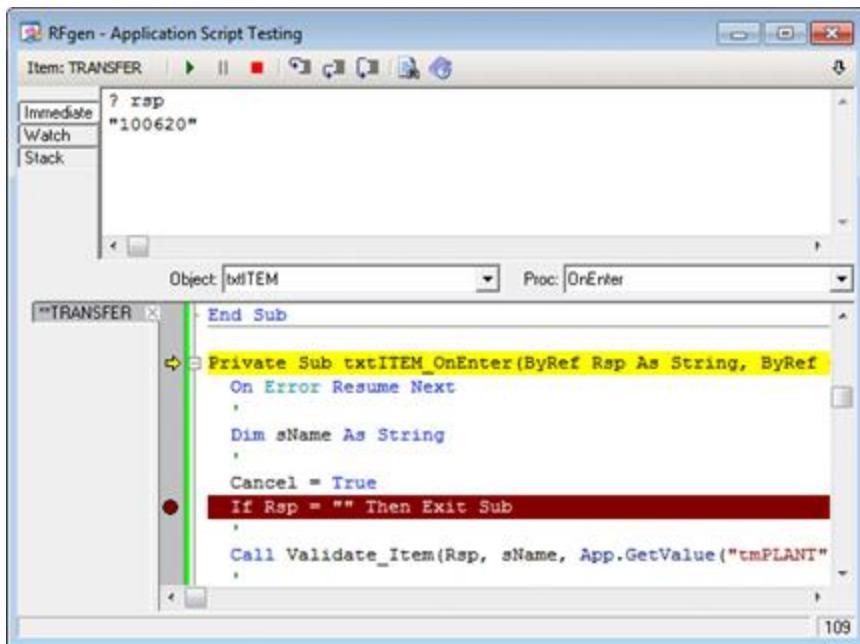
Alt F8 = Step Out (second step button)

Shift F8 = Step Over (third step button).

To Debug Code

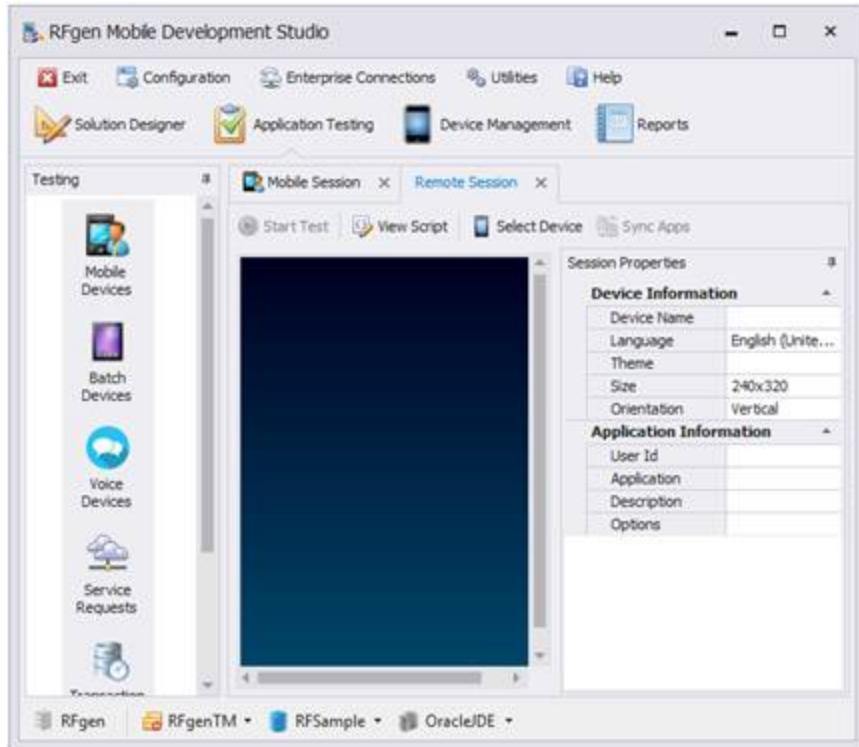


1. Click on the blue pause button and then trigger an event. Code execution will be paused on the first line of that event.
2. To debug code in the Form Load event, add the line 'Stop' so that execution is halted on that line.



3. After you have entered your user ID and password, the menu assigned for the user will display.
4. Use your keyboard/keypad arrow keys to select an application and press <Enter>.
5. In 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.
6. The Application Values grid on the Watch tab shows what your data item (record) looks like as data is added.
7. Click on 'Stop Test' to stop the testing process.

Remote Device Testing



This option enables you to discover clients, and test and debug the application/code on the Mobile Client if its:

- **Remote Debugging** is enabled in the RFgen Configuration file
- Accessible via the network
- Note: Remote Device testing is different than Mobile Device testing because you are testing an application that is stored and launched from the device, not the server. Therefore, to debug the Remote Device, it needs to be on a network connection accessible by the server.
- Once you have debugged the code, you can synchronize using the Sync Apps button. The changes are synchronized based on the profile used to provision the device.

Remote Session Menu Bar



The **Start Test** button toggles the starting and stopping of a remote session.

The **View Script** button switches your view to the script where other debug functions are available.

The **Select Device** button allows you to select the device you want to connect to from a list of devices that are

in session. The option to add a device is provide if its needed.

The **Sync Apps** button will synchronize changes based on the profile used to provision the device.

Debugging Remote Devices

The Remote Device must already have the following:

- The Mobile Client Profile installed
- Wireless access to the mobile device
- The Remote Debugging enabled on the device under RFgen Client Configuration > Device Settings

1. Click on **Application Testing > Remote Devices**.

Note: Remote sessions is not used for Thin Clients.

2. To launch a remote session, select the **Select Devices** icon.

A list of devices on the network should display. Select the device you want to test and debug. (In general, you do not need to Add a Device as the device should already be on the network.)

3. In order to access the device remotely from the server, the device's **RFgen Mobile Configuration > Device Configuration: Remote Debugging** box should be checked.

4. Once you connect with the device, you should have control of the device as if you are the user, and be able to begin testing apps remotely.

Application and debugging is the same as in Mobile Testing with this exception: From the Remote Devices menu, click on the Sync **Apps** button to synchronize changes. Code synchronization between the server and device depends on how the device's mobile configuration file was setup.

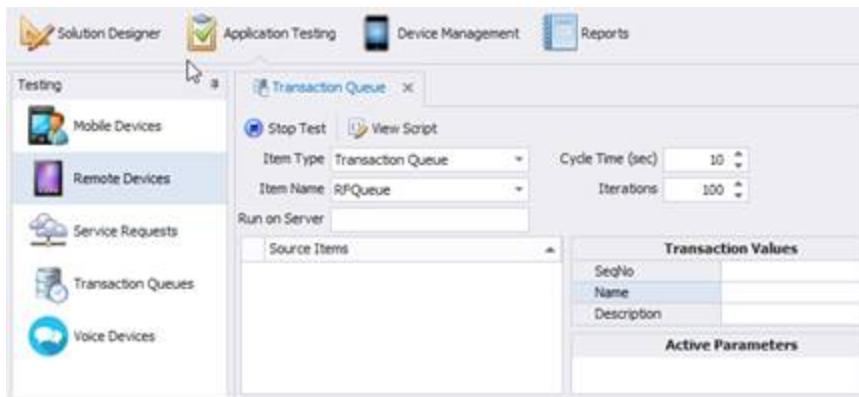
Service Requests Testing

To test RFgen Services between your RFgen server and clients, you'll first need to install and configure an application running in character mode or graphical mode.

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 Queue Testing

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



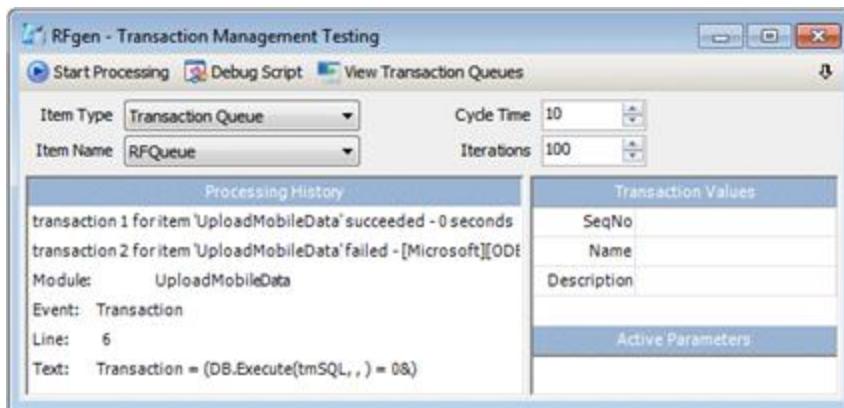
The **Item Type** option switches between queue processing and timed Events. If Transaction Queue is selected, the **Item Name** specifies which queue is to be processed. If Transaction Event is selected, then the Item Name list is populated with the configured events under the Transaction Management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

Run on Server option is used to specify which server should be used in the event you have multiple servers for load balancing. If this field is left blank, RFgen will bypass this field and process transaction queues on all systems connected to it. But if you have the server IP address or server name, RFgen will process transactions with this the server.

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.

Device Management

The Device Management tab is a collection of Windows explorer-like functions that enable the server to communicate with mobile devices for the purposes of deploying RFgen components, installing RFgen authorization code (license to use the software), and performing other utilitarian tasks.



All communication and interaction with the mobile device goes through the CNC, when enables the device to "listen" for connection requests by the server. (For information on obtaining and downloading this application, refer to the *RFgen Installation and Upgrade Guide*.)

Device Management Options

The **Access / Authorized Devices** screen tracks authorized devices and enables administrators to: a) manually authorize mobile clients, and; b) manually authorize Thin Clients (if the server is configured to Restrict Online Access), or b) view automatically authorized Thin Clients. This feature is used for authorizing all types of client connections.

The **Application Deployment** provides various ways to transfer CAB or CNC files to Windows CE\Windows Mobile devices. The server can then update mobile applications, menus, users etc. as part of as needed for data collection. If the server is upgraded, and new client files need to be distributed, this function can help perform this task. Full profiles can also be sent to the device.

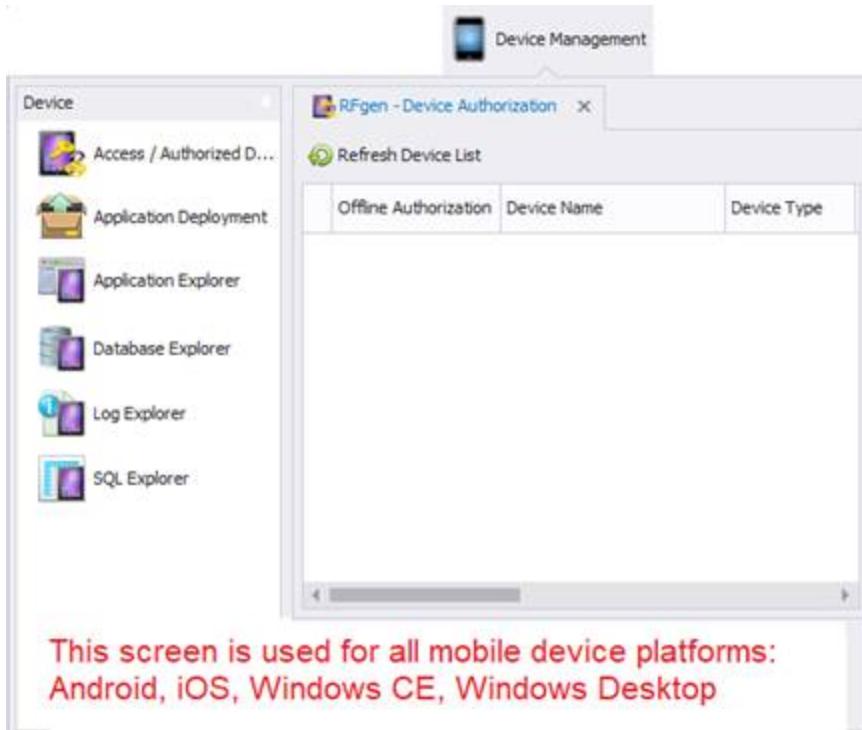
The **Application Explorer** will support exporting components to Windows CE\Windows Mobile devices.

The **Database Explorer** shows the data on Windows CE\Windows Mobile device's database.

The **Log Viewer** will show the error log file on Windows CE\Windows Mobile devices.

The **SQL Explorer** will let the user read and write directly to the user database on the device

Access / Authorized Devices

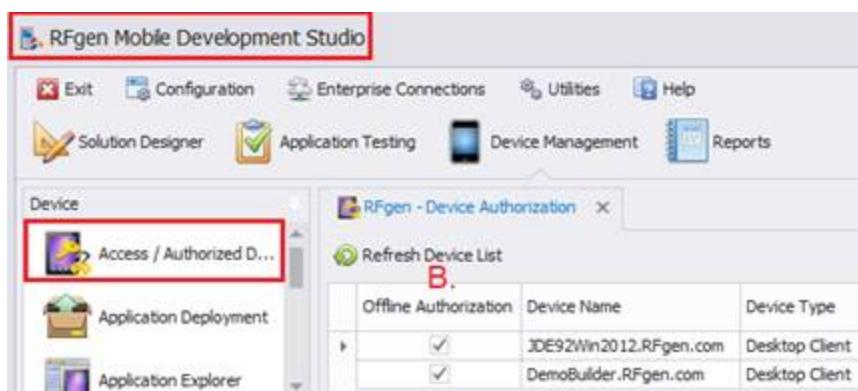
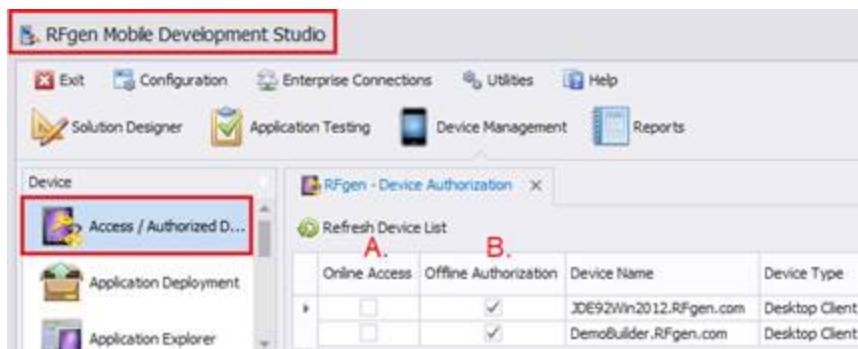
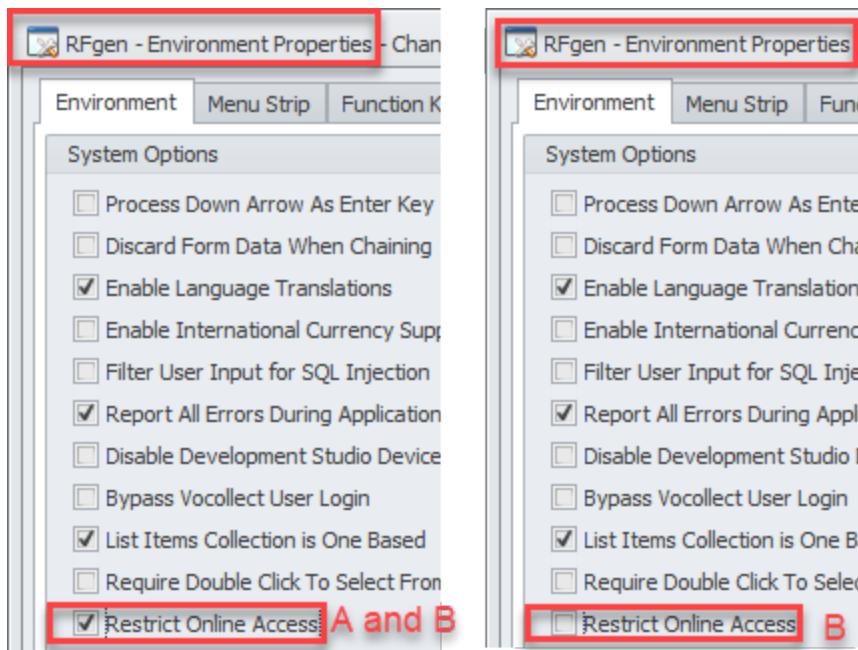


For security purposes, you can configure the server (see [Configuration > Environment Properties > Restrict Online Access](#)) to allow Online Clients (Thin Clients) to connect with the server manually or for efficiently purposes, allow connections automatically. *Manual connections* means you must checkmark which clients are allowed to access the server before they can connect and synchronize information with the server. Automatic means the server will accept Thin Client connections without the intervention of a RFgen administrator.

For details on how to manually authorize a Thin Client, see "[To Authorize or Remove Devices](#)".

All Offline Clients (**Mobile Clients**) require manual authorization. Manual authorizations also require the proper licensing for Mobile Clients on the server. There is no setting to make authorize these automatically. However, once the online or offline client is authorized, its allowed to connect until the profile or device GUID changes.

To Configure “Online Access”



In the screen above:

- A. "Online Access" contains the checkboxes for manual authorization of Thin Clients.
- B. "Offline Authorization" contains the checkboxes for manual authorization of Mobile Clients.

The configuration of manual authorization of devices (hide or show "Online Access" column) is controlled from the **Restrict Online Access** setting on the **Mobile Development Studio > Configuration > Environment Properties** screen.

If **Restrict Online Access** is checked, then all online client connection requests will be rejected by the server unless the server finds they have been approved at a prior time. This also displays columns A and B in Access / Authorized Devices screen.

If **Restrict Online Access** is unchecked, all online clients requesting a connection will be automatically approve.

Regardless of whether **Restrict Online Access** is checked or unchecked, column B will display. All Mobile Clients will be rejected until the device's *Offline Authorization* is checked.

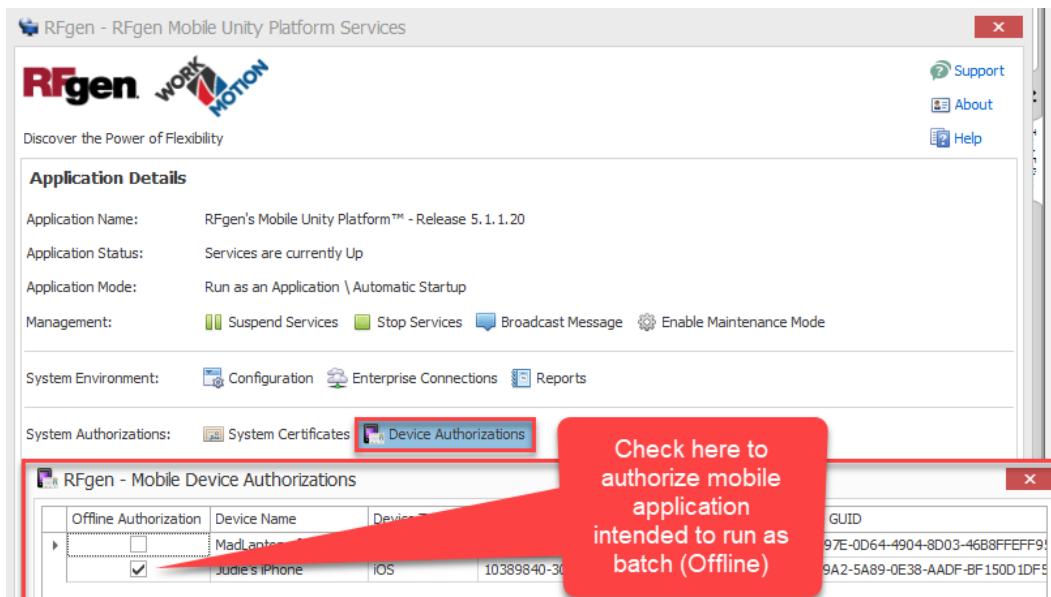
To Authorize or Remove Devices

Each authorization will consume a license for the corresponding type. If automatic authorization is configured, licenses are assigned to online clients on a "first request" basis.

From the server, click on **Device Management > Access / Authorized Devices**.

To authorize a Thin Client: Check the device's **Online Access** box, then click the Refresh button.

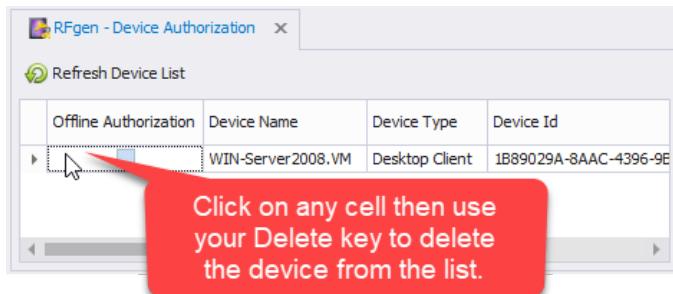
To authorize a Mobile Client: Check its **Offline Authorization** box, then click the Refresh button.



If the Offline Authorization box is checked, the server will deploy a Mobile Client license to the mobile client if

the server has a sufficient number of batch/mobile licenses available. If, however, you click this box, and the client does not have a mobile profile installed, an error message will display.

To remove an authorized device and free-up a license: Delete the row of the device. (Click in one of the cells, and press "Delete".)



If the number of available Mobile licenses are exceeded, the Mobile Client will still be able to connect to the server and receive profile changes, but it will not be able to upload processed data to the server.

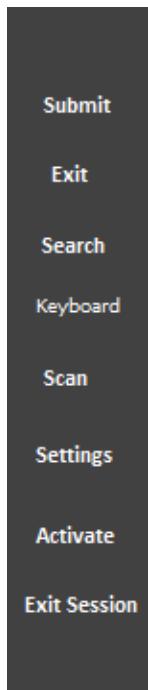
To authorize a Batch Client

If you are using a mobile client in Batch (offline mode or batch mode), a batch license must be installed to the device and then activated/authorized.

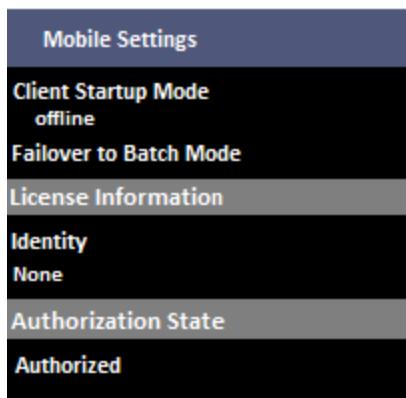
Before you can download a license (via a deployment of the mobile profile), a connection to the server must be established. The connection may require the RFgen Administrator to authorize the device connection from the server.

Android or iOS: Once the mobile profile has been deployed / installed to the client, you activate the license via the "Activate" command from the menu strip.

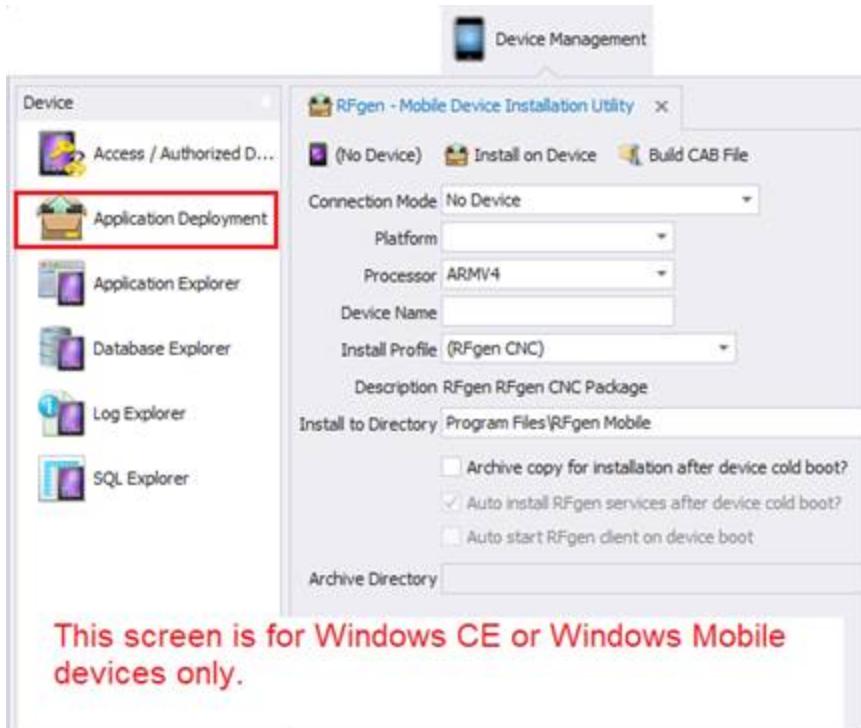
1. Connect to the RFgen server.
2. When your mobile application Login displays, click on the menu button to display the menu strip.
3. Open the Menu strip, and click on **Activate**. This button is only present when the certificate is present for a download.



You can also verify if the device is authorized by reviewing the device's RFgen Configuration > Mobile Settings > License Information: Authorization State.



Application Deployment



The **Application Deployment > RFgen – Mobile Device Installation Utility** tab is used for deploying mobile applications to Windows Desktop or Windows CE systems ONLY. It is not used for Android or iOS devices.

Application Deployment - iOS or Android

To deploy mobile applications to iOS and Android devices, you must first install the RFgen Client software to the device, and during the device connection to server process, the Mobile Profile is deployed to the client.

Refer to *How To Provision iOS and Android Clients* for details.

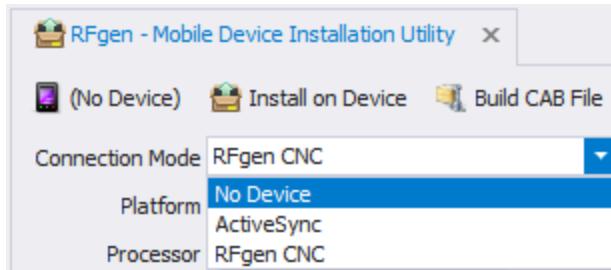
Before you can deploy a mobile app to a Windows CE device, the following are required:

- Installation of the **RFgen Windows CE client.exe** on the server.
- Creation of the CAB file for the Windows CE\Windows Mobile device on the server.
- Installation of the CAB file to the Windows CE\Windows Mobile device.

Once the **RFgen Windows CE client.exe** is installed, you can then use the **Deploy Mobile Applications: Mobile Device Installation Utility** to build CAB file(s) – which are required for use of the mobile apps developed in the Mobile Dev Studio – and then install them to the Windows CE device.

The server also needs the **RFgen Windows CE client.exe** to communicate with the device. It installs the Client Network Control (CNC) service which is maintained as an Installation Profile.

Build CAB File Settings



Before you select any options, decide which connection method you want to use BEFORE you build your CAB files.

Connection Mode Settings

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 (typically with a USB drive). For example, if you plan on simply using a USB thumb drive/flash drive to install the CAB files, then use the *No Device Connection Mode*.

ActiveSync means that the device is connected to the PC using Microsoft's ActiveSync program – This requires a physical wired connection between the server deploying the CAB file and the device. The easiest method for building and installing CAB files is to use the "*Active Sync Connection Mode*".

RFgen CNC means that the CAB will be moved from the server to the device over a wireless network. This requires the installation of the *RFgen Windows CE Client* or *RFgen Windows Desktop Client* and discovery of the device via the device button.

Platform and the Processor Settings

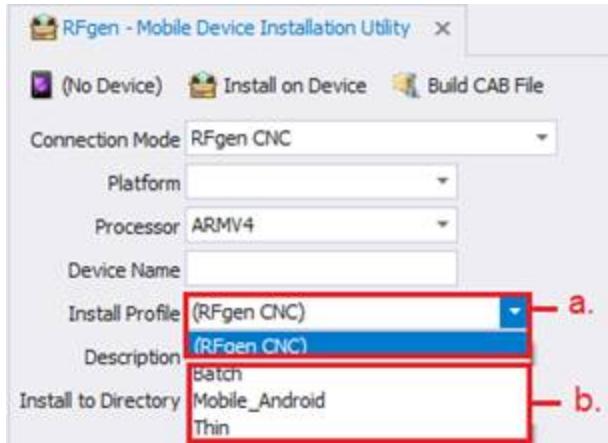
These combine to tell which files must be installed on the device. Usually in the settings on the mobile device, there is an About program that will show these values. If the processor says some variation of "X-scale", it is referring to the ARM processor type. It is recommended to use the latest ARM 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 which client profile they want to deploy.

- a. The "(RFgen CNC)" profile is unique in that it's the ONLY file that will be installed. Use this option only if you unable to install the Windows CE Client.exe package to the Windows CE\Mobile device.

- b. This profiles list is populated from the profiles created under Solution Explorer > Mobile Profiles. Most Windows CE Devices will have the CNC service component already installed if you already installed the Windows CE Client.exe to the device.



The **Install to Directory** is where all files will be placed.

The **Archive copy for installation after device cold boot?** 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 RFgen services after device cold boot?** 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 RFgen client on device boot** 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.

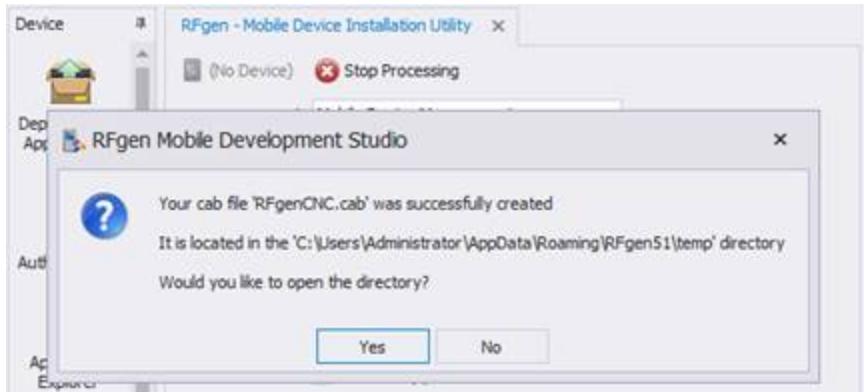
Step 1. Build the CAB File

Downloaded RFgen Windows CE.exe file from:

<https://www.rfgen.com/product-portal>

1. Make sure you install the version that matches the version of the RFgen server.
2. Install the **RFgen Windows CE client.exe file** to your server.
3. On the server: Click on **Device Management > Deploy Mobile Applications**. The RFgen-Mobile Device Installation Utility displays.
4. Complete the form. Refer to the "Build CAB File Settings" above for a description of each field and its list of options.

5. Once your information is complete, click on **Build CAB File**. A message similar to the one below displays.



Depending on which Connection Mode you selected, this process will build these files: RFgenCNC.CAB, RFgenCNC.DAT, and RFgenCNC.inf and RFgenMobile.

Step 2. Select a Method for Transferring the CAB files

If your CAB files were built with the:

- **No Device Connection Mode**, then skip to **Step 4**. It is assumed you plan on copying the files from the server to the devices using physical device such as a USB flash drive.
- **Active Sync Connection Mode**, then follow **Step 3: Active Sync Connection process**.
- **RFgen CNC Connection Mode**, then follow **Step 3: Wireless Connection process**.

Step 3: Wireless Connection Process

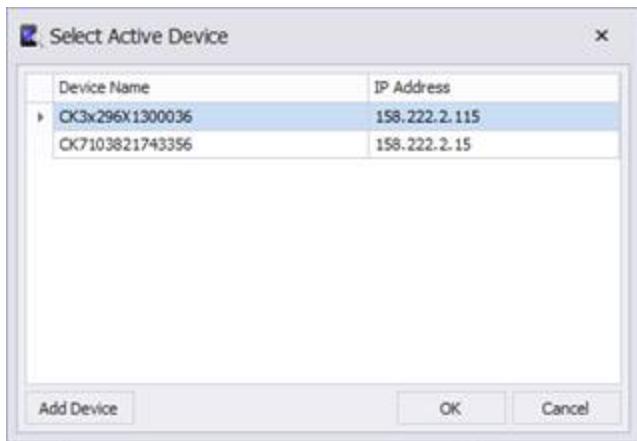
BEFORE you start this process:

Verify your device is on the same network as the server and that the server is accessible.

Install the **Windows CE Client.exe** to the target device. For *install details*, refer to the *Installation and Upgrade Guide*.

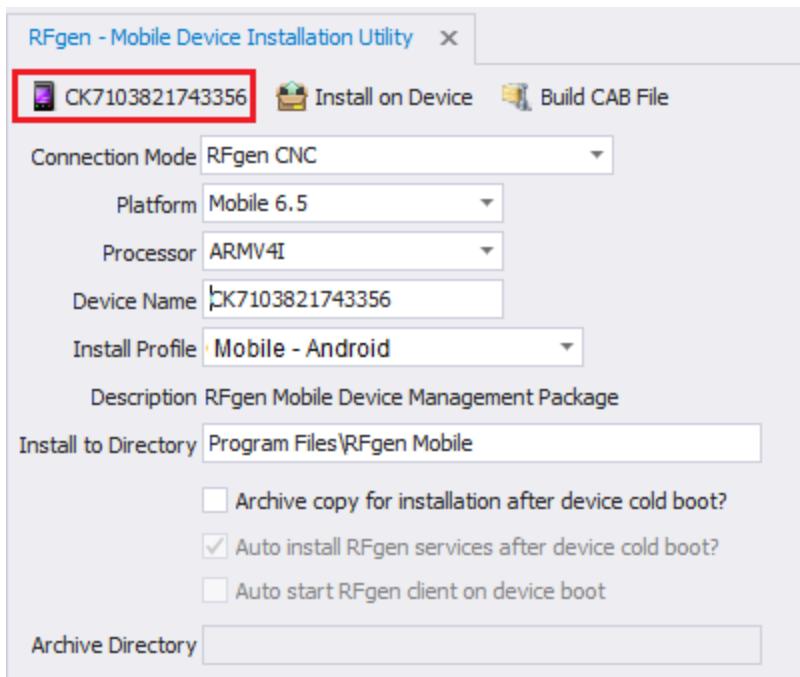
Note: The Windows CE Client.exe installs the RFgen CNC service which enables the client to communicate with the server, and allow the server to perform actions on the device when device is connected via a wireless or cellular connection. It also enables the server to discover the device.

- a. Click on **(No Device)**. The **Select Active Device** screen with a list of discovered devices displays.

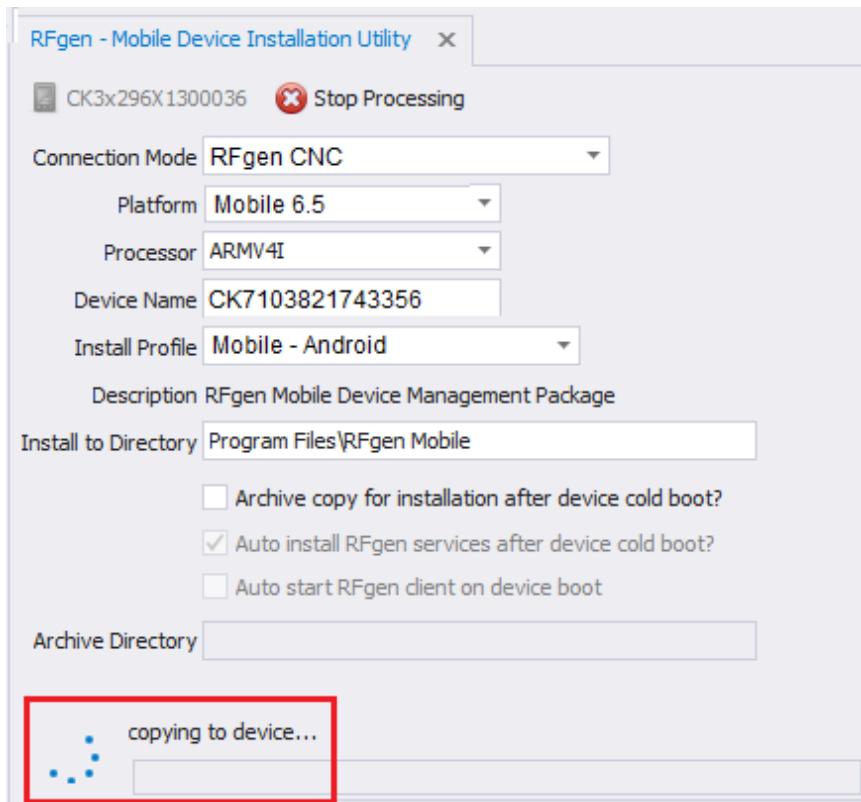


b. Select the device and click **OK**.

c. The icon shows the selected device. You are now ready to install files.



d. Click the **Install on Device** option to download and install the profile to the device. The process status displays near the bottom of the screen.



(If needed, you can also stop a process by clicking on the Stop Processing button.)

e. A brief message on the device may show "Creating RFgen database ..." if you installed a Mobile profile.

f. After the copying to device completes, the mobile device can be disconnected from the server.

g. On the device, navigate to the **Start**, and launch the RFgen Configuration program. This will "unload" the CAB files to the device.

Step 3. Active Sync Connection Process

Follow this process if you built your CAB files with the **Active Sync Connection Mode** and want to install via a USB connection. In this process you will be: a) Selecting the device via the *No Device* button; b) Copying the CAB files to the device via the Active Sync process, and; c) Installing the CAB files using the *Install on Device* button.

PreRequisites:

- The CAB files you built in Step 1.
- Ensure the Windows Mobile Device Center program installed on both the server and the device.
- Ensure you have a physical USB connection between the device/device's cradle and the server.



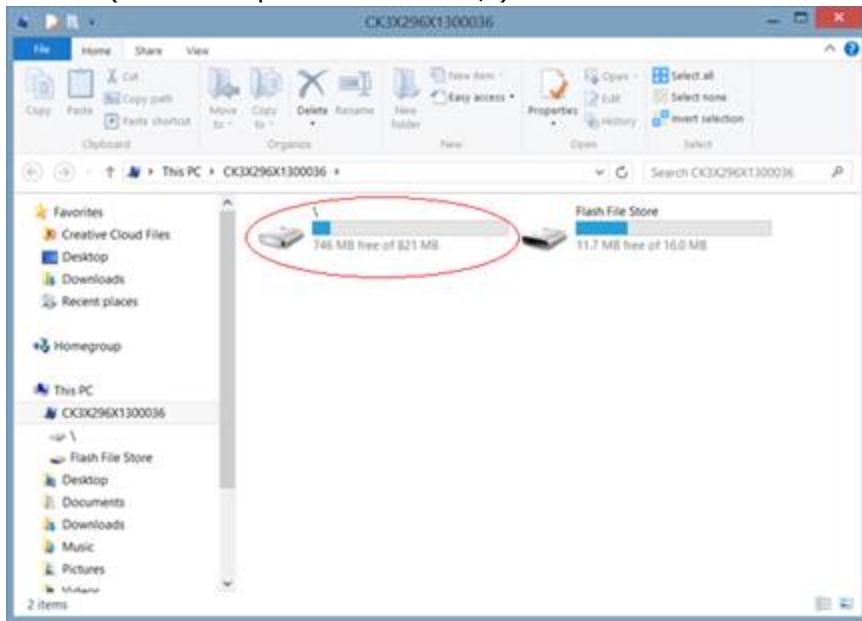
- a. Launch Windows Mobile Device Center.
- b. When the Connection icon shows you are connected, select **Select Connect** without setting up device.



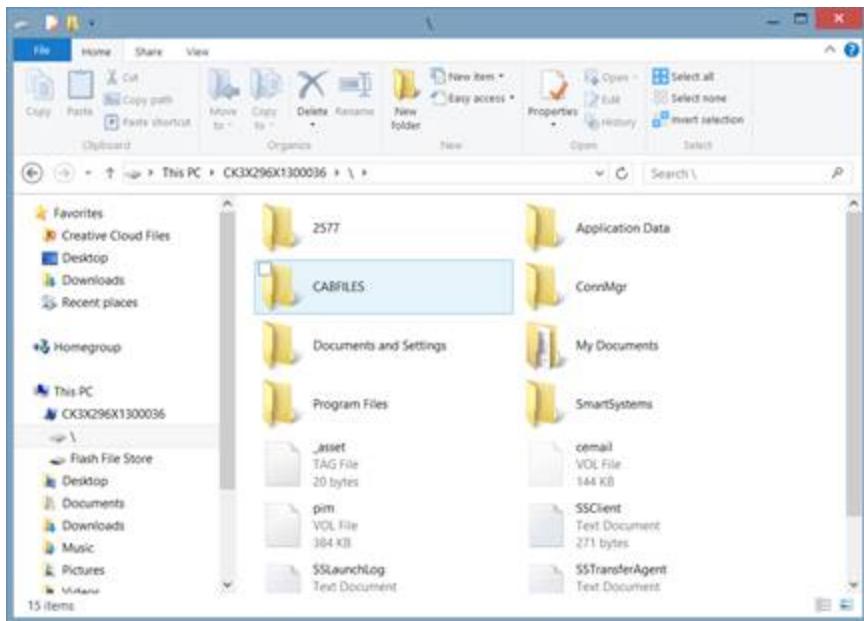
c. Select **File Management – Browse the contents of your device.**



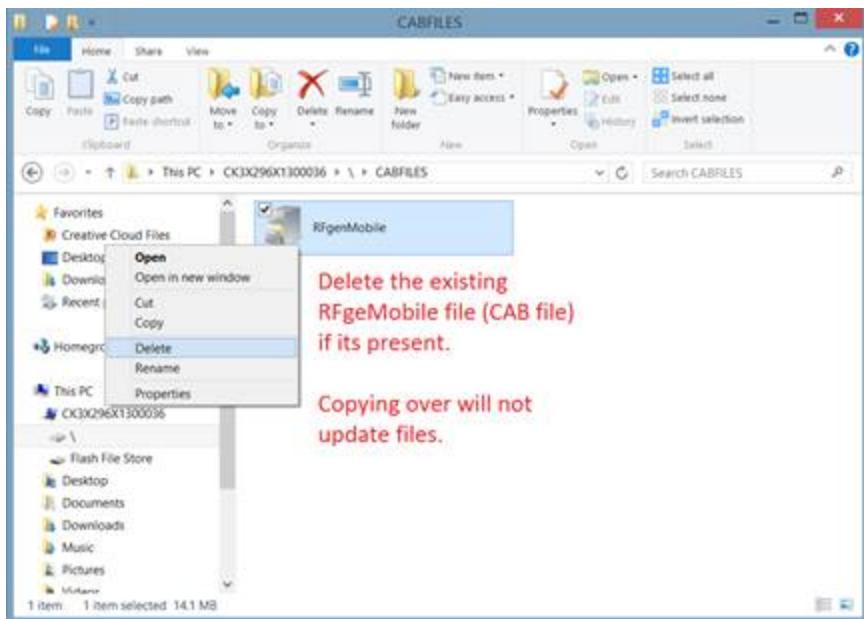
d. You should see the device's storage locations from your File Management view. Select your device install location. (In this example we chose root "\").



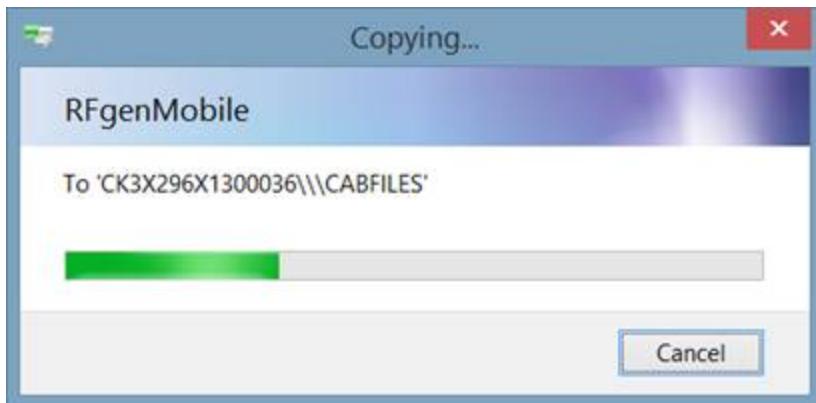
e. Navigate to the **CABFILES** folder.



f. If a cab file is listed, delete this first.



g. Copy the new RFgenMobile file to this location.



h. When its done, this message will display:

Your cab file was created, copied to the device, and the installation started.

i. On the DEVICE, this message will display:

Installation
The previous version of DataMAX RfgenMobile will be removed before the new one is installed. Select OK to continue or Cancel to quit.

Tap **OK** on the device.

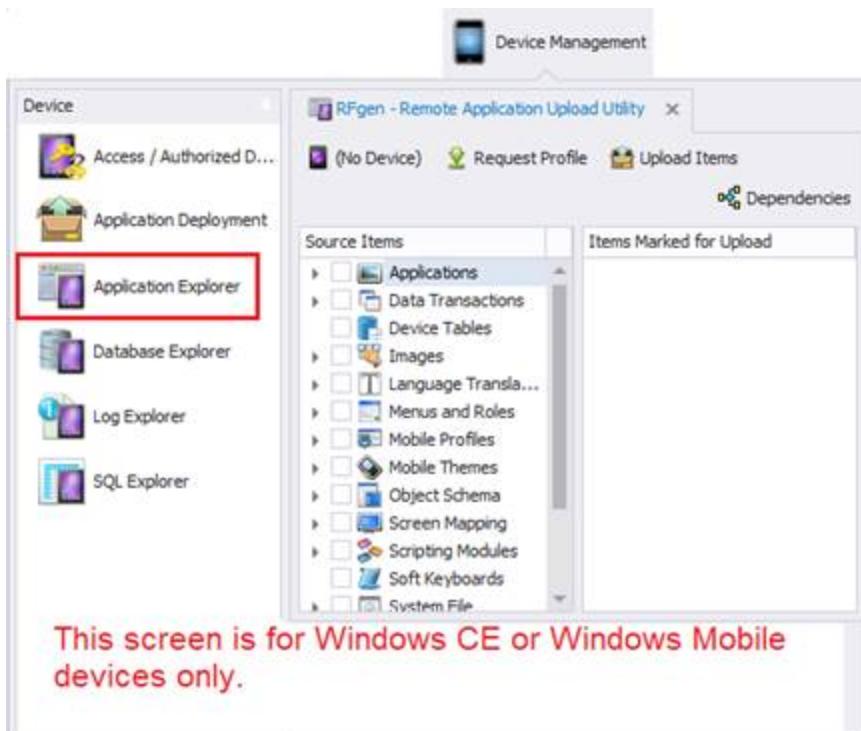
j. On the DEVICE, this message similar to this will display:

Choose a location to install "DataMAX RfgenMobile":
 Device
 iFlash File Store
Space Needed: 14614KB
Space Needed: 781428 KB

k. Select your location and tab **Install**. The status will show its install progress. When its done, this message will display:

RfgenMobile.cab was successfully installed on your device.
If you need more storage space, you can remove installed programs.

Application Explorer



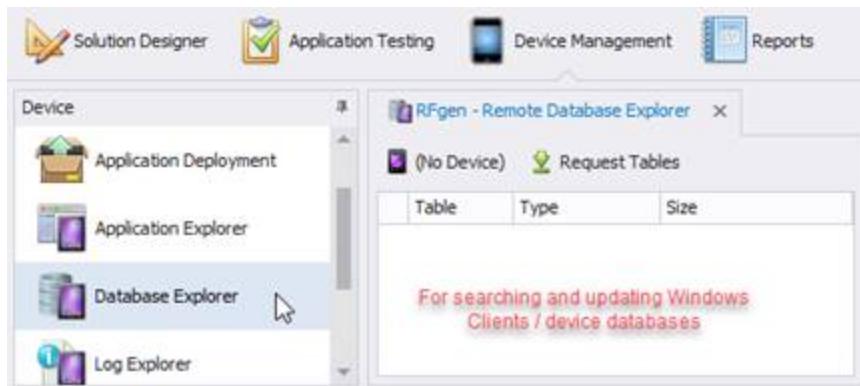
The **Application Explorer** is used to send updated or additional components to Windows CE/Mobile devices that have the RFgen CNC (Client Network Control) software installed on the remote client. This feature is not used on Android or iOS devices.

To update iOS and Android devices, refer to *How To Provision iOS and Android Clients*.

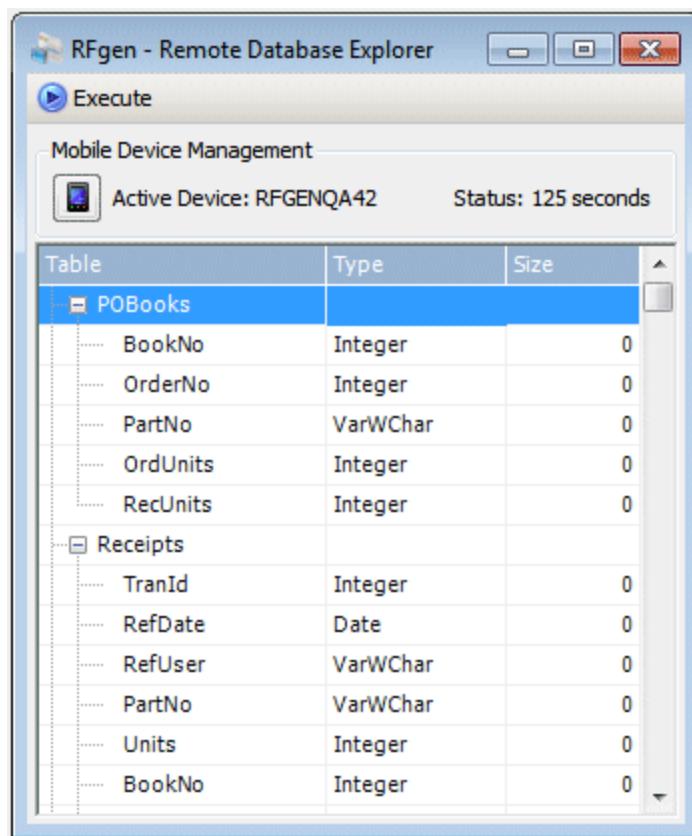
Click on **Active Device** to select the device you plan to export files to.

When choosing components to export, note that Thin client solutions do not have Users, Menus, Applications and other components because they are provided through the server. *Mobile client solutions*) do have users, menus, applications and other components since mobile clients are able to function like mini-RFgen servers when disconnected from the network.

Database Explorer

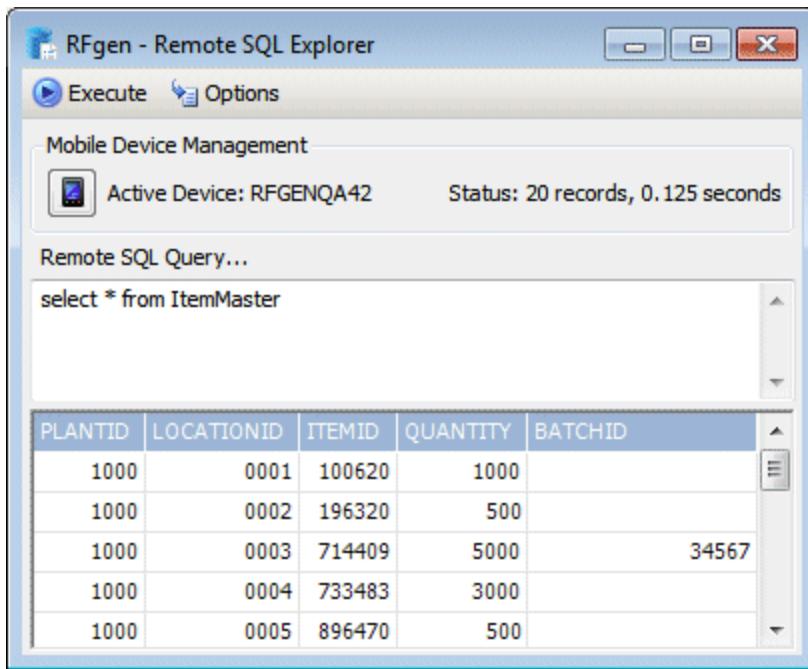


This screen is used to see the schema of the tables stored on Windows client\devices when the device is setup as a mobile client solution.



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.

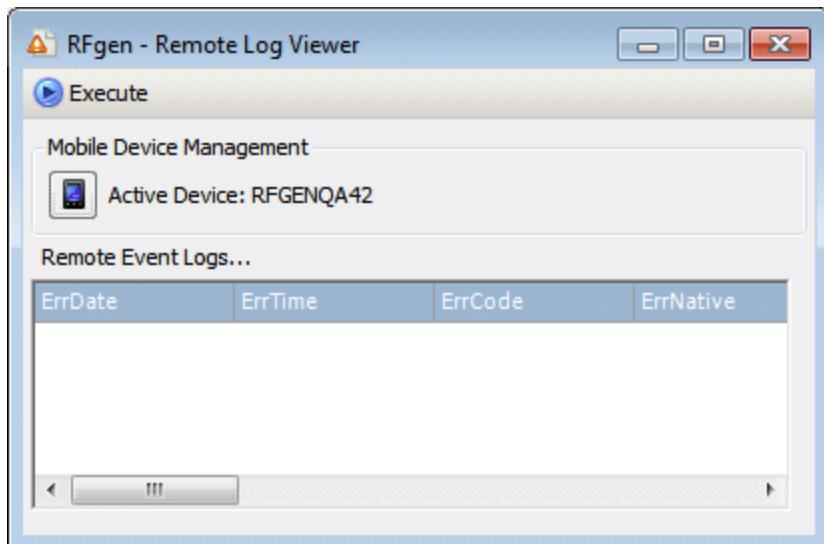


Click on the Remote SQL Explorer menu option and then select the active device. Enter any valid SQL statement, or multiple SQL statements separated by a semi-colon in the Remote SQL Query window and click the Execute menu option. In the case of multiple SQL statements, the statement that contains the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

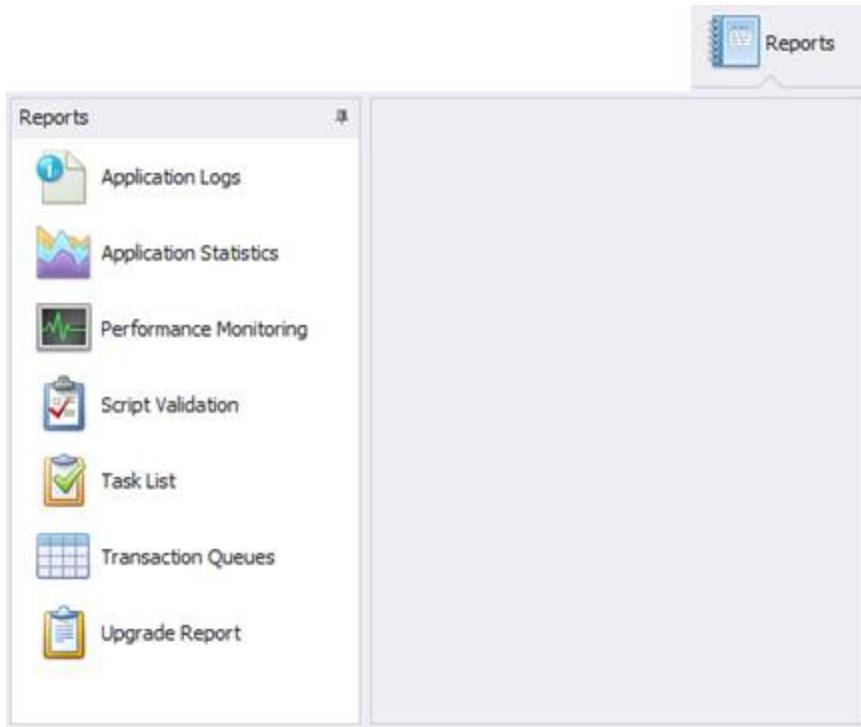
The Options menu item allows for a restricted number of rows to be returned in case the statements will bring back more data than desired. It will also switch the grid display into a list of previously executed SQL statements for easily repeating previous statements.

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.



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.

The **Application Logs (Logfiles)** display system errors. It includes a SQL Filter and Export to Excel tool.

The **Application Statistics** report displays statistics regarding database and ERP transactions.

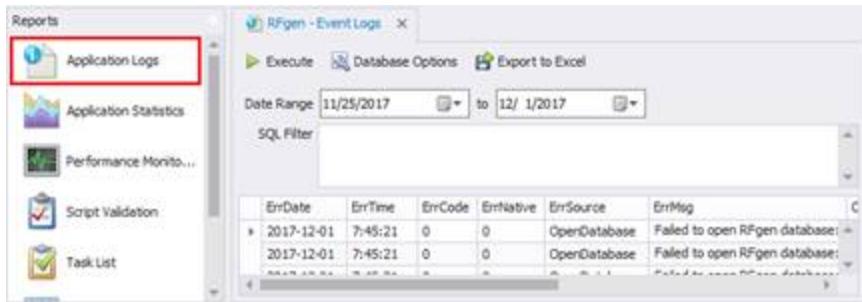
The **Performance Monitoring** log lists events generated by the execution of scripts that exceeded a threshold value. (i.e. Flagged events that exceed processing time thresholds.) It includes Display Options and an Export to Excel tool.

The **Script Validation** report will syntax check all application scripts, VBA modules and transaction macros and display which ones have errors.

The **Task List** includes a list of programmer tasks to be completed if the script was flagged with a comment and the words "TODO".

The **Upgrade Report** will generate a report on what was left unfinished or needs further investigation when you upgrade RFgen via RFgen's Mobile Unity Platform installer. For example, if you were to update from RFgen 5.0 to 5.1, this report may warn you about which applications may need more attention.

Application Logs



Clicking on **Reports > Application Logs** displays a window showing system error messages.

The errors are stored in the IPC.db database.

Application Log Menu Bar



The **Execute** button runs the SQL Filter described below.

The **Database Options** sets the error log's maximum number of rows and allows you to delete the table.

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

To sort errors

There are multiple ways to filter or sort the information in an Error Log:

- By using the Date Range
- By using the SQL Filter
- By clicking the up/down arrow in a column header

ErrDate	ErrTime	ErrCode	ErrNative	ErrSource	ErrMsg
2017-11-30	16:09:09	0	0	OpenDatabase	Failed
2017-11-30	16:09:09	0	0	OpenDatabase	Failed

- The SQL Filter is the “where” portion of the select command in the error log table. For example, to display only those rows which have “RFLogin” as the value in the AppID cell, enter **AppId = 'RFLogin'** in the SQL Filter area and click **Execute**. If you wanted to see only the rows where the ErrDate contained “2017-11-30” you would enter **ErrDate = '20171130'**.
- 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.

To revert to a prior Application /Even Log

If you had executed an SQL statement and wanted to revert to the original log, simply remove your statement and click on Execute again.

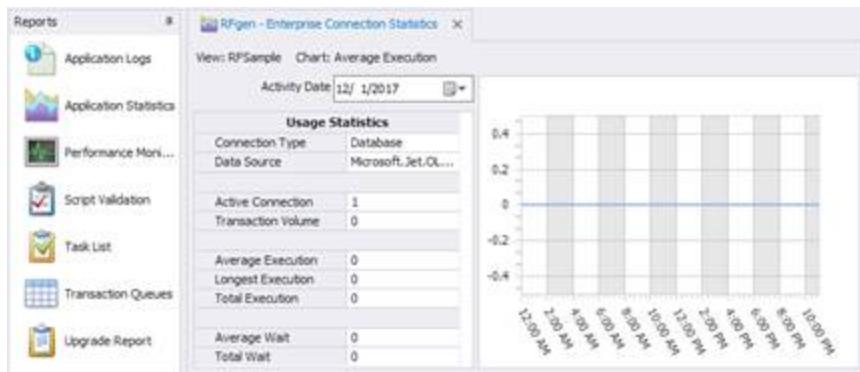
To delete an Application/Event Log

The Application / Event Logs display under **Reports > Application Logs**.

To remove a line item, click on **Database Options** menu and select **Delete Items**.

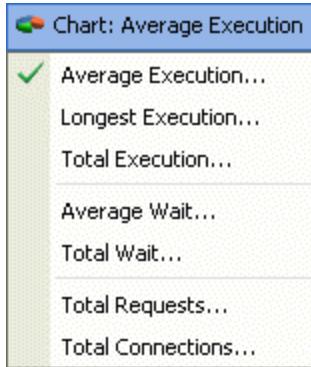
Application Statistics

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.

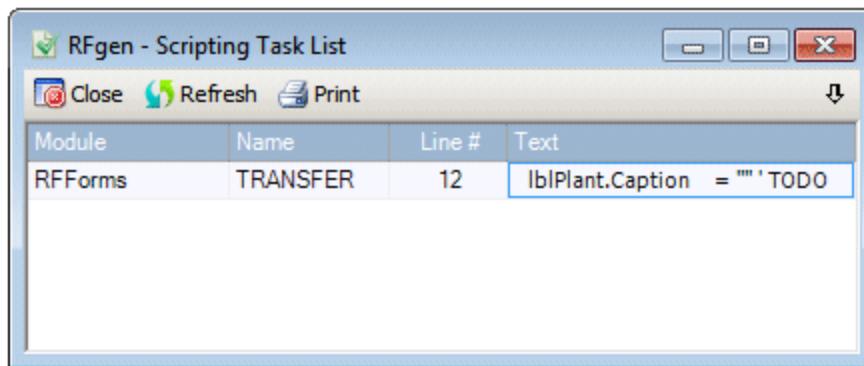
To enable the Statistics, choose the Configuration / Performance Monitoring menu option, select the Record Usage option and change the value from Disabled to some increment for refreshing the data.

Performance Monitoring

The Performance Monitoring report 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.

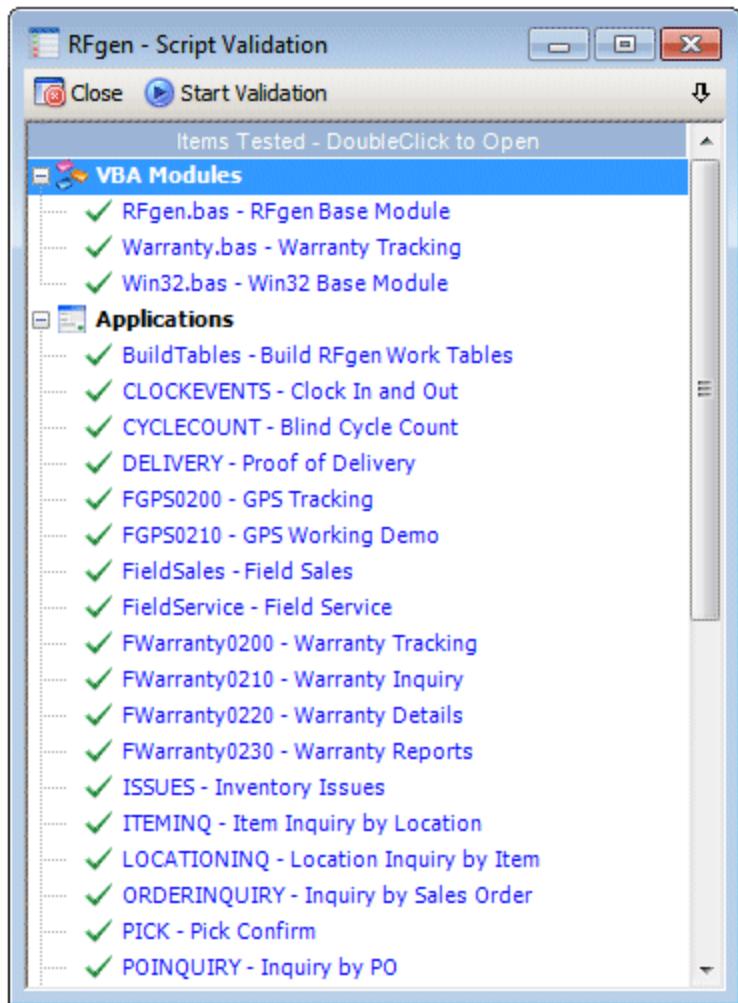
Task List

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.



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.



Queued Transactions

To view queued transactions, click on **Reports > Queued Transactions**.



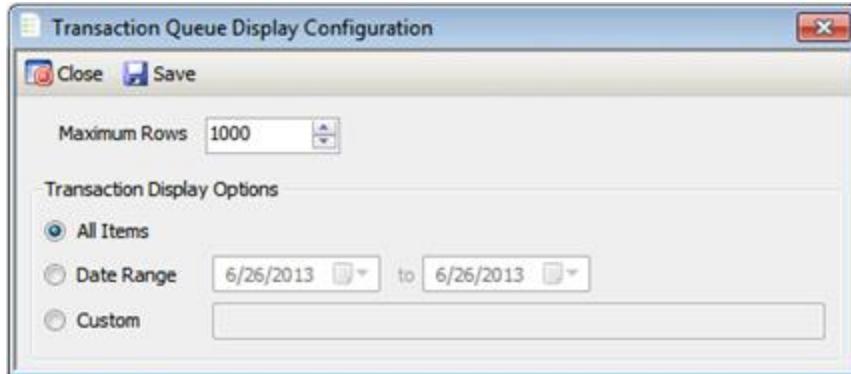
Three types of logs are available: 'Queued' transactions are data collection entries waiting to be posted to your host application (typically because your host has been offline or not available); 'Completed' transactions and 'Failed' transactions may also be displayed. Transactions may be edited, reposted, 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.

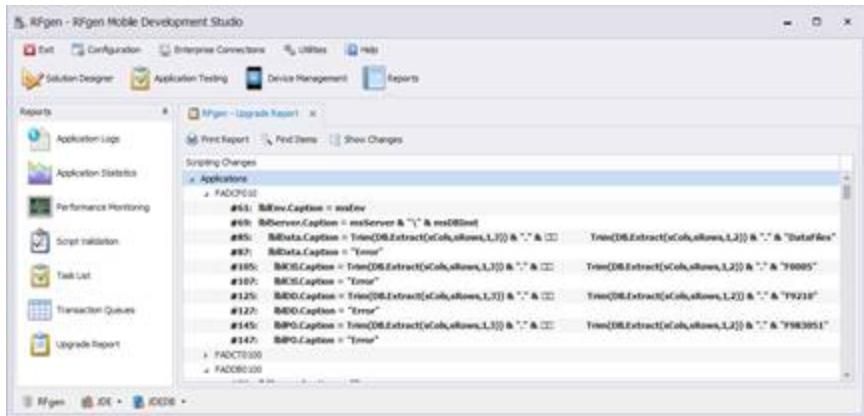
Transaction Display Options – **All Items** shows an unrestricted list of entries and **Date Range** will limit the entries to a date-based on their created date.

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.)

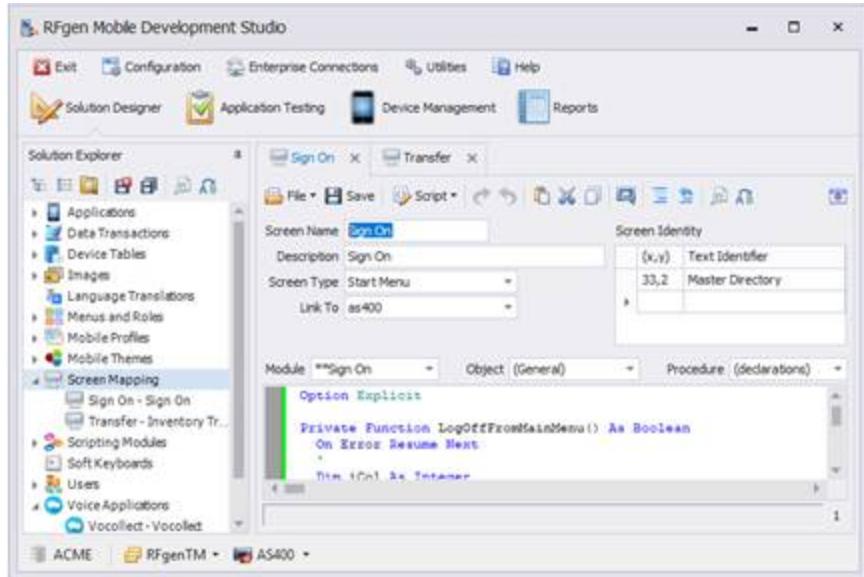
Upgrade Report



When RFgen Mobile Device Studio is upgraded to newer version, if an issue was encountered, the issue will display in this Upgrade Log. If the install was successful, the Upgrade Log will remain blank.

In general, the Upgrade Log will provide "Upgrade Markers" which are indicators that 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.

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.

For more details on how to record macros, see [How to make Screen Mapping Work](#).

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 for Screen Mapping

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 - Screen Mapping

There are many ways a 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 remappings 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.

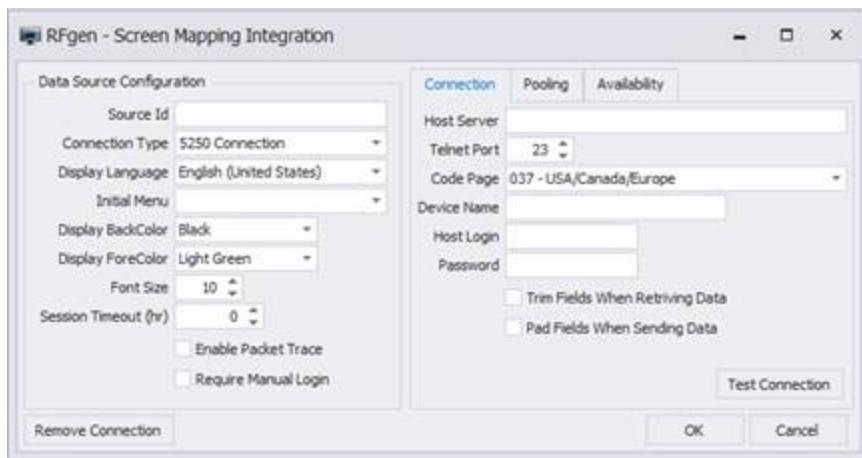
See the VT220 Key Mapping section for more information on Hot-Keys.

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 on **Enterprise Connections > Add New Enterprise 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 the **Back Color**, **Fore Color** (the color of the font) 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.

If the **Require Manual Login** is checked, a connection request is created between the user and the ERP system. If this box is unchecked, the user login uses the ERP connection between RFgen and the ERP system.

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.

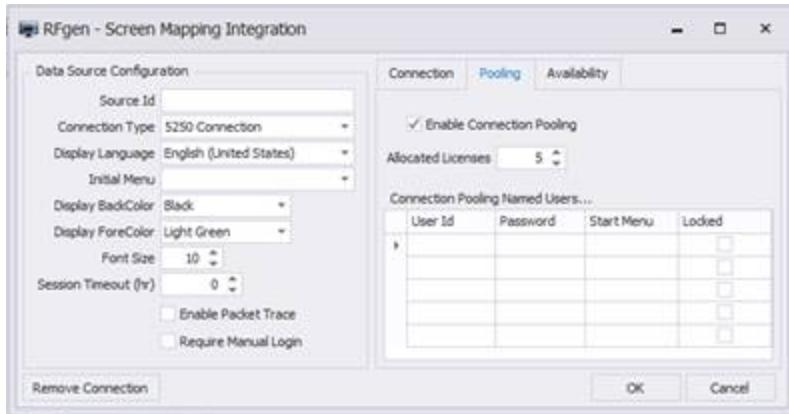
The **Host Login** and **Password** fields are used only if SSH is used when connecting to the host system. Under the VT220 options, if Connect via SSH is checked then the Host Login and Password are required.

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 from RFgen to an ERP 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.

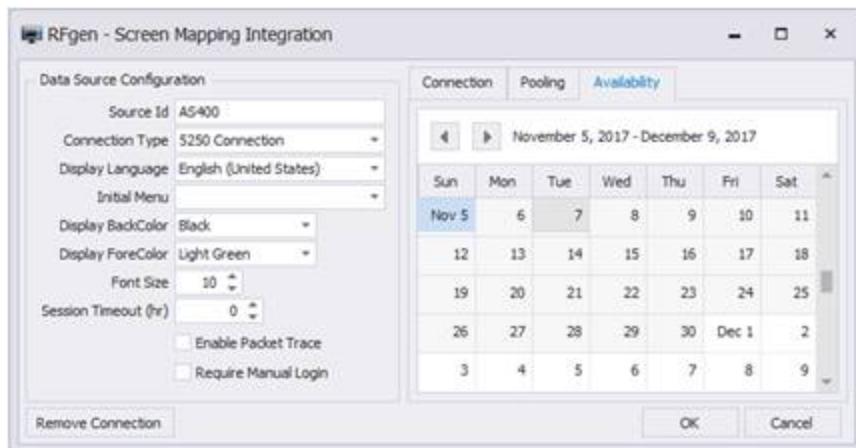
Locked The ability to lock a session means that the session can ONLY be used with the specified main menu and will not allow other main menus, even if all other available sessions are in use. For example, there are 10 pooled sessions, five locked on main menu A and five locked on main menu B. If a session 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.

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.

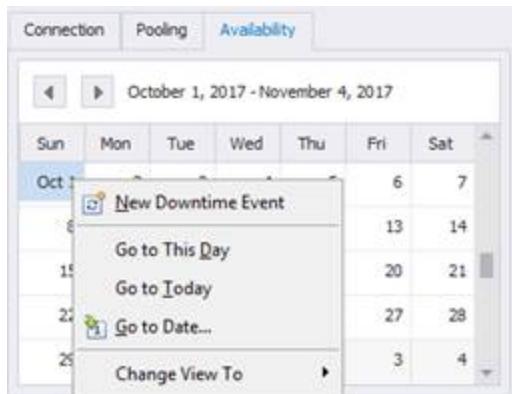
For example, there are 10 pooled sessions available and the first five have one main menu assigned and the last five have a second main menu assigned. When a session with the second main menu is requested, the 6th session handle will be used. This is only significant because of the Locked property.

Availability

The Availability tab in the 'Screen Mapping Integration' 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.

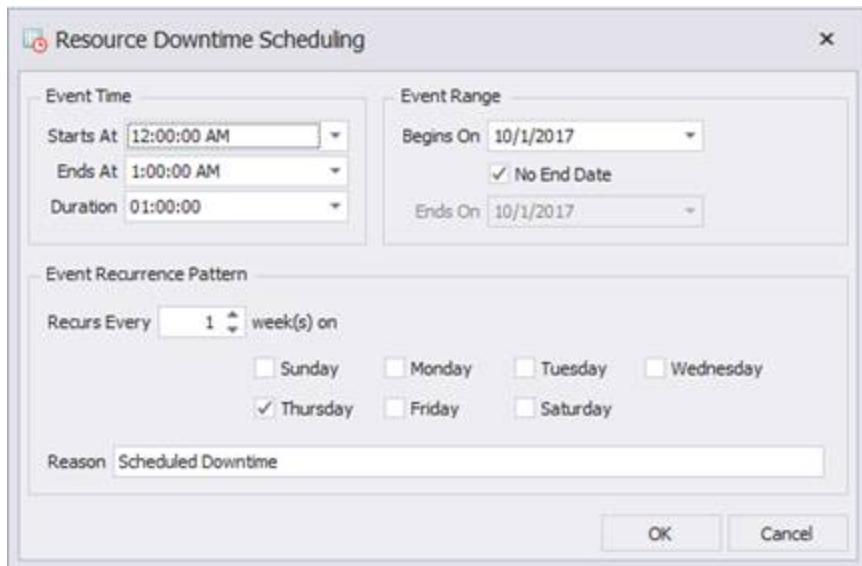


To schedule downtime, right-click on the date or days in the calendar and select the appropriate item from the menu.



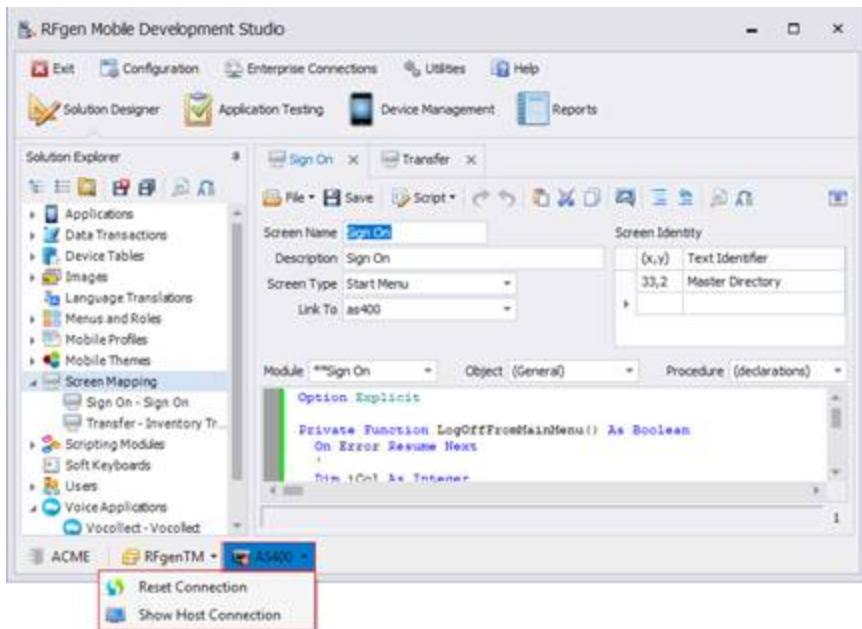
In this example, the New Downtime Event was selected.

In the Event Time box, the connection will be unavailable every Thursday, for 30 minutes between 12 AM and 1 AM beginning Oct 1, 2017 and reoccur until an End Date is supplied.

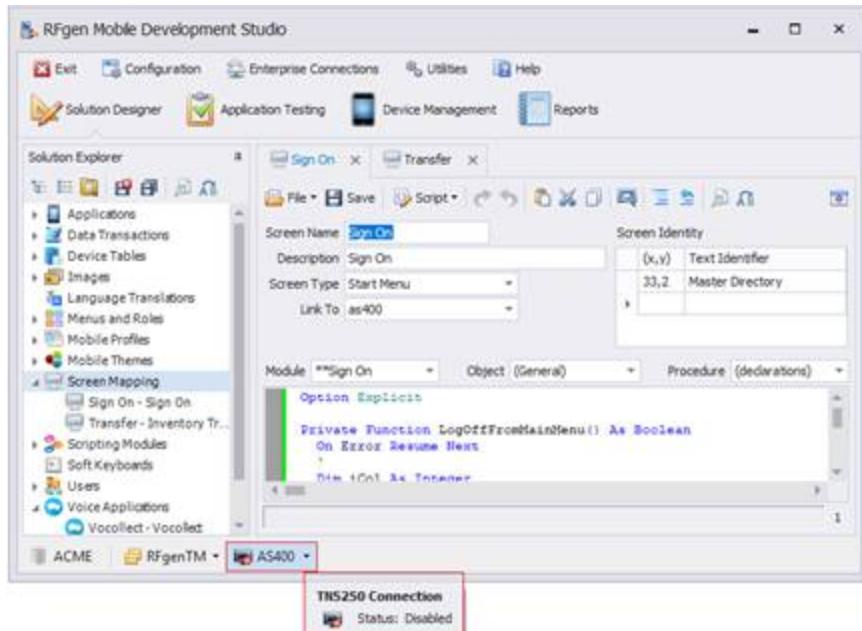


During the down time, transactions can be 'queued' for automatic posting to your host when the connection becomes available.

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.

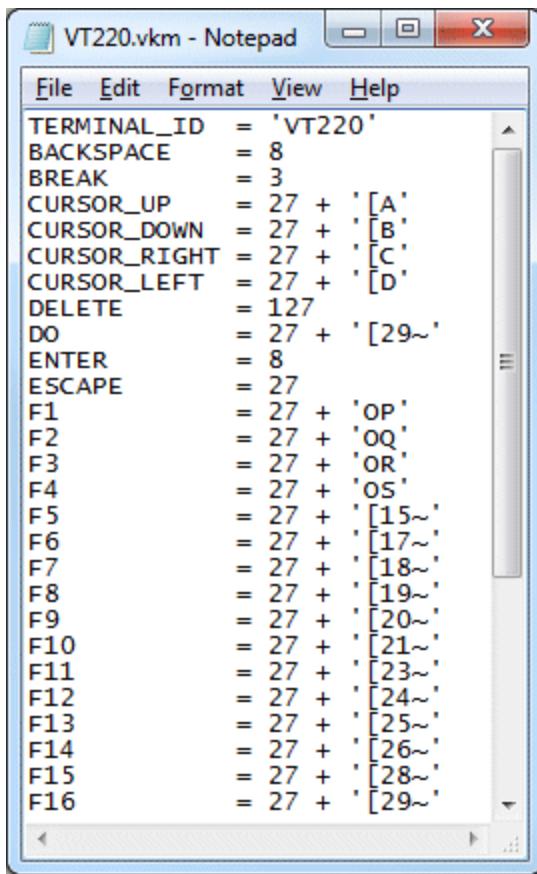


With a host connection established, your screen mapping development project is ready to be started.

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

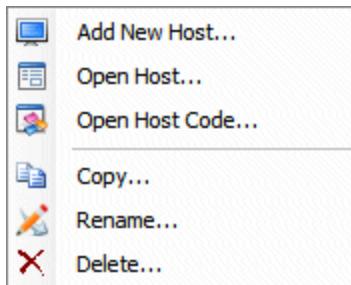
file called VT220.VKM will be placed in the install directory and can be edited with a text editor.



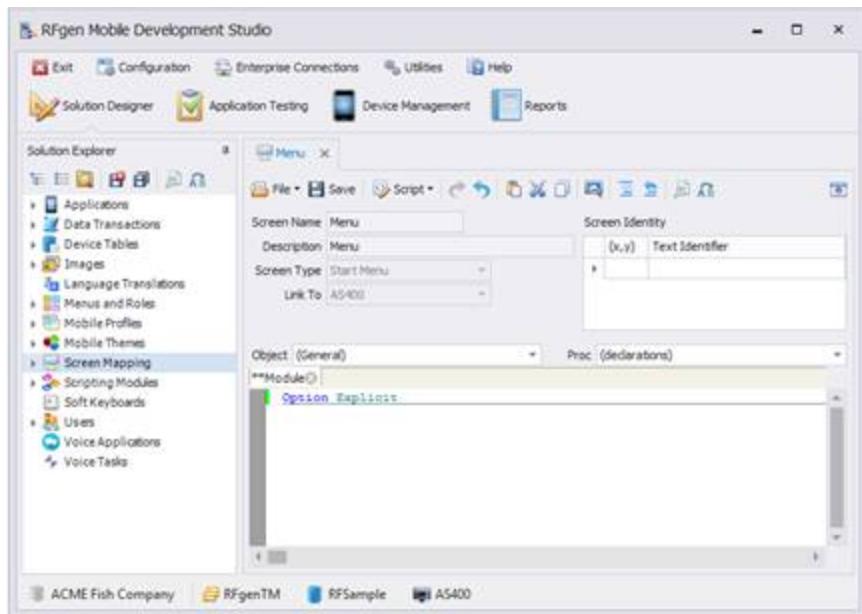
```
VT220.vkm - Notepad
File Edit Format View Help
TERMINAL_ID = 'VT220'
BACKSPACE = 8
BREAK = 3
CURSOR_UP = 27 + '[A'
CURSOR_DOWN = 27 + '[B'
CURSOR_RIGHT = 27 + '[C'
CURSOR_LEFT = 27 + '[D'
DELETE = 127
DO = 27 + '[29~'
ENTER = 8
ESCAPE = 27
F1 = 27 + 'OP'
F2 = 27 + 'OQ'
F3 = 27 + 'OR'
F4 = 27 + 'OS'
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~'
```

How to record a macro for a screen map

Right-click on **Screen Mapping** to add a new host, or create a new screen mapping macro or right-click on the title of an existing macro to make additional changes.

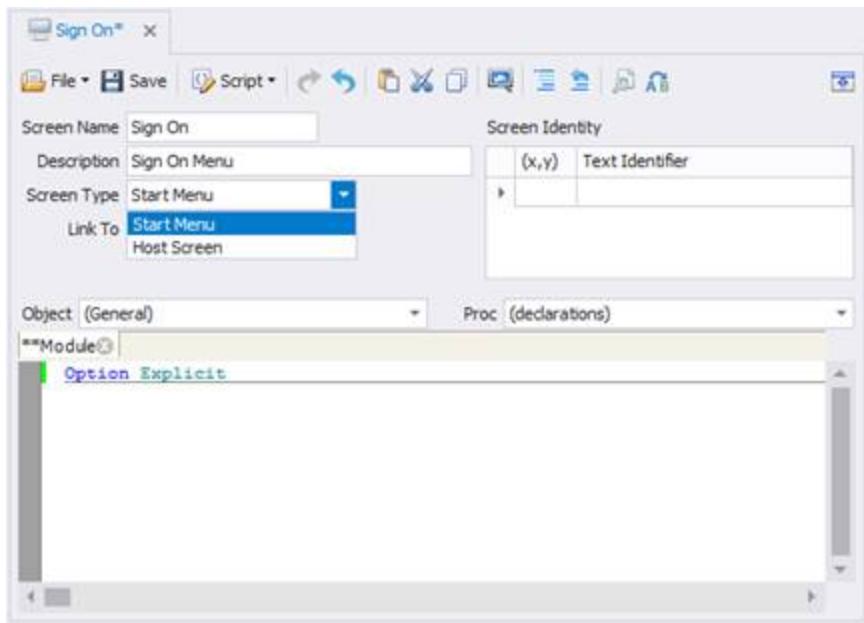


Double-clicking on an existing macro opens the Host Screen Editing window.



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



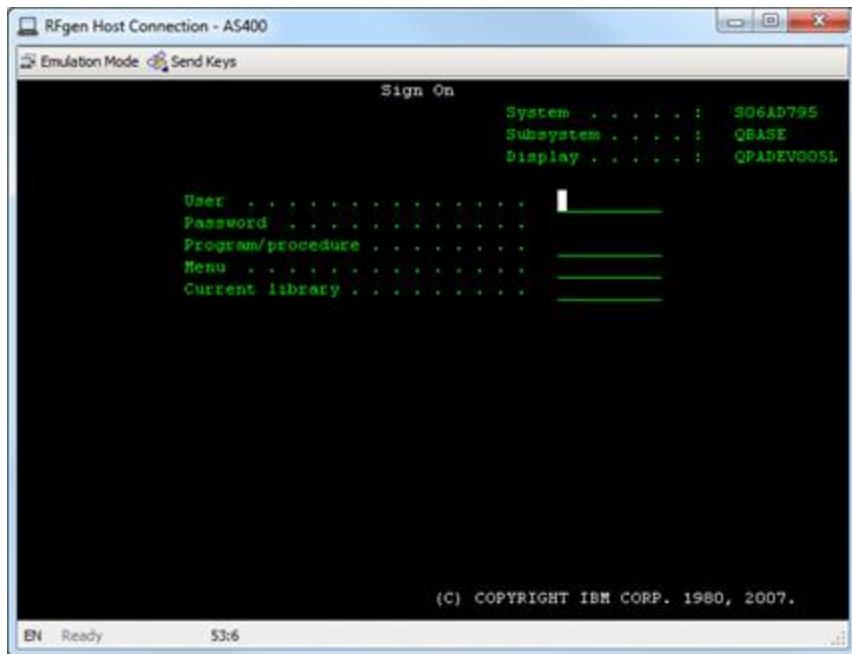
Begin by entering the **Item Name** of the Start Menu macro. This will be the name given to the primary main menu used to log the host system in. Fill in the **Description** field, select Start Menu for the **Screen Type** and **Link To** the name of the screen mapping data connection.

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.

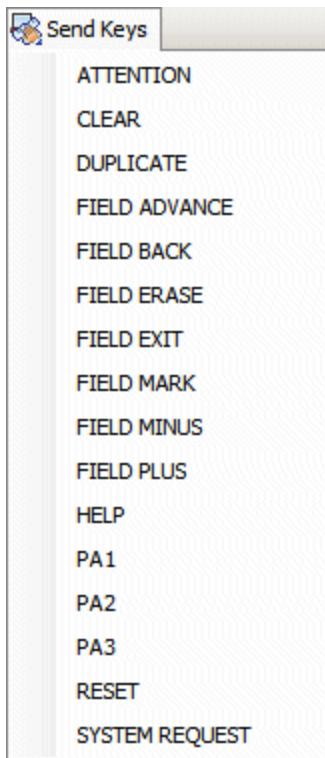
When recording these macros the Host Session window will appear. In the case of this host system, it is an IBM AS/400 connected to RFgen using a TN5250 telnet protocol.



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.

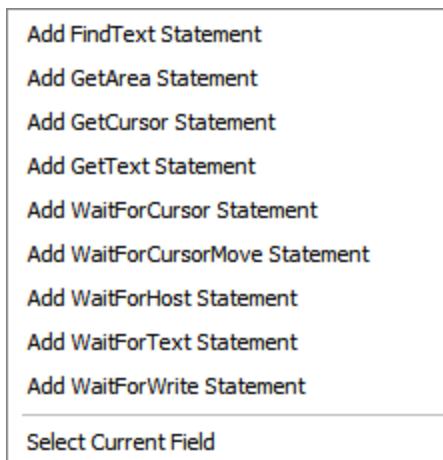


Clicking on a menu item will send the selected key to your host. If you are currently recording a macro, the selected key will become part of that macro.

Start Menu Macro – Record System Sign on

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.



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
▶
<input type="button" value="Save"/> <input type="button" value="Cancel"/>

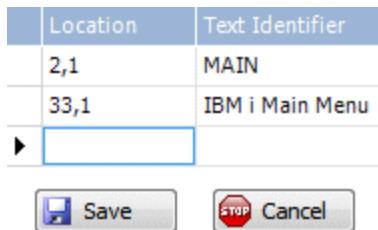
When completed, click Save. Cancel may be clicked at any time to cancel the recording session. An internal macro called **LogOnToMainMenu** is recorded by this step.

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.

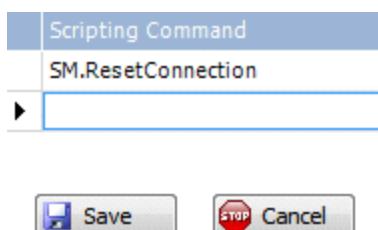
Enter the keystrokes necessary to logoff from the main menu. During the recording phase, the user may right-click on the host system to bring up a menu of additional commands that may be inserted into the script. This was described in the first step.

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:

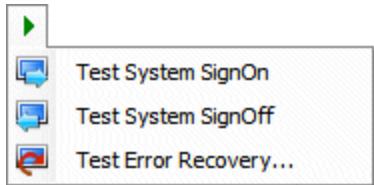
```
Private Function AbortNavigation() As Boolean
    On Error Resume Next
    '
    SM.ResetConnection
```

```
        '
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:

To test the **LogOnToMainMenu** macro, your host screen should first be positioned at your login screen.

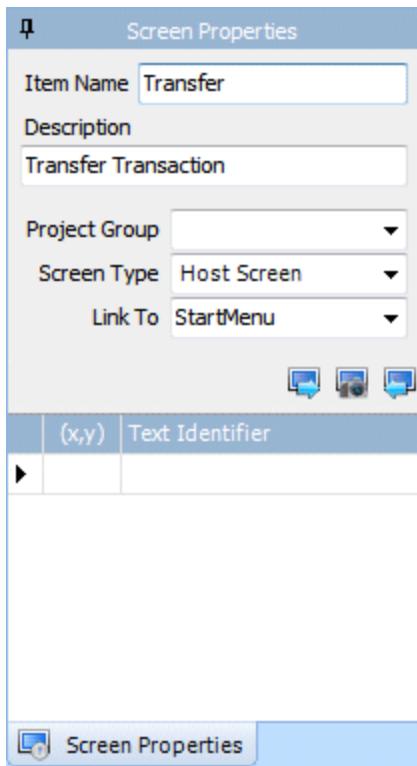
To test the **LogOffFromMainMenu** macro, your host screen should be positioned at your Main menu.

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).

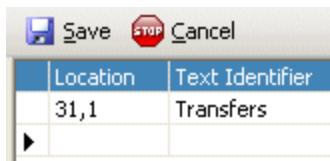
Enter the keystrokes necessary to navigate to the transaction screen. During the recording phase, the user may right-click on the host system to bring up a menu of additional commands that may be inserted into the script. When completed, click 'Save'. 'Cancel' may be clicked at any time to cancel the 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.

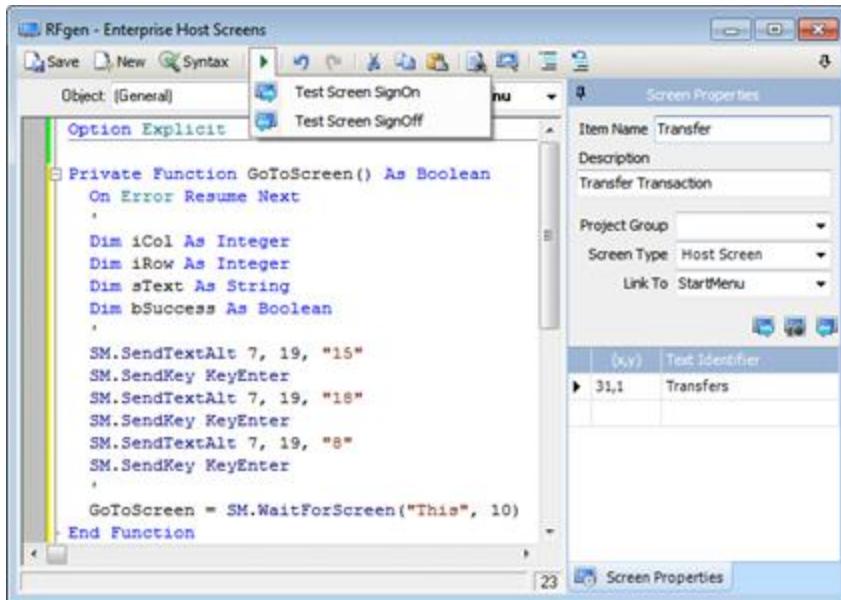
Host Screen Macro – Return to the Main Menu

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.



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:

To test the **GoToScreen** macro, your host screen should first be positioned at your main menu.

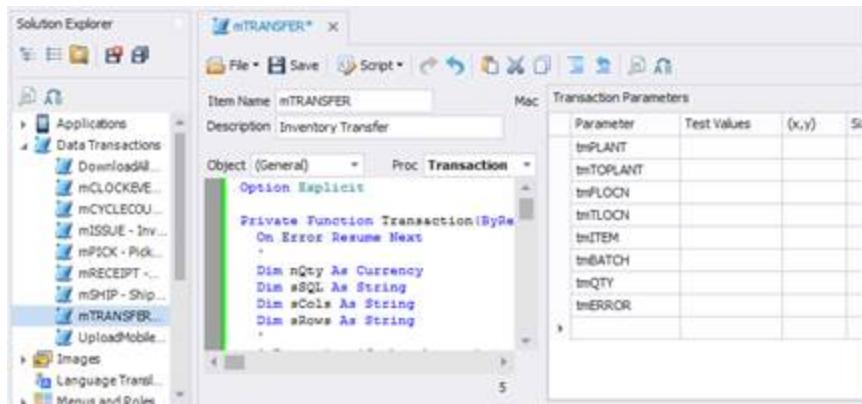
To test the **Transaction Script** macro, your host screen should be positioned at your transaction screen. This macro was created when recording the Enter a Sample Transaction option.

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.

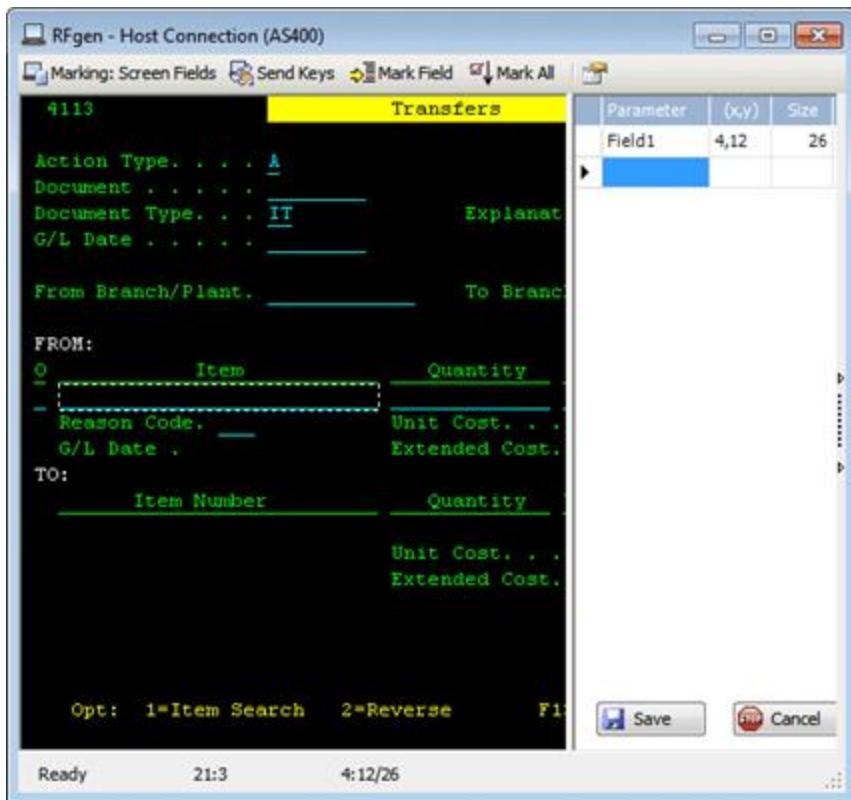
Building a Host Transaction Macro

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.

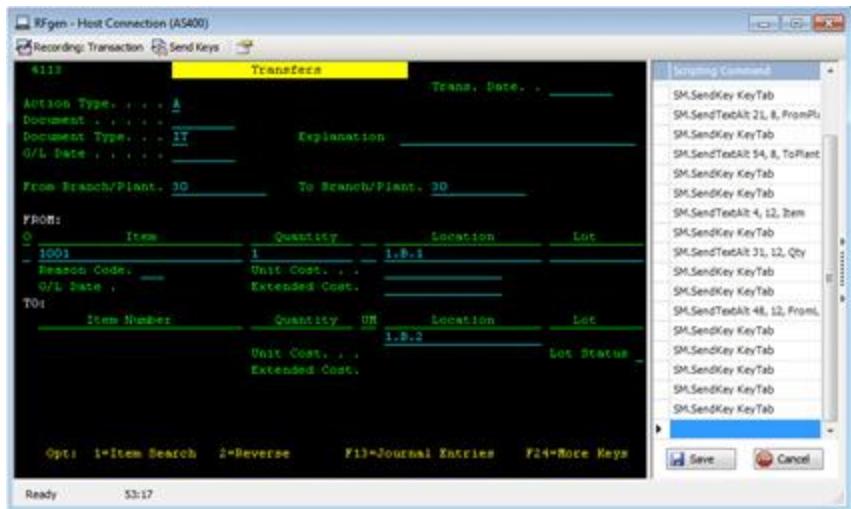


Click the first toolbar button above the parameter grid to navigate the host screen to the correct location.

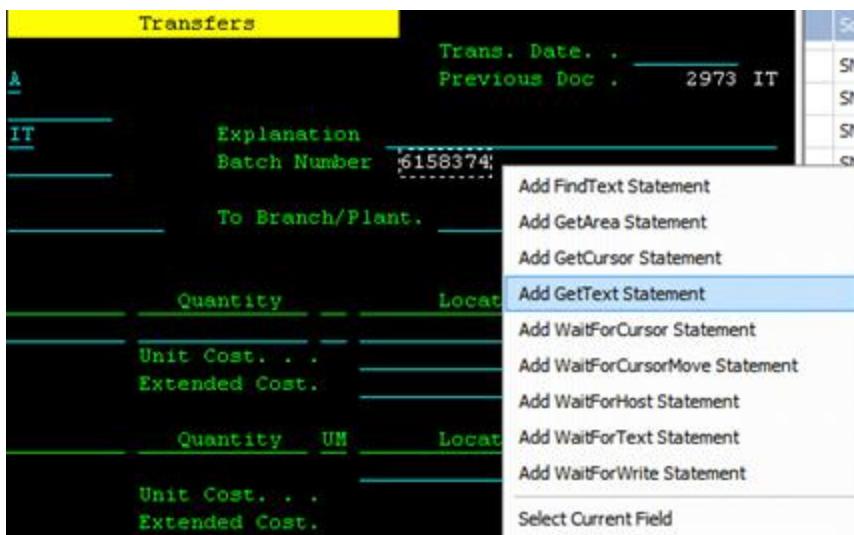
Next click the second toolbar button to mark all the fields on the screen that will be required for data entry.



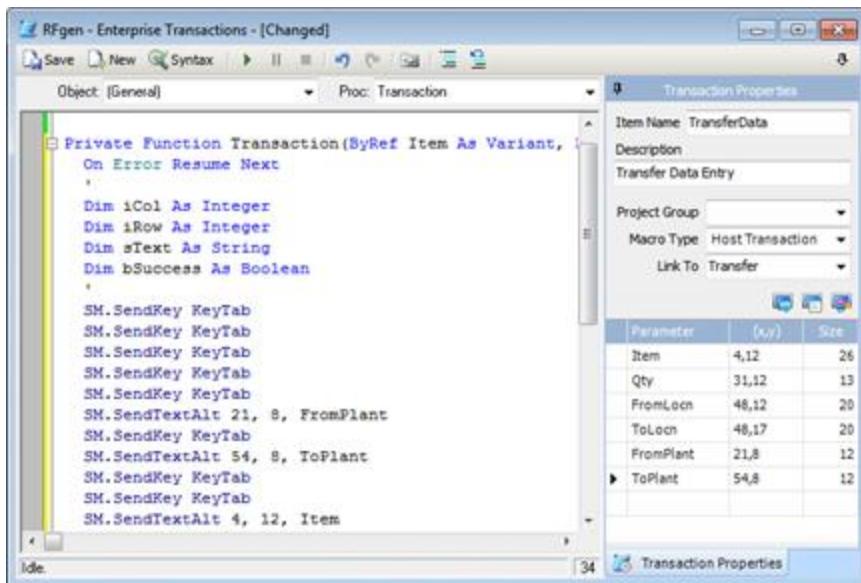
Finally click on the third toolbar button to record a sample transaction.



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.



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.

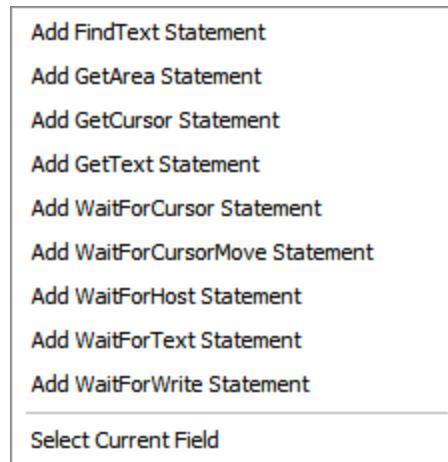
Save and exit the Transaction Macro design screen. The last step is to create an application that links to this macro.

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. 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.

Install RFgen Client Software

The **RFgen** client software enables mobile devices to:

- Communicate with the RFgen server
- Configure the Mobile Client profiles from device settings
- Communication with 3rd party, mobile device management tools

Once the client's software is installed and a connection is established with the server, the user is given the option to download a *Thin Client* or *Mobile Client*. (The actual name may be unique for the customer's organization, but the installation type under the Mobile Development Studio > Mobile Profiles is "Thin" or "Mobile."

Thin means the client can only launch mobile apps and transact data when connected to the server. *Mobile* means the client can host mobile apps and process transactions, and store data tables locally without being connected to the server and/or the network. Mobile mode requires a Batch license to upload data.

After the user selects the client option, the server downloads the mobile profile, and the objects (mobile applications, menus, tables etc.) listed in the profile to the client. Users can then begin working once all solution components are installed and the device authorized.

Although client solutions are designed for centralized deployment from the server to multiple device platforms, the initial install and configuration process varies for each platform.

About RFgen Client Packages

The **RFgen Android Client** installs software to Android devices which enables them to communicate with the server and run applications developed in the Mobile Device Studio.

The RFgen iOS Client installs software to iOS (Apple) devices (but not Macs). This enables RFgen to communicate with the iOS device and run applications developed in the Mobile Device Studio.

The **RFgen Windows Desktop Client** enables communication with the RFgen server, and runs on almost all Windows OS systems.

If you have a tablet that can run a full version of the Windows OS, then it can probably support the client.

The **RFgen Windows CE Client** package installs software on Windows Mobile and Windows CE-based devices and enables communicate with the server and run applications developed in the Mobile Development Studio. In addition to the client package, Windows CE devices also require the installation of **CAB** files.

Thin Client Overview

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.

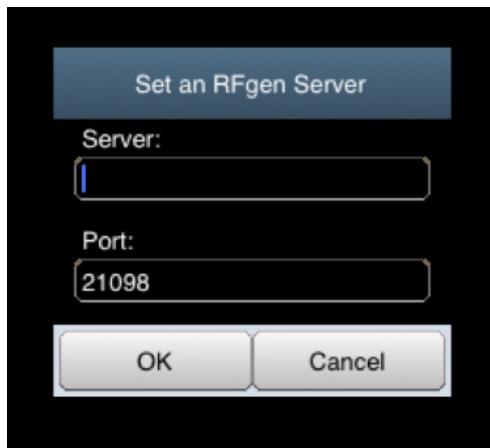
If the wireless device goes out of range of the network, the mobile device screen will appear to stop since the server cannot order the screen to refresh. The client on the mobile device will continue attempts at recon-

necting and will then resend the last piece of data entered. 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.

Connect and Deploy to RFgen Clients

Once you have created your mobile profiles in the Mobile Development Studio, and installed RFgen software to target devices, you enter the RFgen Server hostname or IP address in the RFgen screen. This will request the server to communicate with the client and download the mobile profile of your choosing.



Note: If you do not want to download from the server over a network and your device is Windows Desktop, CE or Android, [click here for alternate methods of installing a mobile profile](#).

Client Network Control Service

On the mobile device there is an icon  that represents the CNC (Client Network Control) 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.

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, CNC 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.

The CNC service is installed when the Windows Desktop Client.exe, Windows CE Client.exe, or RFgen Client for iOS or Android is installed.

If needed, the CNC can also be packed and installed separately to a Windows Mobile/Windows CE device. Refer to the Device Management > Mobile Device Installation Utility in the Mobile Development Studio for more details.

Third-Party Mobile Device Management Tools

RFgen supports third-party MDM tools for configuring the RFgen iOS client.

For more details refer to the **RFgen 5.1 Installation and Upgrade Procedures Guide**, which is available from your Program File\RFgen51\Documentation folder on Windows OS up to Windows 8, or is listed from the Apps on Windows 2012 and 2016 servers.

Auto Upgrade of 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 CNC service so a process outside the client process on the device can execute the commands to replace the client application.

Finally, when a client detects that it is necessary to upgrade the user may be given a choice to perform the upgrade.

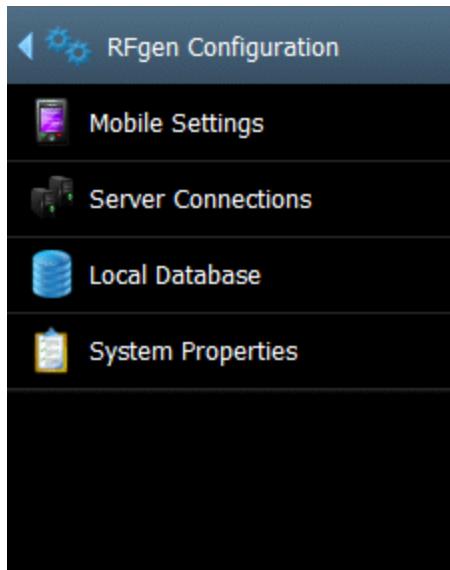
Windows Mobile Configuration

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 51 program from the list.

RFgen Configuration



There are multiple configuration pages available. The first one is the Mobile Settings page.

Mobile Settings

User Mobile Settings to change how your device starts up, receives updates, displays your application screens, change the language, scans barcodes (using the camera in your device) and many other settings. Some values are preset by the developer while others can be changed by the user.



Client Startup Mode

For Mobile Installation Types, this option determines if the Mobile client will startup in a connected state or off-line (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.

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

Remote Debugger

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.

Reprovision the Device

If checked, the device will clear its existing profile so the user can select a new or updated mobile profile when the device connects with the server. If unchecked, the existing profile is retained and the device bypasses the option to download a new profile when it connects with the server.

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.

Append Return After Scan

If "Return After Scan" is enabled, after a barcode is scanned, the cursor will go to the next field. If its not enabled, the cursor will remain in the same location and the user will need to tap the Return/ Enter key to continue to the next field. Older versions of the RFgen client will display one checkbox. (Check adds a return; No check does not add a return). The newer versions of the RFgen client may display one of these four options:

None - Return is not appended at the end of a scan transaction. (Return is disabled.)

Camera - A Return is appended to a scanned transaction. (Return is enabled.)

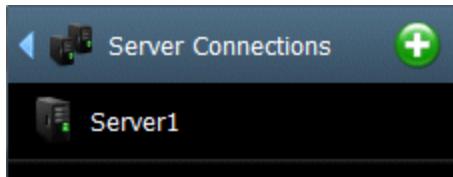
Native Wedge - Two Return characters are appended to a scanned transaction. (Return is enabled.)

Camera & Native Wedge - Depending on which device is used, one or more return characters are appended to a scanned transaction. (Return is enabled.)

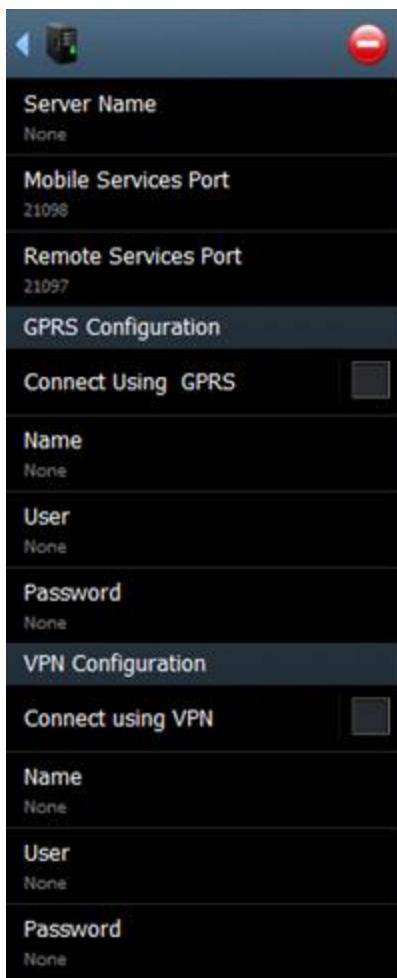
Maximum Picture Width and **Maximum Picture Height** are used when taking advantage of the device's camera. If a picture is taken and its size is greater than the maximum allowed, the image will be resized down to the max width or height, whichever comes last so that the aspect ratio is not changed.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

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 client can connect to any server. Adding the server's name 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 uses to share an application with the client when its connected to the server.

Remote Services Port

This is the port number the server is using for passing data between the client and server, when the client is off-line.

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

If a database table was included in the CAB file of a Windows CE profile (Mobile Profile), then this option should show the type of database that was used when the Table was added to the profile.

Provider

This field specifies which database application 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.

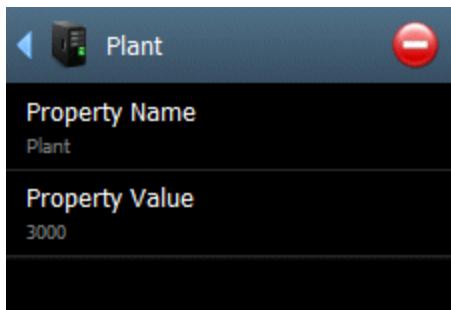
User

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



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 Windows CE 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 con-

figuration 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.

Activating Android or iOS Mobile Clients

On Mobile Clients, the RFgen administrator can deploy authorization certificates to RFgen clients on Android or iOS systems through the server **Device Management > Device Authorizations** feature.

Deployment of the certificate occurs after the device has received a Mobile Profile and the RFgen Administrator has setup the certificate to be deployed to a specific device (from the server).

On the device, if you click on the Menu Strip button (also called the System Menu), the "Activate Button" should display.

Clicking on "Activate" is similar to downloading and installing the certificate. Once its successfully installed, the Mobile Device setting will change to "Authorized."

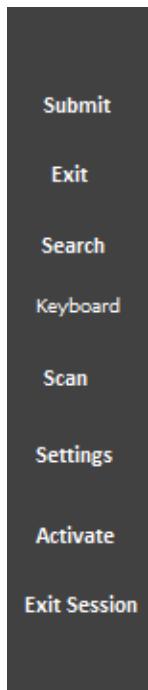
To authorize a Batch Client

If you are using a mobile client in Batch (offline mode or batch mode), a batch license must be installed to the device and then activated/authorized.

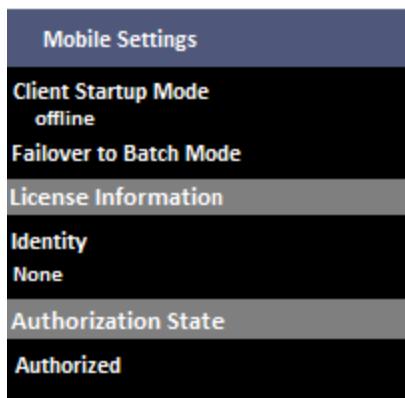
Before you can download a license (via a deployment of the mobile profile), a connection to the server must be established. The connection may require the RFgen Administrator to authorize the device connection from the server.

Android or iOS: Once the mobile profile has been deployed / installed to the client, you activate the license via the "Activate" command from the menu strip.

1. Connect to the RFgen server.
2. When your mobile application Login displays, click on the menu button to display the menu strip.
3. Open the Menu strip, and click on **Activate**. This button is only present when the certificate is present for a download.



You can also verify if the device is authorized by reviewing the device's RFgen Configuration > Mobile Settings > License Information: Authorization State.



Mobile Unity Platform Console

The **Mobile Unity Platform Console** is a graphical interface for administration of your server at a glance. It also allows you to:

- Identify the version of RFgen installed on the server.
- Check the running status of server services. It tells you if your services are up or down, and whether the server is in Maintenance Mode.
- Stop or start the server service and allow or disallow users to connect (Maintenance Mode), and broadcast a message to all connected users.
- Configure the server and setup connections to database systems and enterprise (ERP systems).
- View reports (application event logs) and export them to Excel.
- View server authorization status, number of licensed uses. If your server is not yet activated, it can be activated through here. (See Web Authorization.)
- View which mobile devices are connecting with your server and controls which ones are authorized to connect.

When started, the Server enables multiple communication sessions between your server and your remote devices (up to the number of authorized users).

System Environment > Configuration > Application Services may be used to specify a different port. The service will administer to all types of clients (GUI-based devices, XML, Vocollect, etc.) simultaneously.

Refer to Configuring Your System Environment for more details.

Accessing the RFgen Server Console (Mobile Unity Platform Console) and Services

If you want to display the RFgen server console from your Microsoft Windows System Tray, follow these steps.

For this procedure, first add your Mobile Unity Platform console icon to the Windows System Tray. For details refer to your Microsoft Windows System Help.

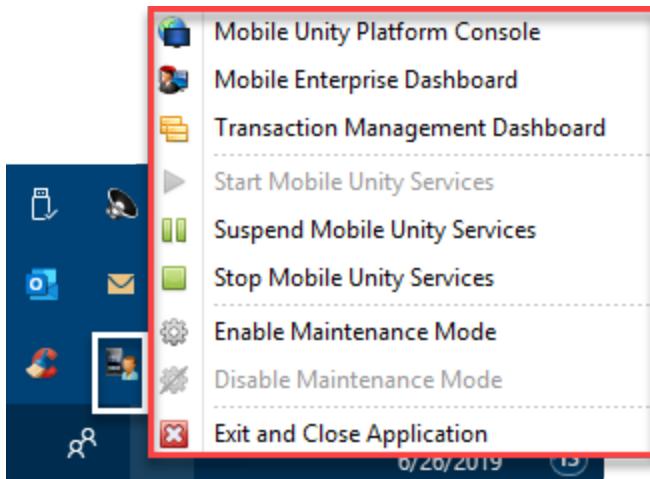


1. Click on the Mobile Services Manager icon  in your Windows system tray. Note that there are two modes the server runs in -- Enabled Maintenance Mode and Disabled Maintenance Mode.

If the Mobile Services Manager is in Maintenance Mode it displays this icon: 

If the Mobile Services Manager is not in Maintenance Mode, the Services Manager does not change.

2. The Mobile Services Manager Menu icon displays.



3. Click on the Mobile Unity Platform Console.
4. The RFgen Mobile Unity Platform Services Console displays.



To Activate (Authorize) the Server

THE RFGEN SERVER MUST BE AUTHORIZED TO RUN

If your RFgen Server is licensed, but unauthorized by RFgen, the application services are suspended (paused). To activate it, you will need a system authorization certificate installed to your server. This certificate can be obtained several ways:

- [Email 'support@rfgen.com'](mailto:support@rfgen.com)
- [Phone request to RFgen Support](#)
- [RFgen Web Authorization process](#)

Activation via the web

To authorize the service for permanent operation using the web, follow these steps if the server has never been authorized, or was once authorized and the System Id has not changed. If your server has been authorized before or your System ID has changed, then contact RFgen Support for assistance.

From the **Mobile Unity Platform Console > System Authorization**, click on the System Certificates.

The RFgen – Service Authorization window displays.



1. Enter your **Customer Id** and **Serial Number** and click the **Web Authorization** button.
2. For example:
Customer ID
7382 ;Serial Number KXI8N-9384
3. Your system ID is filled in automatically by the Mobile Unity Platform Console.
4. Click on the Web Authorization button.
5. The window flashes (nothing else displays.) When done, the License Status changes to "Permanent Authorization."

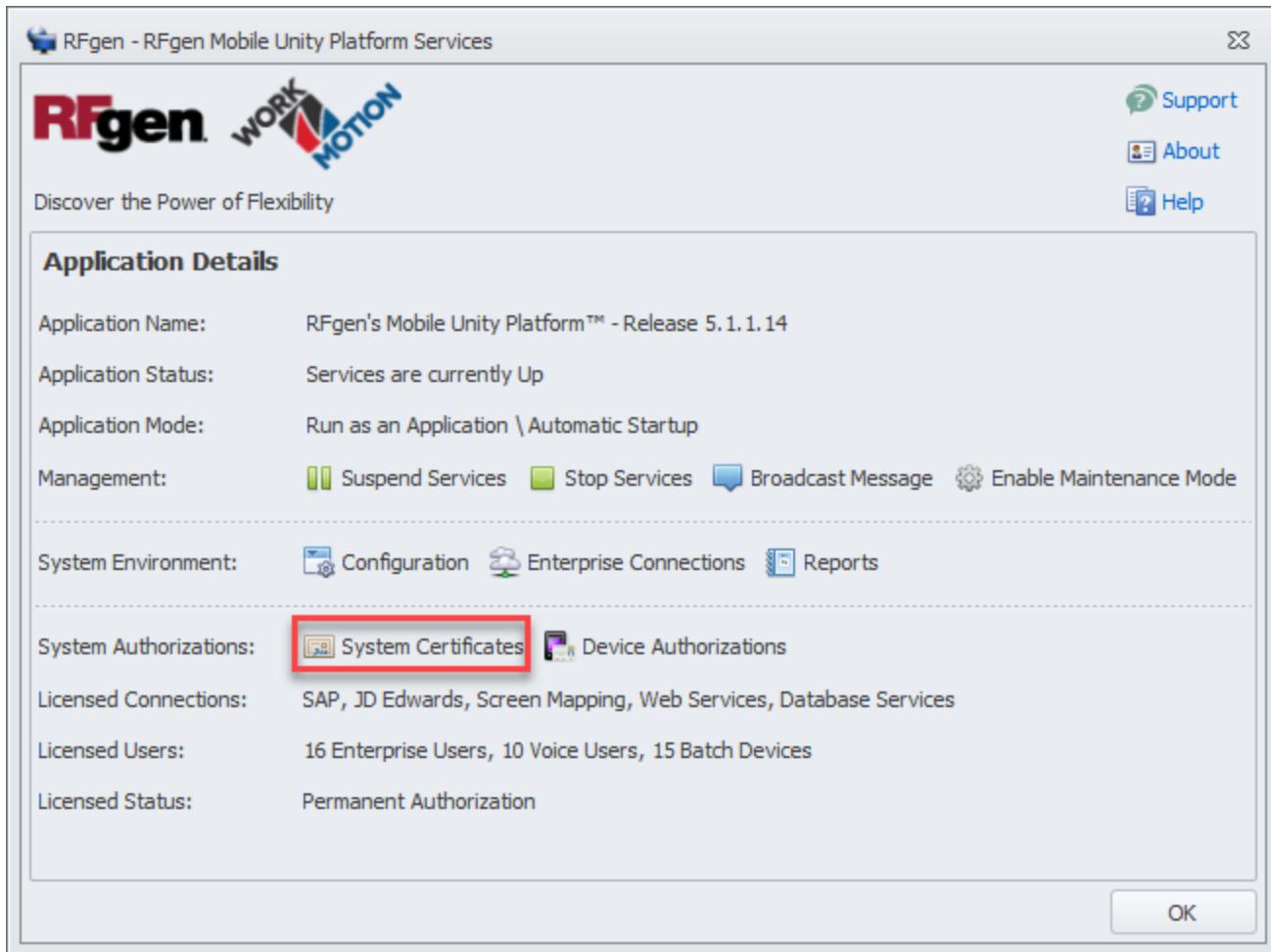
System Authorizations:	<input type="checkbox"/> System Certificates <input checked="" type="checkbox"/> Device Authorizations
Licensed Connections:	SAP, JD Edwards, Screen Mapping, Web Services, Database Services
Licensed Users:	16 Enterprise Users, 10 Voice Users, 15 Batch Devices
Licensed Status:	Permanent Authorization

System Authorizations:	<input type="checkbox"/> System Certificates <input checked="" type="checkbox"/> Device Authorizations
Licensed Connections:	SAP, JD Edwards, Screen Mapping, Web Services, Database Services
Licensed Users:	16 Enterprise Users, 10 Voice Users, 15 Batch Devices
Licensed Status:	Permanent Authorization

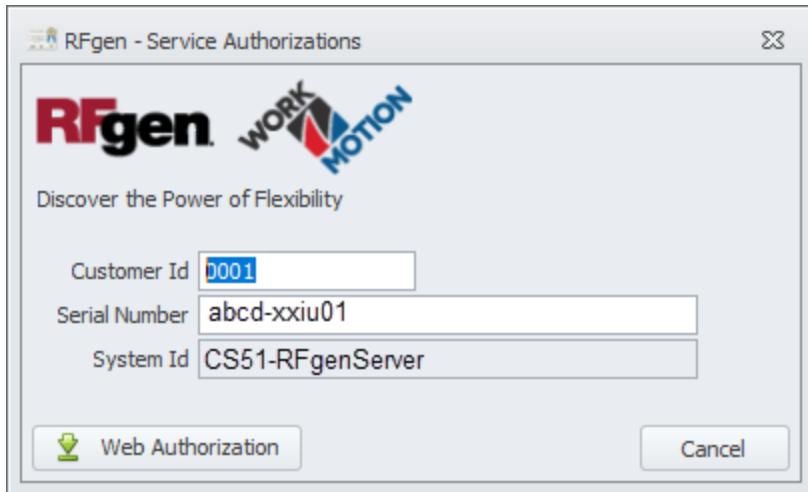
System Certificates

Once your system authorized by RFgen Datamax Software, your server services will be enabled and allowed to connect with Thin clients.

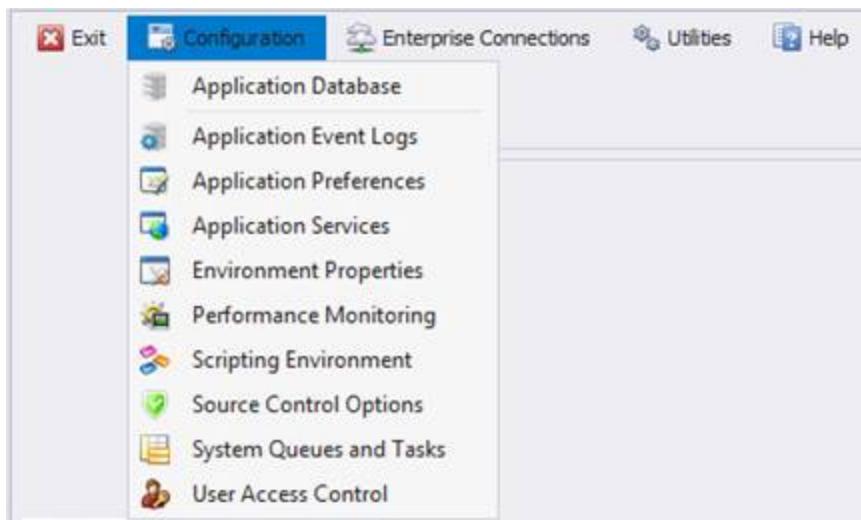
The RFgen server console displays your system certificate when you click on the **System Certificates** link in the RFgen Mobile Unity Platform Console.



System Certification Example:



Configuring the Server



The Configuration Menu functions are as follows:

The **Application Database** configures the database storing all the solution objects.

The **Application Event Logs** database configuration is used to log events and reset connections for specific databases.

The **Application Preferences** is for user interface themes and locale, default design mode, and scripting.

The **Application Services** screen include port configurations, load balancing, server run mode, service credentials, security and controller failover monitoring.

The **Environment Properties** settings including system options, timeout values, Pre/Post-amble scanner defaults, embedded graphical menu options, function key options, system properties and the device authorization scheme.

The **Performance Monitoring** sets thresholds for backend communication for debugging purposes.

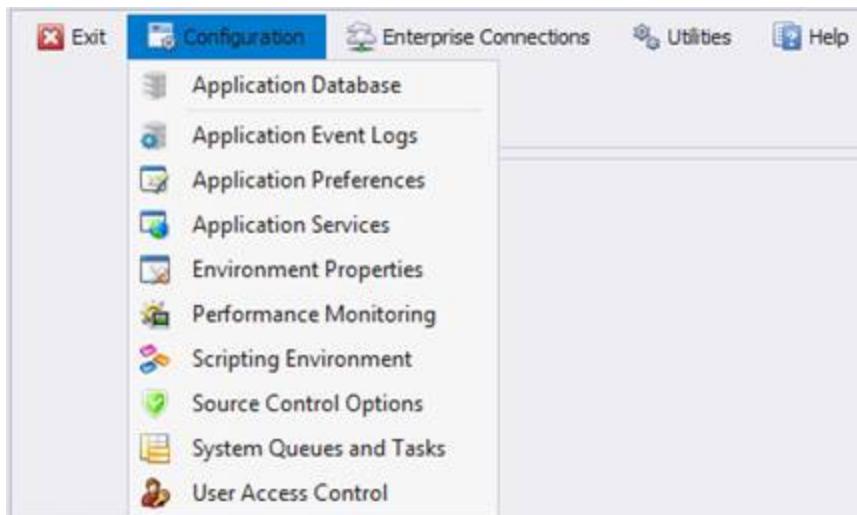
The **Scripting Environment** settings to allow direct access to Active Directory Objects (ADO) and XML language extension parameters and have them globally loaded into BAS files.

The **Source Control Options** allow developers to use a third-party source control product if its plug-in is supported.

The **System Queues and Tasks** configure all queues and timed event macro executions.

The **User Access Control** console allows you to authenticate connections between the RFgen Server and the **Mobile Platform Unity Management Console**, and **User Management Console**.

Configuring the Server



The Configuration Menu functions are as follows:

The **Application Database** configures the database storing all the solution objects.

The **Application Event Logs** database configuration is used to log events and reset connections for specific databases.

The **Application Preferences** is for user interface themes and locale, default design mode, and scripting.

The **Application Services** screen include port configurations, load balancing, server run mode, service credentials, security and controller failover monitoring.

The **Environment Properties** settings including system options, timeout values, Pre/Post-amble scanner defaults, embedded graphical menu options, function key options, system properties and the device authorization scheme.

The **Performance Monitoring** sets thresholds for backend communication for debugging purposes.

The **Scripting Environment** settings to allow direct access to Active Directory Objects (ADO) and XML language extension parameters and have them globally loaded into BAS files.

The **Source Control Options** allow developers to use a third-party source control product if its plug-in is supported.

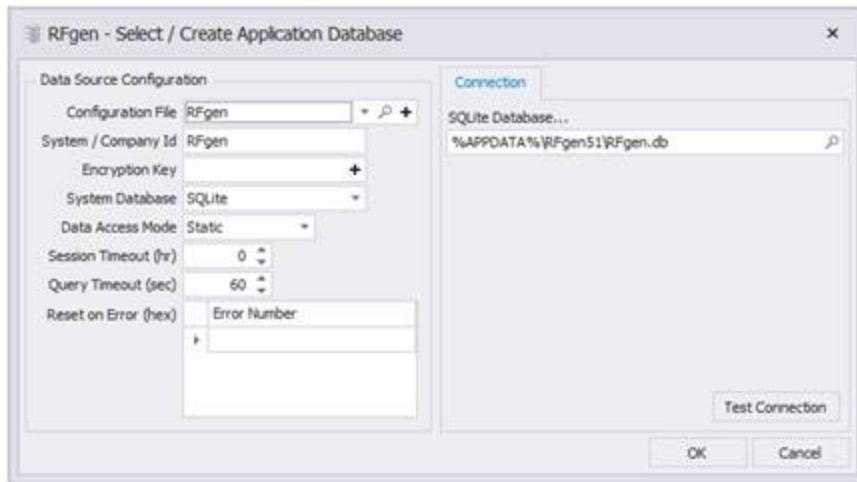
The **System Queues and Tasks** configure all queues and timed event macro executions.

The **User Access Control** console allows you to authenticate connections between the RFgen Server and the **Mobile Platform Unity Management Console**, and **User Management Console**.

Configuring the RFgen Application Database

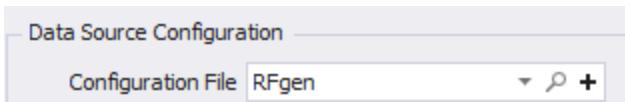
In order to provide a database for storing and maintaining your RFgen Mobile Applications (which help run your Transactions), you need to connect to a database application/server/system to your RFgen server/system.

From the Mobile Development Studio or Mobile Unity Platform Console: Click on **Configuration > Application Database**.



Data Source Configuration Values

By default, a Configuration File called 'RFgen.rfc', defines the profile of the solution database, as shown below.



If you need to change the rfc file or select a different rfc file, you can use the list, search or plus (+) icons to browse to the %APPDATA%\ProgramData\RFgen51 folder.

An '**Encryption Key**' entry provides users the ability to encrypt their Application Database. This feature 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 **Encryption Key** textbox, and click the + 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.

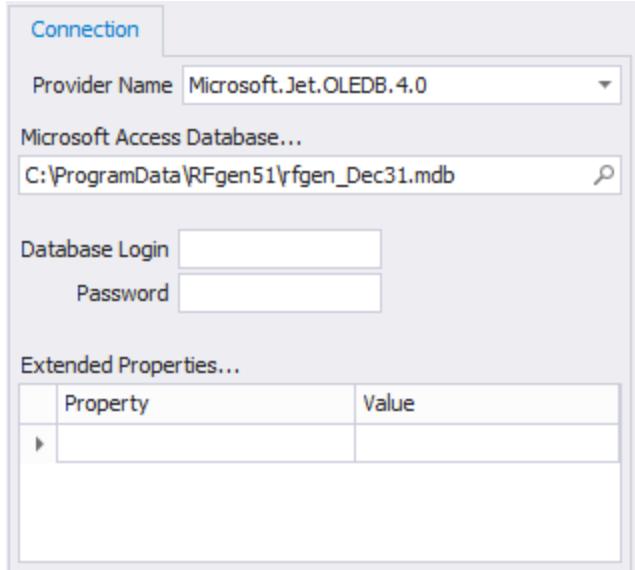
Data Access Mode sets the cursor to either Static or Dynamic when retrieving data from the database. Usually, Static is best because it is fast and safe. However, if you have a database like Pervasive that will actually make a copy of the data from the database system to the RFgen system when using a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

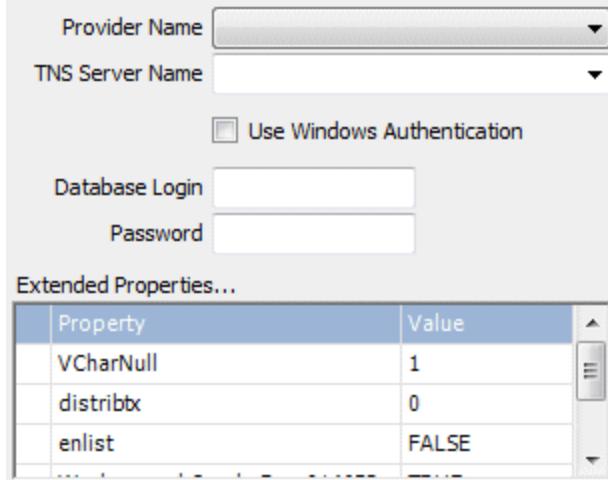
Reset on Error is a list of hex values that if returned by the ODBC driver will cause a reset of the connection. The process for adding a value is to first get the error number from the error log. Example: the error log shows -21456327. Use the Windows calculator in Programmer mode, select Dec and Dword 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.



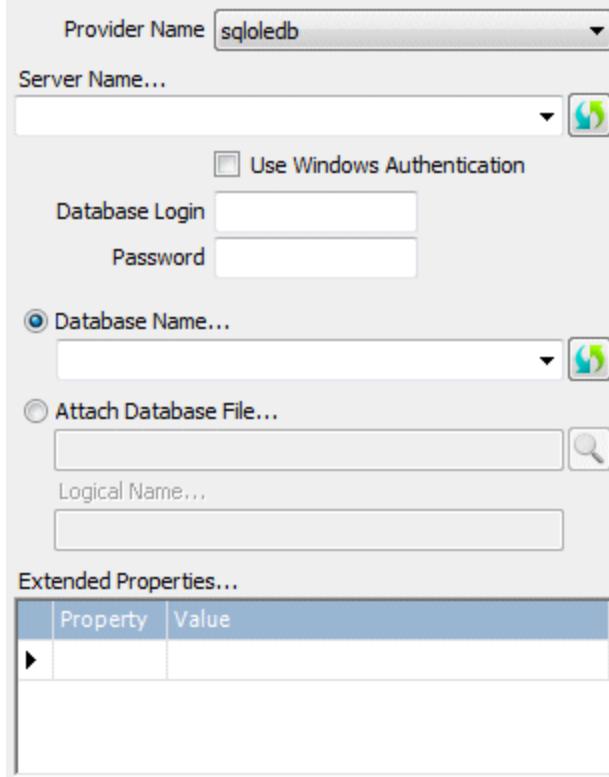
For an **Access** database, select the appropriate Provider Name for the type of system (32 bit or 64 bit). The path, login, password and extended properties are then used to make the connection.

In the case of Access most of these fields are not necessary.

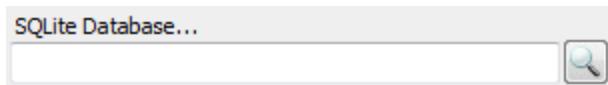


In the case of **Oracle** ODBC is not used but the TNS Server Name points to the Oracle server. Also specify the Provider Name and review the Extended Properties for accuracy. The Use Windows Authentication option will

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



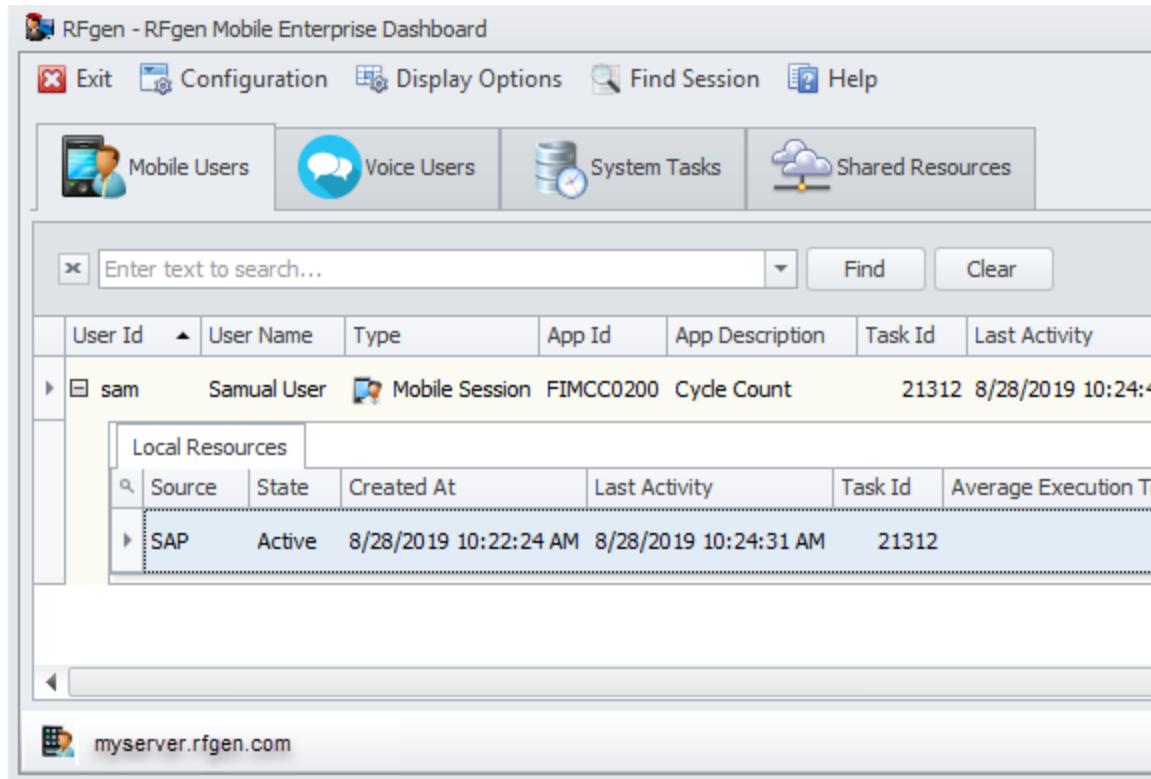
For **SQL** Server specify the Provider Name, Server Name and Database Name. The Use Windows Authentication option will take advantage of the Active Directory when connecting to the database. If you want to connect directly to the MDF file itself, specify the Attach Database File option and locate the 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.

Mobile Enterprise Dashboard

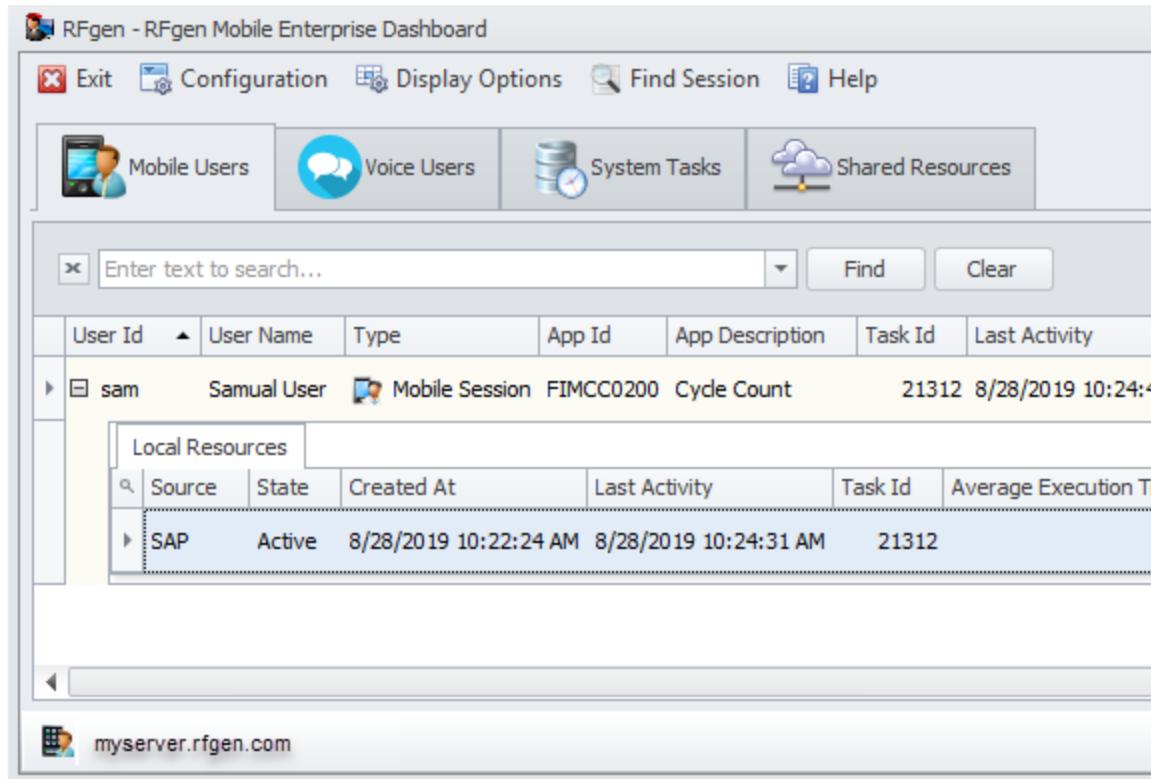


The Mobile Enterprise Dashboard allows you to view and manage the remote sessions running under the server. You can change your views to monitor different types of sessions. For example: Mobile Clients, Voice Users, System Tasks or Shared Resources.

As devices log in, they are displayed in the dashboard.

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.

Mobile Enterprise Dashboard Menu



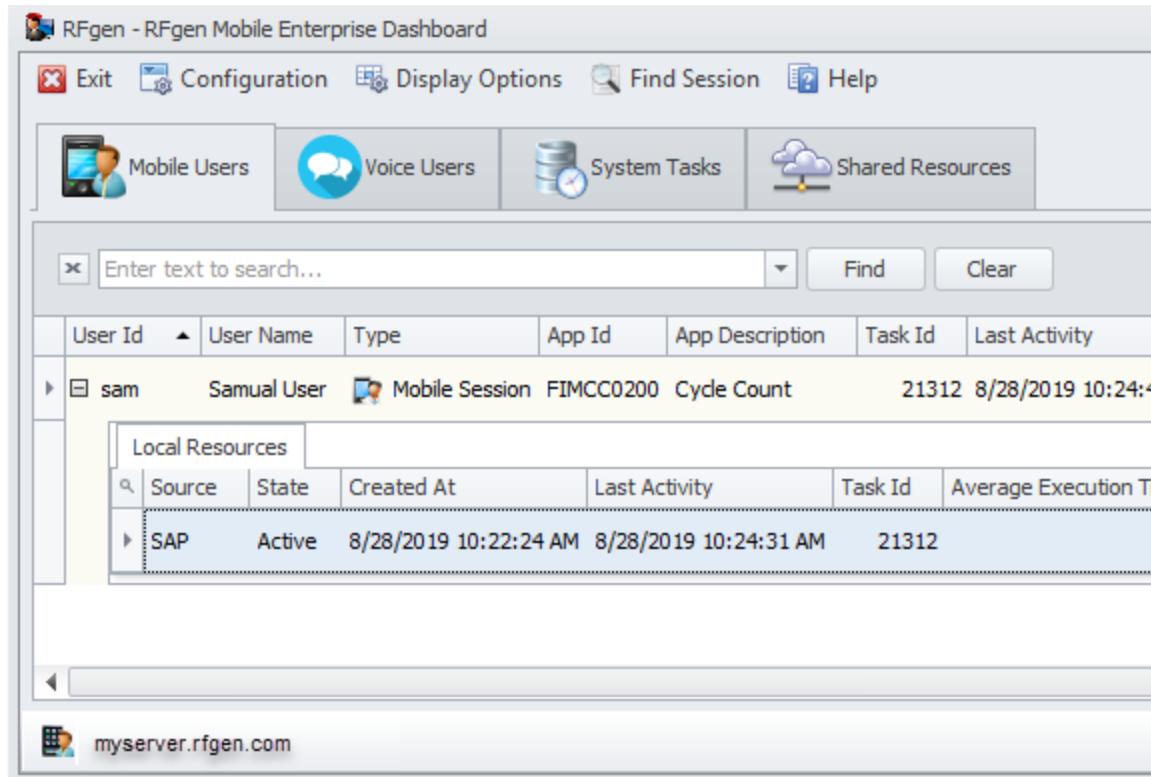
Configuration has two settings: *Application Preferences* and *Select Server Group*. The *Application Preferences* allows you to change the Windows Theme for the dashboard. *Select Server Group* stores a list of discovered RFgen servers. Once a connection is established, the selected server will display at the bottom of the Dashboard. If the connection is invalid, a red (-) will appear.

Display Options lists the column headers used to view user connection information in the Dashboard area.

Find Session allows you to find a specific session when you many sessions going on in the dashboard. For a description of each item in this menu, refer to **To Configure Your Views**.

Help menu allows you to access the topics from the RFgen Manual, obtain information on how to access Support, and view version and platform information about the Dashboard.

Overview of Dashboard Views



The **Mobile Enterprise Dashboard**, enables you to monitor and manage remote sessions running under the server. This includes:

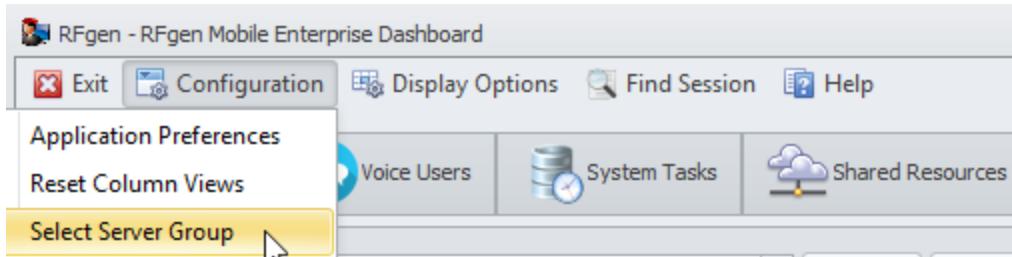
- **Mobile User** sessions - shows the session by user name/ID once the user logs in.
- **Voice User** sessions for users of voice applications.
- **System Tasks** sessions by the system or data source (i.e. SAP, Oracle etc.)
- **Shared Resources** for viewing pooled sessions (where you have license pooling setup for an ERP)

Through this dashboard, the administrator can perform tasks such as joining a session, send messages to a user, and suspending or terminating their session. Specifics about a session can also be collected. For example, if you want to see how long a task is taking to execute, you can look for this information on the System Tasks tab.

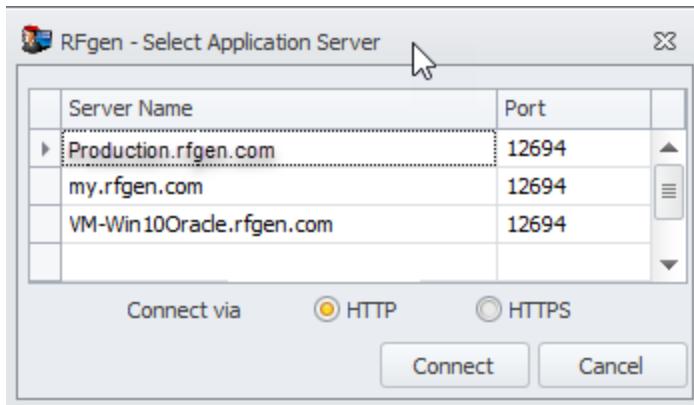
Note: When the session ends the display disappears.

RFgen Server Connections

1. From the RFgen Mobile Enterprise Dashboard, select: **Configuration > Select Server Group**.



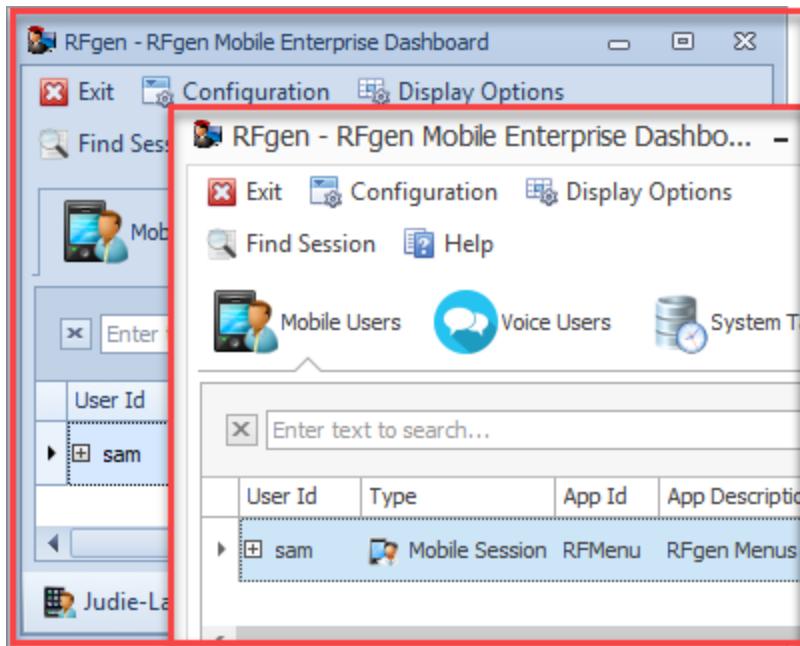
2. The **Select Application Server** screen displays.



Select the server as the source for viewing information in the dashboard.

3. Select the connection type (HTTP versus HTTPS).
4. Enter your credentials if required to access the server.
Press Connect.
5. The server/server group you connected to displays in the lower left corner of the dashboard.

To Set the Language in Your Dashboard

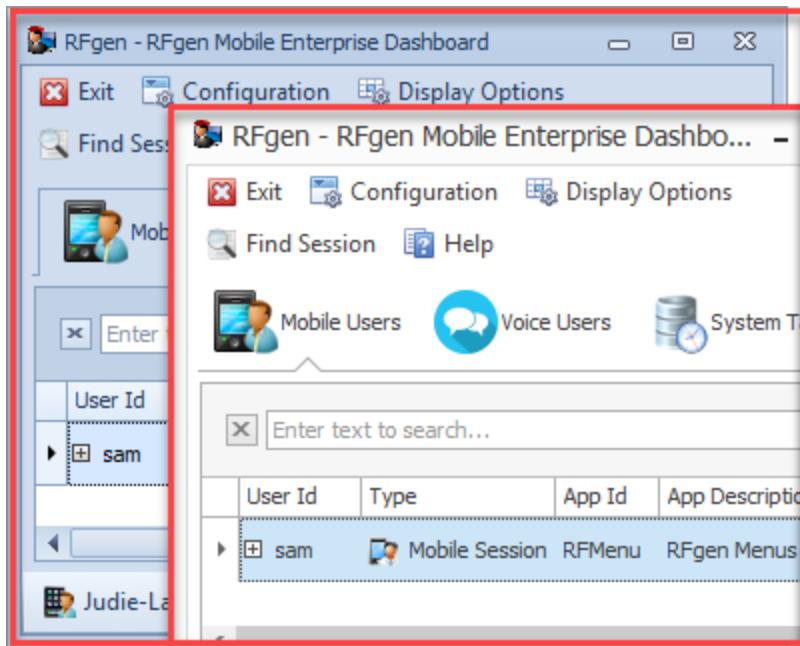


You can change the local\language your RFgen Mobile Enterprise Dashboard from English to any of the languages listed in the Application Interface drop down menu. This will only affect your view of the dashboard and will have no affect on the Mobile Applications viewed by end users.

The languages supported by the RFgen are: English, Arabic, Chinese, French, Japanese, and Spanish.

1. From the RFgen Mobile Enterprise Dashboard, click on Configuration > Application Preferences.
2. Select the language you want to use from the Application Interface drop down menu. For example, English to Arabic. You can also set the Default Locale (i.e. specify the language used in a region such as English (United States)).
3. Click **OK**. The screen will shutdown immediately. (Or, you may need to manually restart to make the changes take place.)

To Change Your Dashboard's Appearance



You can change the general look and feel of your RFgen Mobile Enterprise Dashboard by changing its skin/Application Skin. This will only affect your view of the dashboard and will have no affect on the Mobile Applications viewed by end users.

1. From the RFgen Mobile Enterprise Dashboard, click on Configuration > Application Preferences.
2. Select the Application Theme from the drop down menu.
3. Click **OK**. The screen changes immediately to the chosen theme.

To Configure Your Views

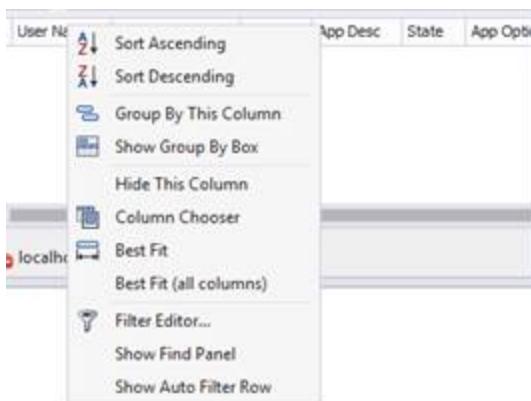
Select the icon for which you want to design a view. For example, select the **Mobile Users** icon.

In the display panel area, you can choose to:

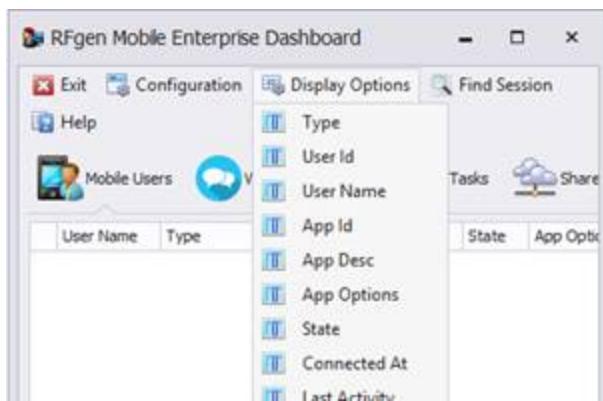
- a. Rearrange the order of columns – by selecting then dragging it to its new location
- b. Hide a column – by selecting this from the Right-Click Edit menu
- c. Add a column from the **Display Options** menu
- d. Sort and filter columns using the options from the Right-click menu.



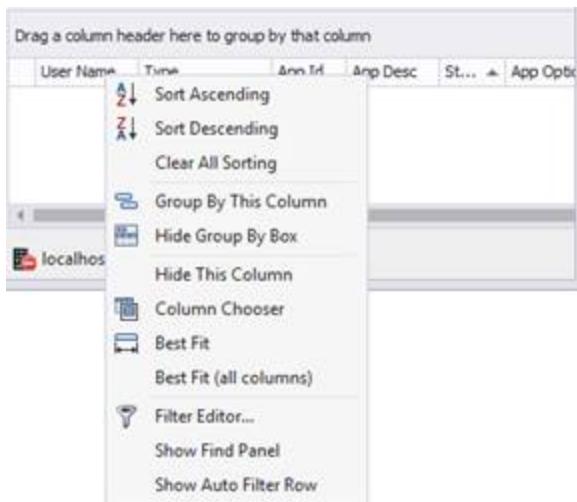
Example a. Rearranging columns



Example b: Hide or Show a Column



Example c: Add a column from Display Options menu



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.

User Id	Type	User Name	App Id	App Desc	State	Task Id	Co
▶ + sam	Mobile Device	Demo	RFMenu	RFgen Menus	Active	3416	11

This view appears when the dashboard is first started. As data entry devices log in, each appears in its own row.

User Id	Type	User Name	App Id	App Desc	State	Task Id
▶ - sam	Mobile Device	Demo	RFMenu	RFgen Menus	Active	3416
Local Resources						
▶ - JDE_DEMO	Active	11/29/2017 5:20:37 PM	11/29/2017 5:20:37 PM	3416	M	

The pointer indicates which row is selected.

The “+” icon allows you to expand the details for the selected row (logged-in device). The “-“ will hide the details for the selected row.

User Id	Type	User Name	App Id
▶ - sam	Mobile Device	Demo	RFMenu

To bring up the filter icon, click on the background of a column header.

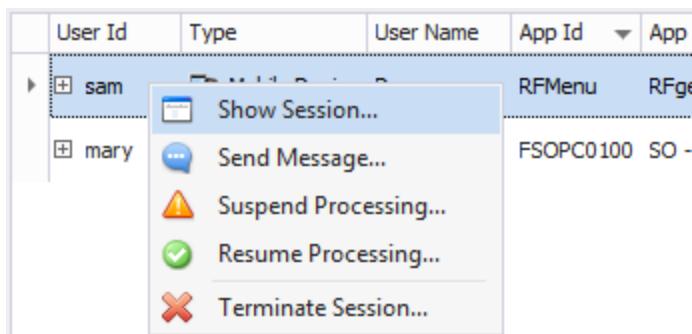
User List						
User Id	Type	User Name	App Id	App Desc	State	App
[+]	(Custom)	mary	RFMenu	RFgen Menus	Active	
[+]		mary				
		sam				

You can also view a summary of the values for a given column by clicking on the filter icon at the top of a column heading.

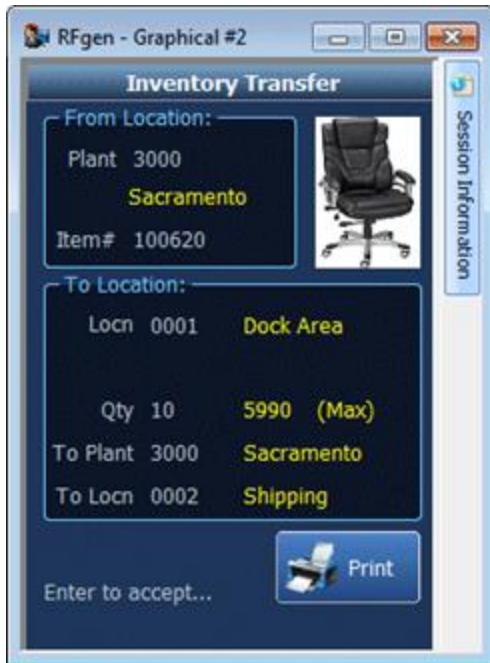
See [Display Options](#) for more details on these headings.

To Monitor and Interact with an Active Client

1. To monitor or interact with an active client session, simply right-click on the row of the device show session you want to monitor. A selection menu will appear.



2. Click on **Show Session...**. The selected client device screen window will appear on your screen.



3. While the session window is open, all activity for the device will appear in the window.

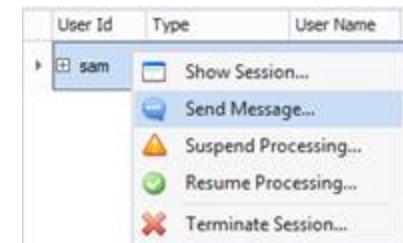
To take control of the session, click inside the screen display area and interact with the prompts. The same screen will appear on your screen. At this point the user on the client will see your actions.

To end your session, click on the "X" in upper right corner to end your remote session. The client session will continue to run unless you used the "Suspend Processing..." or "Terminate Session" commands from the right click menu.

To Broadcast a Message

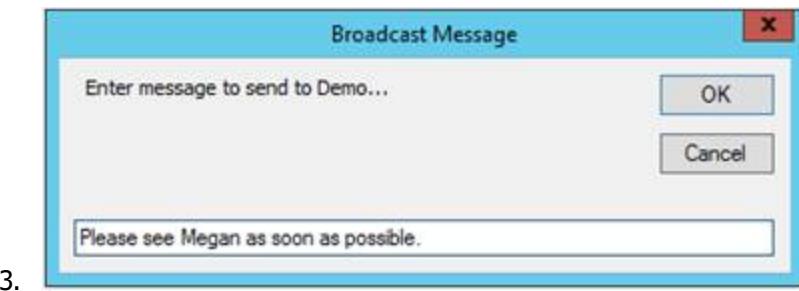
Send Message allows you to send a message to the device user from the Mobile Enterprise Dashboard.

1. To send a message to a specific client, right-click the row of a device show session and select **Send Message...** from the menu.



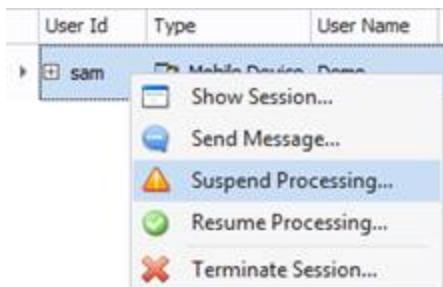
2.

A message box will display. Enter your message and click **OK**.



The client will get a pop-up message on their screen.

To Suspend or Terminate a Session

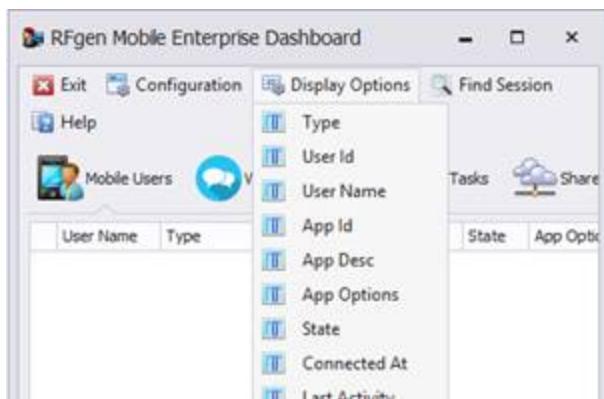


To temporarily stop this session from collecting data, choose **Suspend Processing** from the menu.

To terminate this session, click **Terminate Session**. This action will terminate the communication session for the remote device.

Display Options

The Display Options menu lists the column headings that are used in the Mobile Users, Voice Users, System Tasks, and Shared Resources tabs. To change the column headings, select **Display Options** menu and then right-click on the column heading you want to add or change. The list below describes each header type.



The Display Options details are:

Type – Type of connection. For example, a Windows Desktop connection will show up as a Mobile Device.

User ID – the user ID or operator who logged in.

User Name – The full name of the logged in user

App Id – shows the menu or application screen name currently being viewed by the user.

App Desc – the description of the current form.

App Options – any passed in parameters to the current form from the menu

State – shows either Disabled or Active depending on the suspend status of the client connection.

Connected At – shows when the connection was established.

Last Activity – shows when the very last keystroke was made by the user.

Server Name – Is the name you assign the RFgen server or its IP.

Task Id – is the process identifier of the client session executable that can be located in the processes list of the Task Manager.

IP Address – assigned IP address of the device.

GUID – 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.

Platform – Describes the platform of the RFgen Client, which can be: Windows Desktop, Windows CE or Mobile, Android, or iOS client.

Size – the size of the screen display used by the client's application.

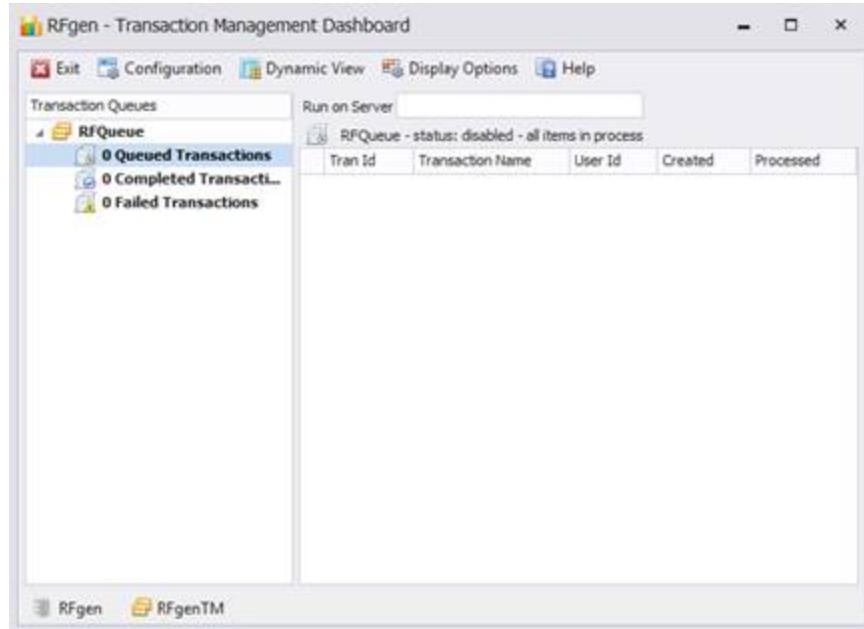
Theme – the mobile theme that is used by the client's application.

Locale – The Microsoft Locale ID value used by the client's application. For example, 1033 is English - United States.

RTL – Right-to-Left (versus Left-to-Right) setting – which is the orientation of the application for the locale of the client. For example, English is read from right to left.

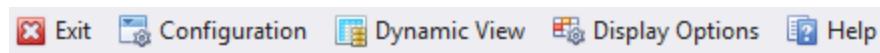
Transaction Management Dashboard

The Transaction Management Dashboard is used to manage 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 is offline or not available, or the batch client is offline, and cannot connect and sync up with the host server); "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.

Configuring the Transaction Management Dashboard

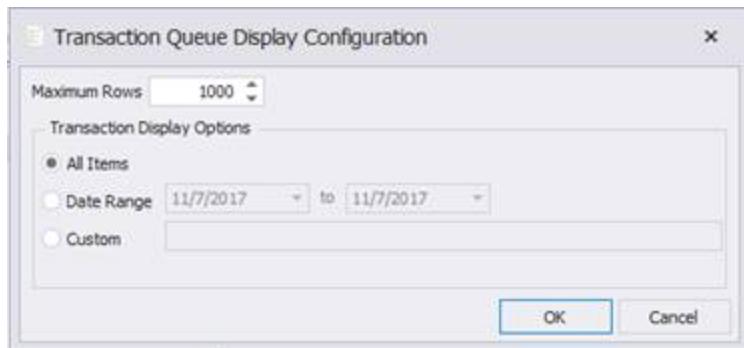


The **Configuration Menu** contains:

- **Application Preferences** - for setting your [user interface themes](#) and [locale](#).
- [**Application Database**](#) which stores the solution objects displayed in the User Management Console.

The **Dynamic View** button toggles between Static and Dynamic views of transactions. The Static View will display transactions that have already occurred whereas the Dynamic View displays transactions as they occur in real time.

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.

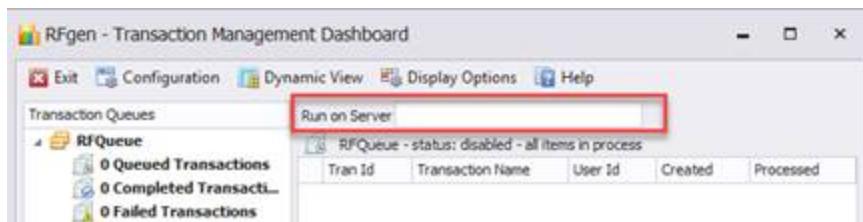
The **Custom** option is an ability to specify your own Where clause for the lookup. The actual names of the fields in the Queue database must be known as well as the type of field. An example would be:

where SeqNo = 1

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

The **Help** will display the online help for the Transaction Management Dashboard.

Run on Server

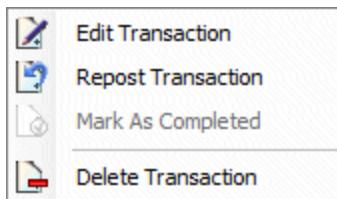


The **Run on Server** field is used to specify which server should own processing of a transaction if you have multiple servers connected to RFgen. If this field is left blank, RFgen will continue to work with all server(s) connected to it. If a server IP or "localhost" is entered, RFgen will work only with this server for queued transactions.

RFqueue Status Message



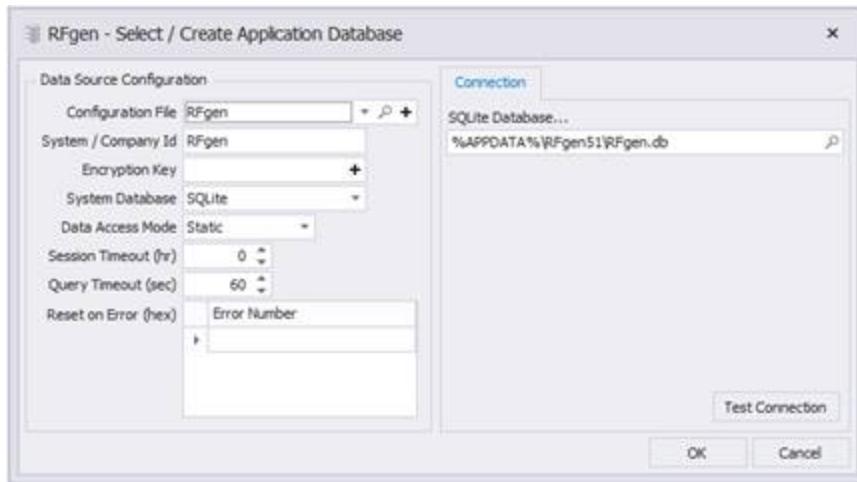
If you receive this message, your RFQueue is disabled because the Processing Cycle Time value is set to "0". To change this, go to the **Mobile Development Studio >Configuration > System Queues and Tasks** to modify the value.



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.

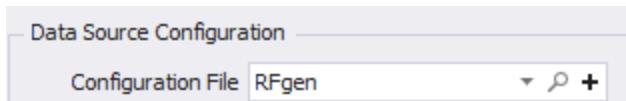
To Create or Select Your Data Source

In order to provide a database for storing and maintaining your RFgen Mobile Applications (which help run your Transactions) you need to connect to a database application/server/system to your RFgen server/system.



To configure/create your datasource, from the Transaction Management Dashboard menu bar, click on **Configuration > Application Database**. The RFgen - Select / Create Application Database Displays.

By default, a Configuration File called 'RFgen.rfc', defines the profile of the solution database, as shown below.



If you need to change the rfc file or select a different rfc file, you can use the list, search or plus (+) icons to browse to the %APPDATA%\ProgramData\RFgen51 folder.

This **Configuration File** was created when the Mobile Unity Platform software (RFgen Server) was installed. It identifies a Microsoft Access file called 'RFgen.mdb' located in the C:\ Users \ <username> \ AppData \

Roaming \ RFgen51 directory as the database that contains the programming items (Applications, Menus, Users and VBA code) written with the Mobile Development Studio, including the pre-scripted 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.

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 "rfc" 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** (in seconds) 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

Connection

If connecting to [Access Database...](#)

If connecting to [Oracle Database...](#)

If connecting to [SQL...](#)

If connecting to [SQLite...](#)

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.

System DataBase is Access

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.

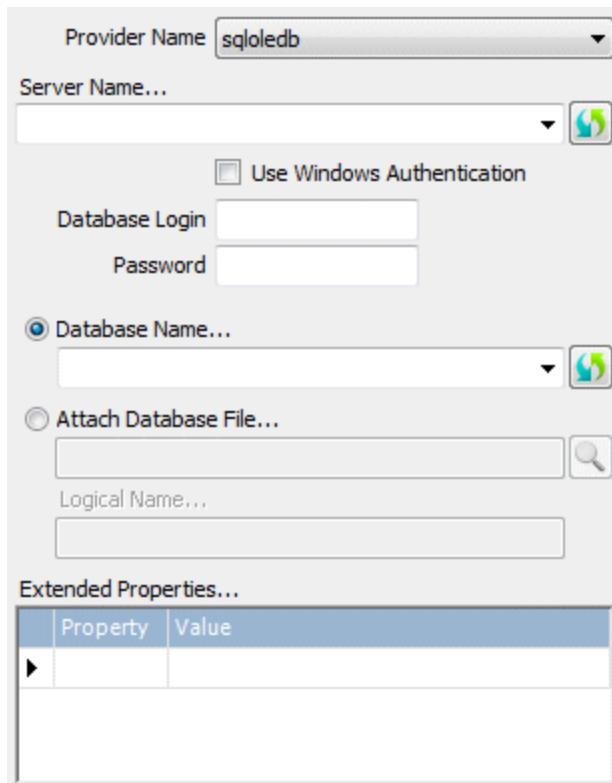
The screenshot shows a configuration dialog for an Access database. It includes fields for 'Provider Name' (dropdown), 'TNS Server Name' (dropdown), and 'Use Windows Authentication' (checkbox). Below these are fields for 'Database Login' and 'Password'. At the bottom is a section titled 'Extended Properties...' containing a table:

Property	Value
VCharNull	1
distribbx	0
enlist	FALSE

System DataBase is Oracle

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.

System DataBase is SQL



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.

System DataBase is SQLite



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.

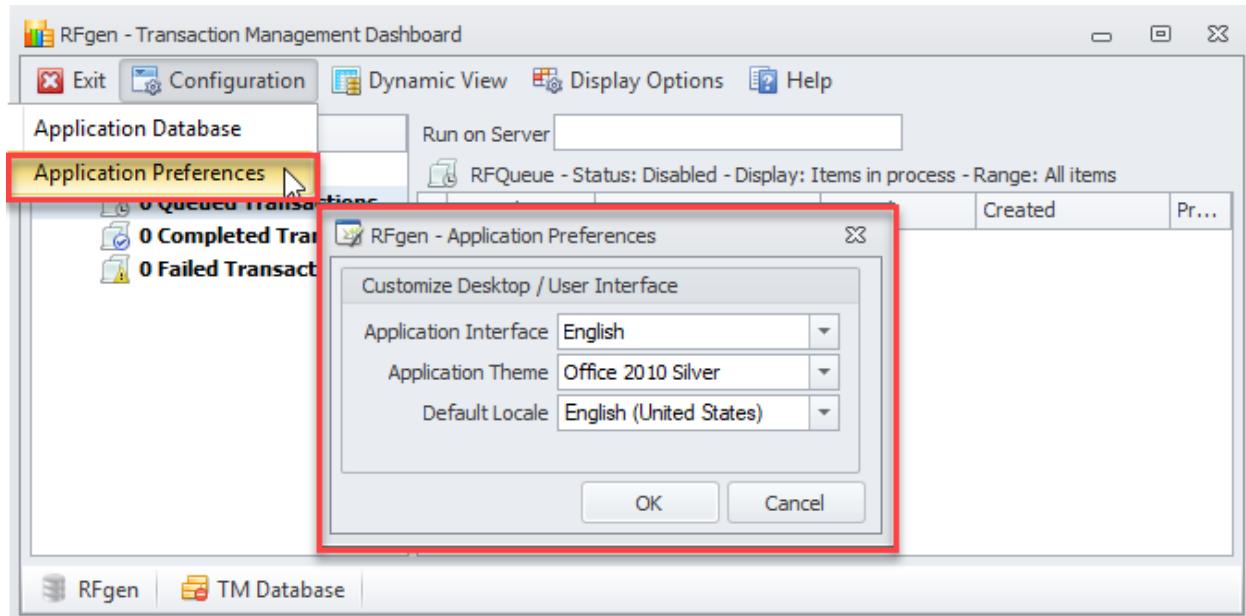
Finally click on the **Test Connection** button to verify connectivity. If the database has not already been setup to support the solution tables they will be created at this time. Clicking the Save Changes button will also create what is necessary but won't test the connection. Either button will also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

To Change Your Dashboard's Theme

You can change the general look and feel of your RFgen Transaction Management Dashboard by changing its skin/Application Skin. This will only affect your view of the dashboard and will have no affect on the Mobile Applications viewed by end users.

1. From the RFgen Transaction Management Dashboard, click on **Configuration > Application Preferences**.
2. Select the Application Theme from the drop down menu.
3. Click **OK**. The screen changes immediately to the chosen theme.

To Set the Locale in Your Dashboard

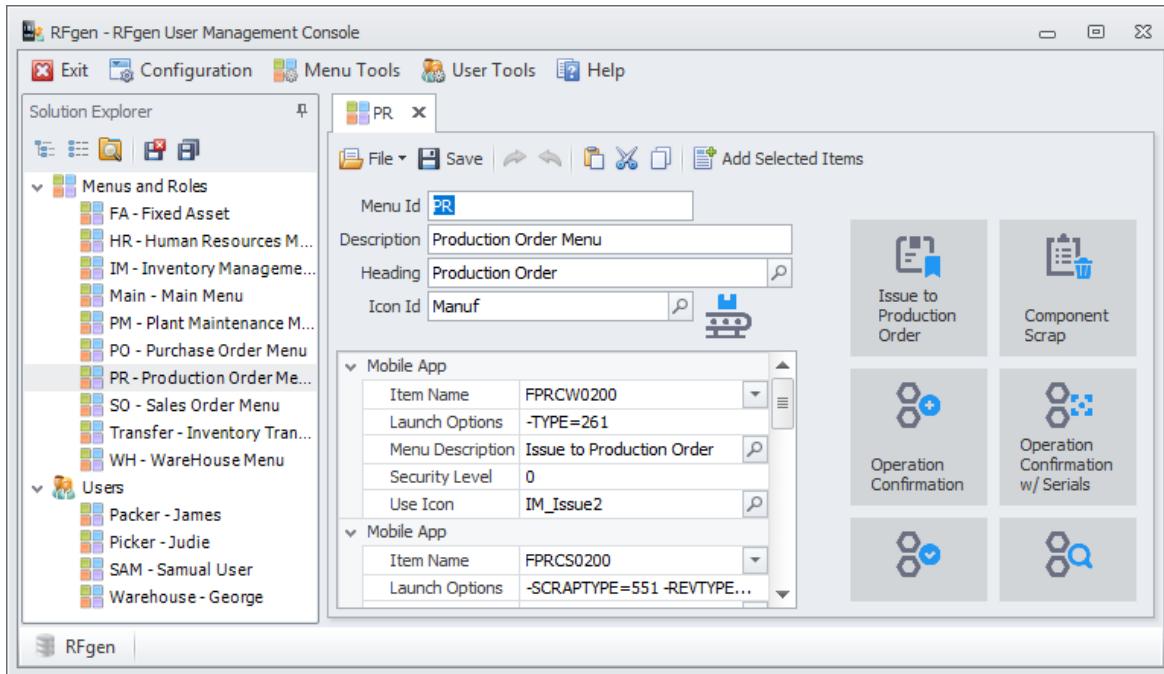


You can change the local\language your dashboard from English to any of the languages listed in the Application Interface drop down menu. This will only affect your view of the dashboard and will have no affect on the Mobile Applications viewed by end users.

The languages supported by the RFgen are: English, Arabic, Chinese, French, Japanese, and Spanish.

1. From the RFgen Transaction Management Dashboard, click on **Configuration > Application Preferences**.
2. Select the language you want to use from the Application Interface drop down menu. For example, English to Arabic. You can also set the Default Locale (i.e. specify the language used in a region such as English (United States)).
3. Click **OK**. The screen will shutdown immediately. (Or, you may need to manually restart to make the changes take place.)

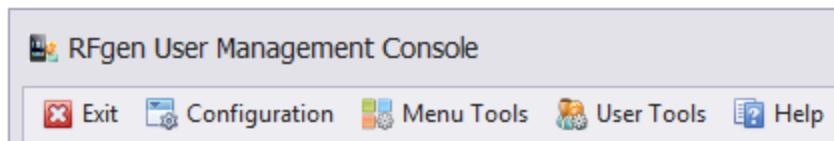
User Management Console



The User Management Console (UMC) enables warehouse managers to add, remove and manage Mobile Application users to/from the RFgen Mobile Unity Platform (RFgen Server). This allows the manager to manage users, assign roles and specific applications to each user without having to ask the RFgen Administrator for help. If additional changes are required to applications, code, macros or resources, these changes can be performed by the RFgen Administrator through the RFgen Mobile Development Studio.

The menus and functions in the User Management Console are a subset to those in the RFgen Mobile Unity Platform (RFgen Server).

User Management Console Menu



Exit allows you to exit the console.

Configuration Menu displays:

- **Application Preferences** - for setting the language, such as English or other locals for all applications, user interface themes (the coloring scheme of your User Management Console), and the Default Local for your User Management Console.

- **Application Database** - sets the Data Source (the database that stores the mobile applications that you assigned to users).

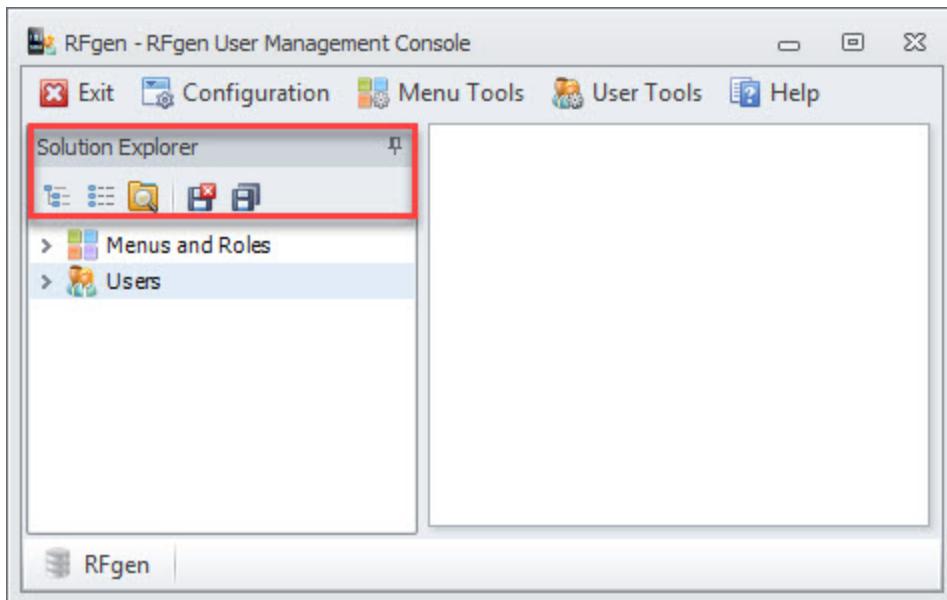
The **Menu** menus provide:

- Search for a Menu(s)
- Import and export Menus to Excel.

User Tools menus:

- Search by user attributes, then export the list.
- Import and export of users to Excel.

User Management Console Menu Bar



Icon Descriptions

Expands all nodes in the tree

Collapse all nodes in the tree

Closes all open objects

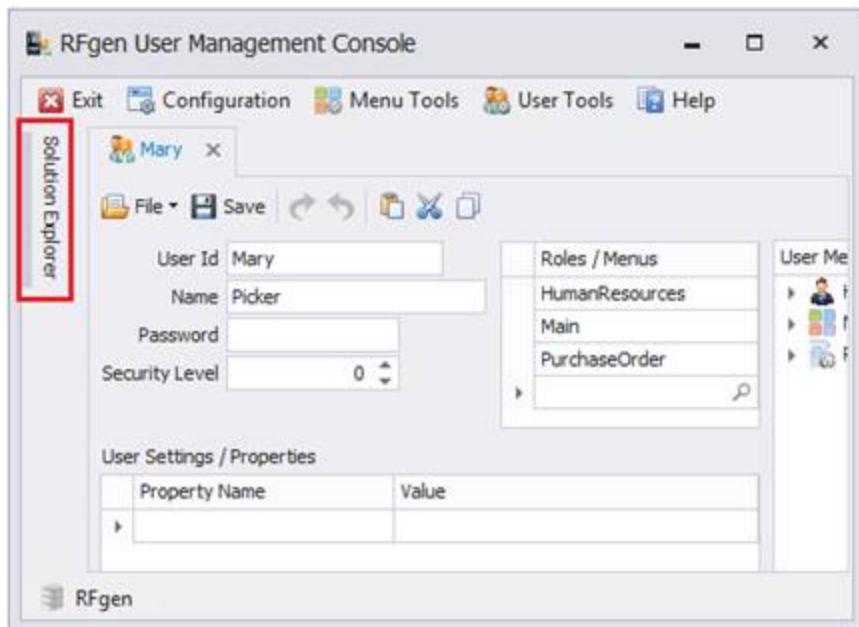
Saves all unsaved objects

Searches and replaces content in files

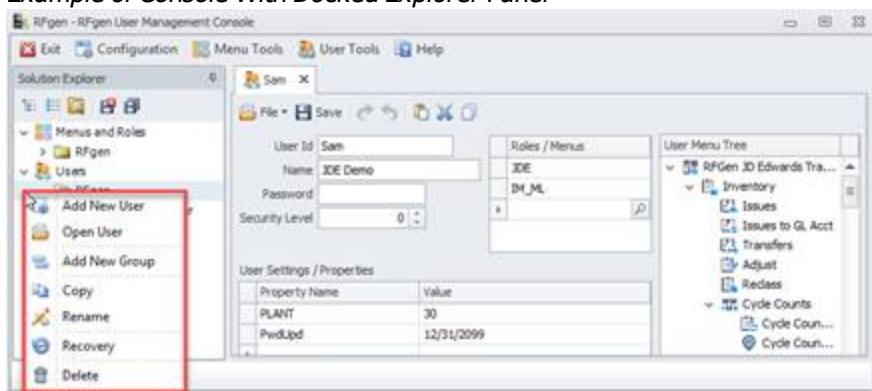
Allows you to dock the Solution Explorer panel to the left side of your Mobile Development Studio screen and toggle the hide or show bar which then gives you more space in the Studio. When the panel is docked, you

can click on the blue bar and hide the panel. To unhide the panel, click the blue bar. To redock a panel, click the pin again.

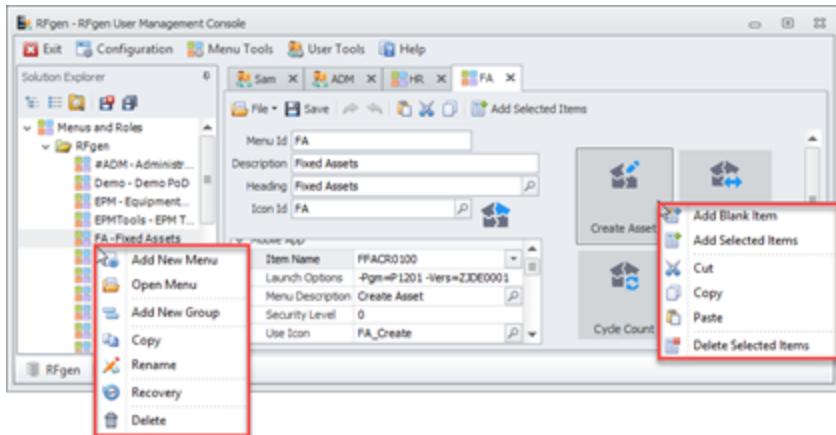
Example of Console With Solution Explorer Panel Is Selected



Example of Console With Docked Explorer Panel



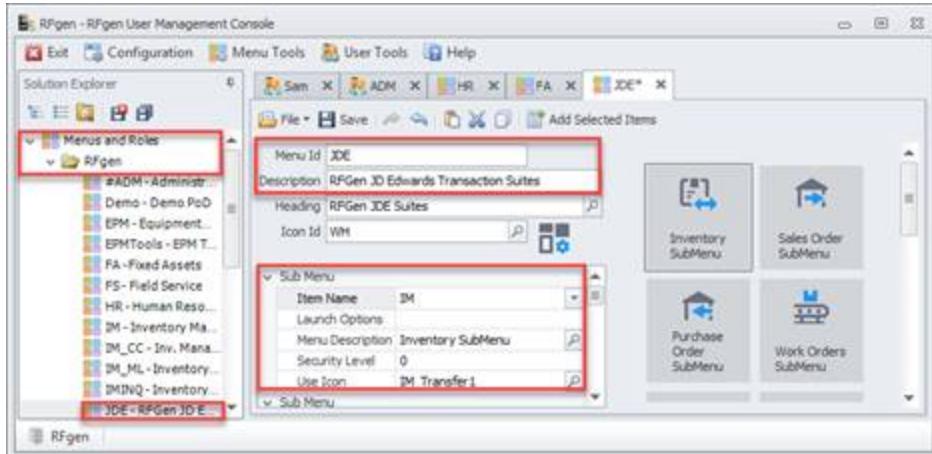
Menu and Roles Overview



The *Main Menu* is an example of a parent menu which is can be setup as the first menu that displays after a user logs in on their mobile device. The Sub Menus such as the *Inventory*, *Purchase Order*, *Sales Order*, and *Human Resources Menus* display when the user selects the *Main Menu*.

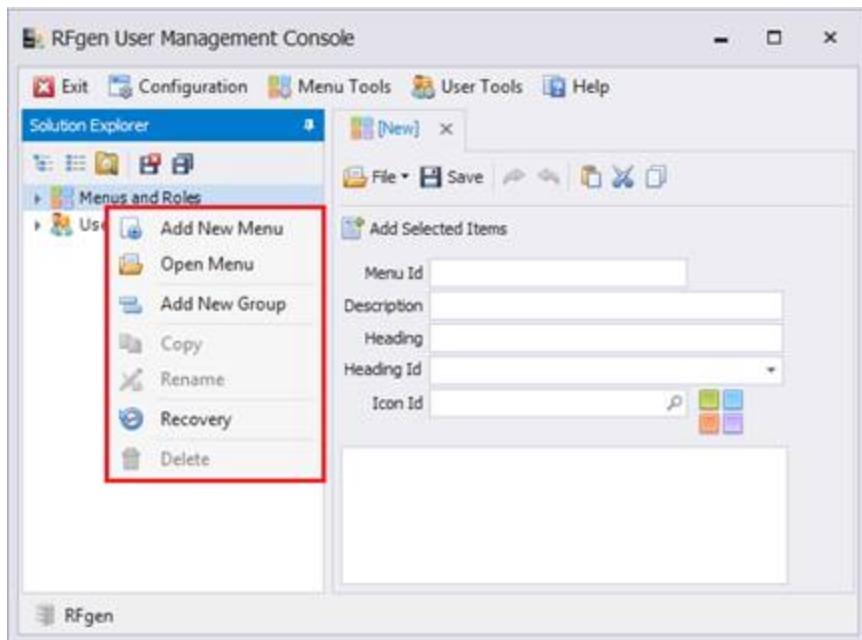
Once the menus, submenus and applications are setup, you can assign the menus to user accounts in the **Solution Explorer > Users** tree. This helps control which Mobile Apps a user will can access on their mobile device or Windows desktop system.

Note: Since "Menus and Roles" and "Users" are functions also in the RFgen Mobile Development Studio, changes made to there would update the same database that the User Management Console points to. Therefore the lastest entry will "win" when updates are made.

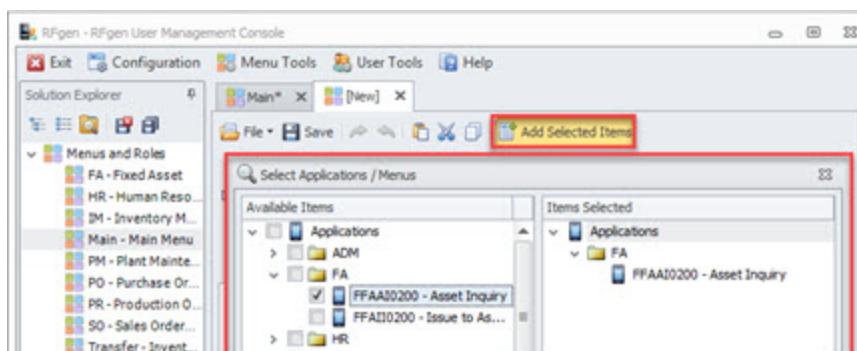


To Add Menus

This feature is available in the User Management Console and the Mobile Development Studio.

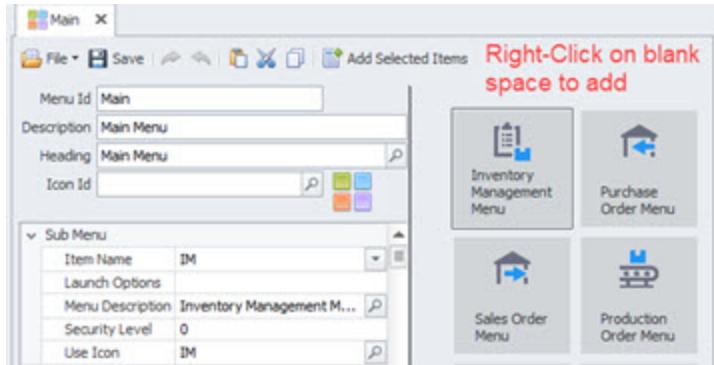


1. From the **Solution Explorer**, right-click on **Menus and Roles** and select **Add New Menu**. A blank form displays.
2. Complete the Menu Id, Description and Heading. The Icon ID and Heading are optional. Localization of Heading strings requires a corresponding string **ID** in the **Solution Explorer > Language Translation > Locale** grid. The **Solution Explorer > Configuration > Application Preferences: Language Set** Locale Name must also be set to the same Local as the one used in the Language Translation grid.
3. Click on **Add Select Items**. A window similar to the one below displays.



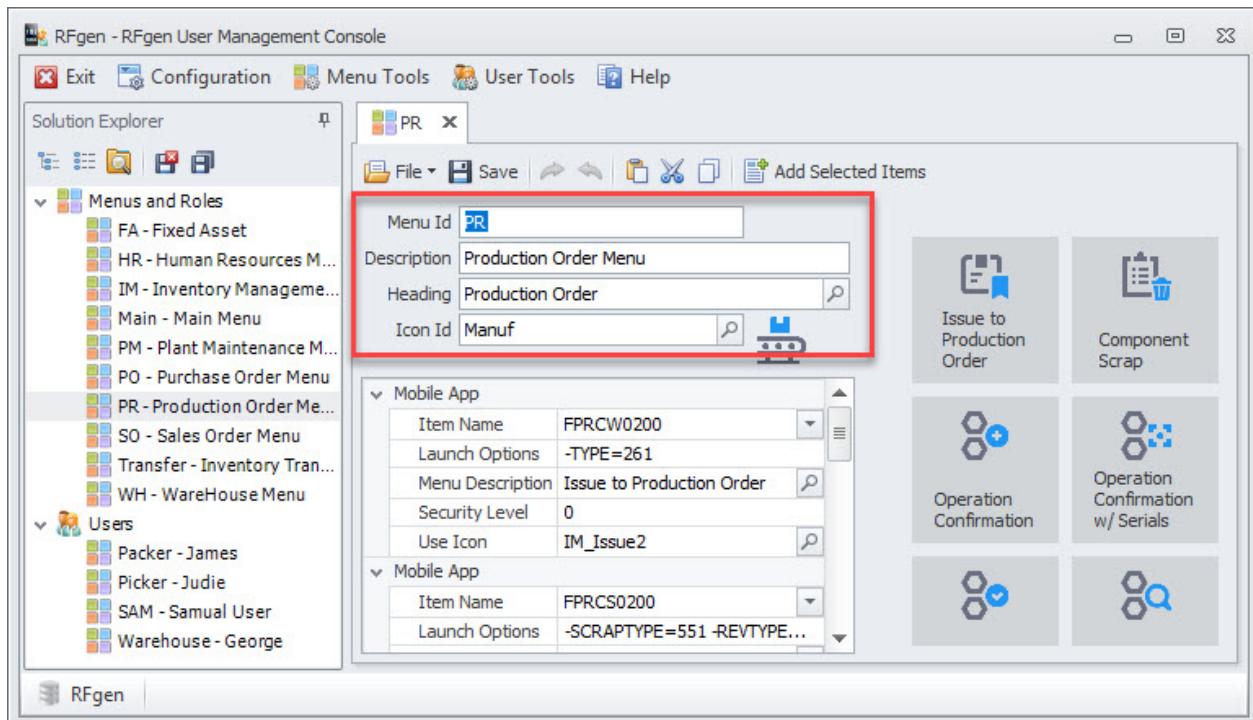
4. Expand the tree to view the item to be selected. Check the application or another menu from the list and click **OK**.
5. A **Mobile App** grid displays with the new entry.

6. Here's an example of an "Example" Menu:



7. From this point, you can also add new menus by right-clicking on a blank space.

User Management Console Menu Fields



The **Menu Id** is the name of the menu that is used when assigning the menu to users.

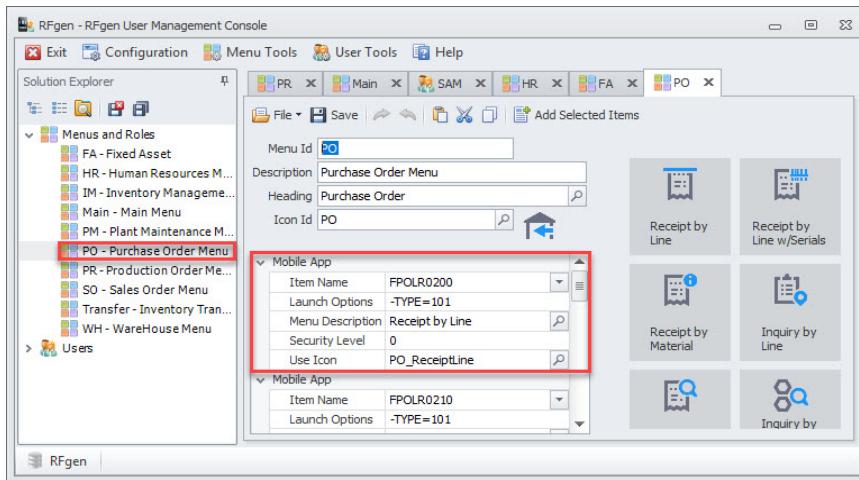
The **Description** is required and only shows in the Menu list.

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.

Mobile App Fields

These define the application icons that display under a Menu (Main Menu) or Sub Menu in the User Management Console. The icons and descriptions (items in the green boxes) are examples of what the user would see on his or her Mobile Device after they selected the menu (i.e. Inventory Menu).



The **Item Name** contains the name of the application that will be launched when its icon is selected.

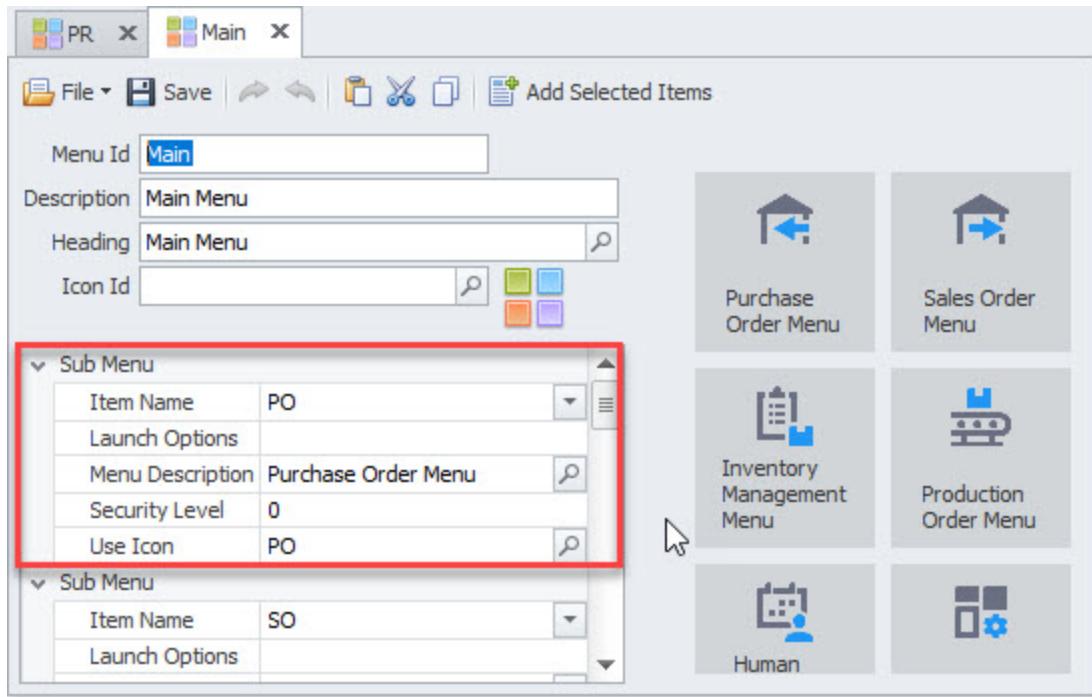
The **Launch Options** can be used to define how the application is launched. See *Launch Option Details* below.

The **Menu Description** appears under the Sub menu icon.

The **Menu Description ID** field is optional. Its used if you want the Menu Description translated or localized.

The **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.

Sub Menu Fields



These define the child menus to the parent menu; they are used to categorize and group mobile applications. For example, the *Inventory*, *Purchase Order*, *Sales Order*, and *Human Resources Menus* are the “Sub Menus” that display when a user clicks the *Main Menu*.

The **Item Name** is the sub menu name.

The **Launch Options** can be used to define how the sub menu is launched. See *Launch Option Details* below.

The **Menu Description** appears under the Sub menu icon.

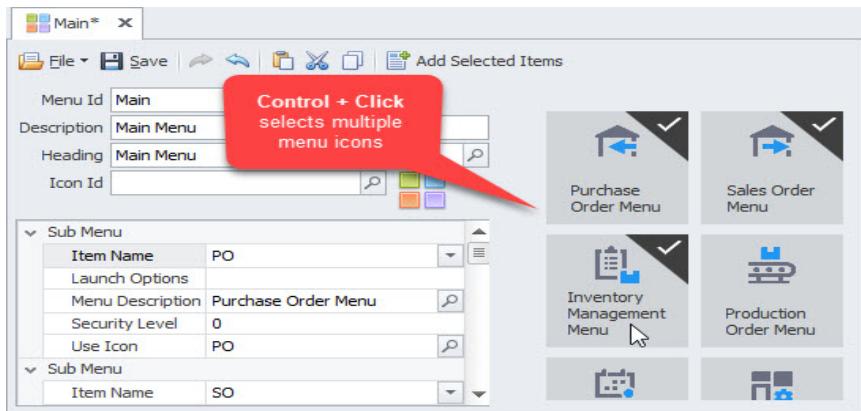
The **Menu Description ID** field is optional. Its used if you want the Menu Description translated or localized.

The **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.

The **Use Icon** allows you to select the image for this menu.

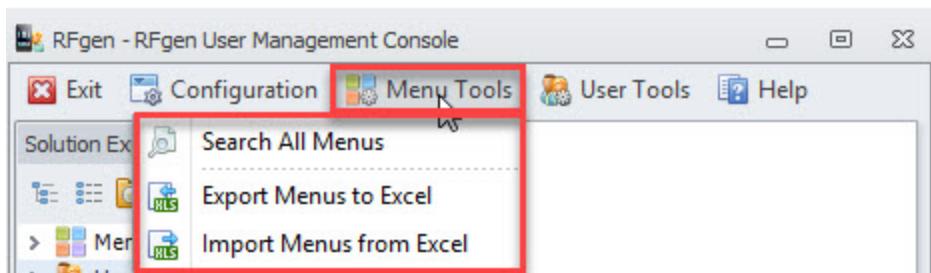
Menu Item Selection Tips

To Select Multiple Icons



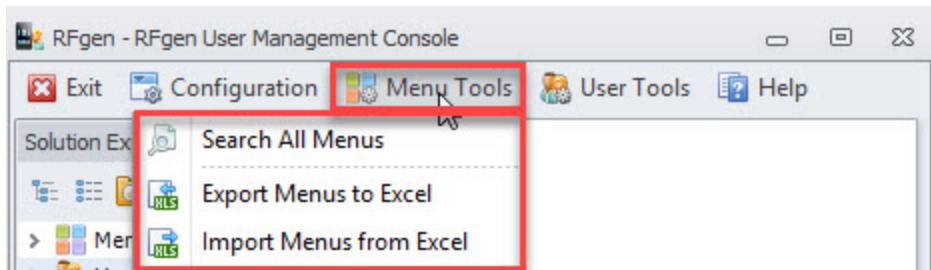
Use **Control+Click** to select multiple menu icons. A check mark will display in the upper right corner. Then make your change.

Search for Menus



1. To search for menus, click on the **Menu Tools** button. The **Search All Menus Utility** screen displays.
2. Click in the **Search Mode** and select the desired search type. This filter allows you to search the menu by its name or icon etc.
3. Enter your search information in the **Text to Find** box. You can also select the option to match the upper or lower case text and by the whole word. Click **Find**.
4. If the information is found, it displays in the area below. From here, you can then copy the list to Excel.
5. Click the **Excel** button and enter the destination location. Click **Save** when done.

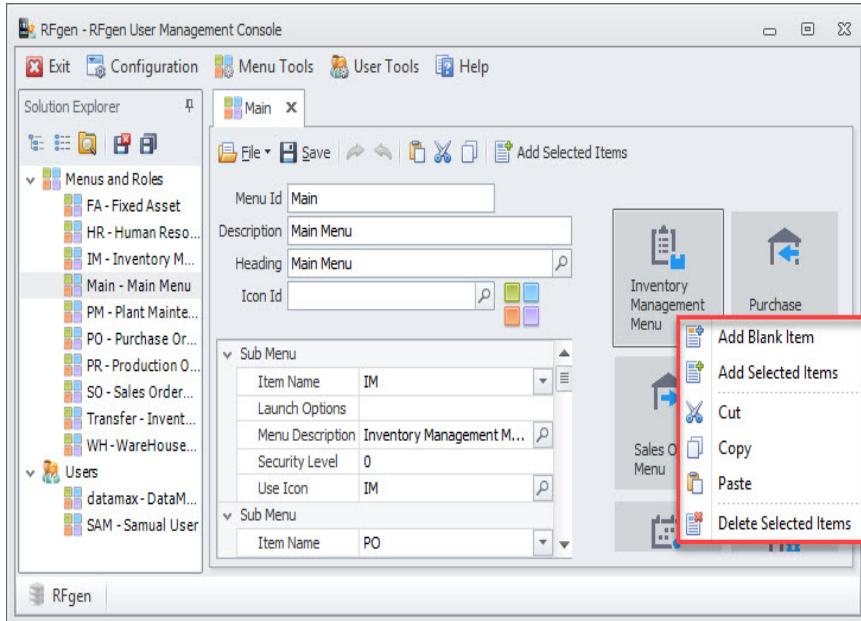
To Export or Import Menus



1. To export or import users to a Excel spreadsheet, click on the **User Management Console > Menu Tools> Export (or Import) to Excel** option. An **Export Utility** screen displays.

2. Enter the destination for the export (or import) in the **Export Location** (or **Import Location**) box.
3. Check the Menus you want to include and click **Export** (or **Import**) button.
4. A confirmation screen displays. Click **OK**. Your list should appear in the location specified.

To Remove or Rearrange Menus

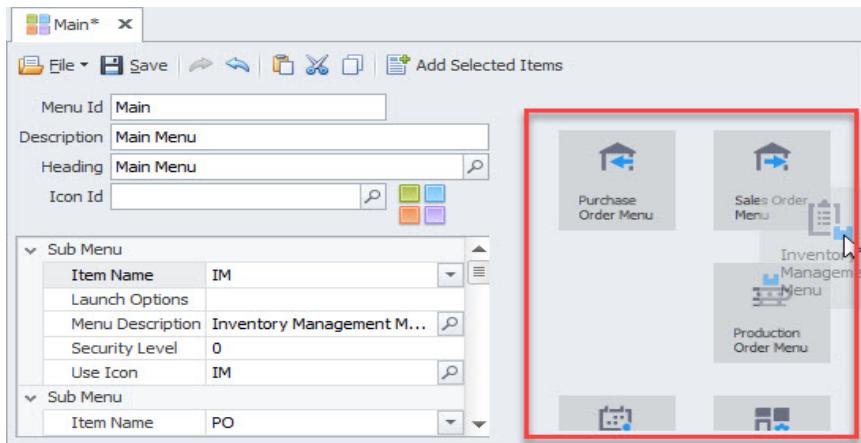


Clicking on the background will bring up the icon menu as shown above.

To Remove a Menu and its Form

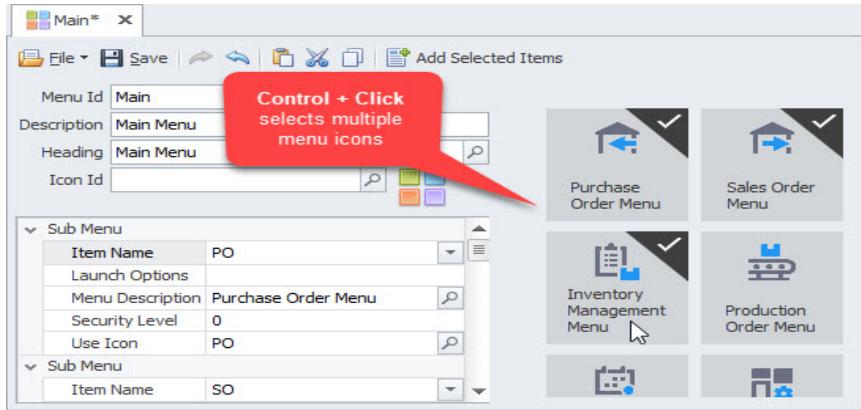
Click on the existing menu icon then click on **Delete** on your keyboard. Or, right-click on the icon and select "Delete Selected Items" from the menu. This menu allows you to add, remove or modify the menu form when you select the menu form's icon.

To Rearrange Menu Icons



To arrange the submenu icons so they appear in the order that you want, select the icon box, then drag it to its new location.

To Select Multiple Icons



Use **Control+Click** to select multiple menu icons. A check mark will display in the upper right corner. Then make your change.

Launch Option Details

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.

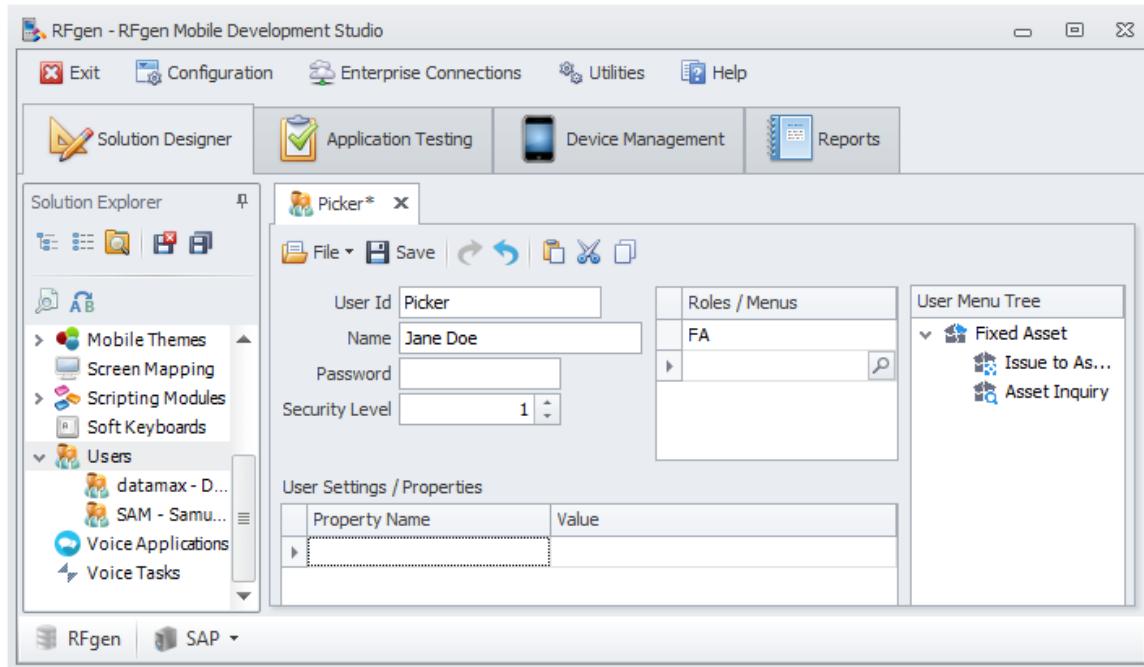
In a case where you **want the user to select from two variations** of the same application, add the application to the menu twice, give them different descriptions and then use two different command **Options**. Using the format –VARNAME=VALUE will create variables with values that can be referenced 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.

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.

Adding or Removing Users



If you want to assign specific applications (i.e. Inventory Management or Purchase Order Receiving) to specific users (or user roles), you can do this by:

1. Setting up unique Menus in the **Solution Designer > Menus and Roles** tree.
2. Assigning specific applications to each unique Menus and Roles menu.
3. Adding a user to the **Solution Designer > Users** tree.
4. Assigning the user to the unique Menu or Role.

When the user logs in, they are only given access to the menu that was assigned to them.

You can also prevent unauthorized access to your mobile apps, by configuring user Identifications and passwords and by setting the security access levels in the **Users** tree.

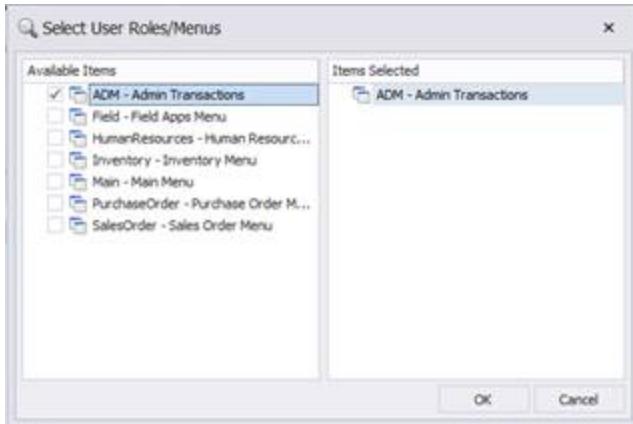
To add a user

TIP: Create your menus (or roles) and assign applications to the menus or roles before you create your users.

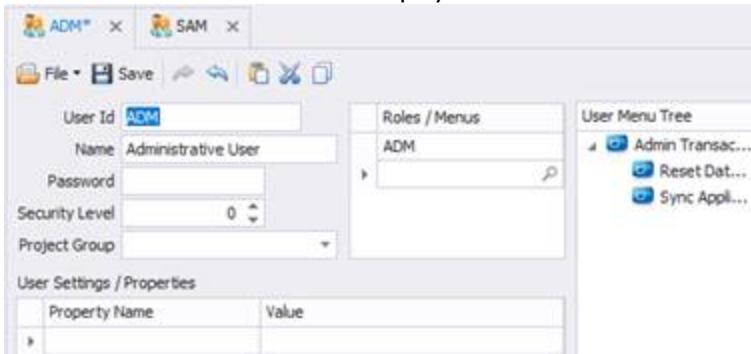
1. Navigate to the **Solution Designer > Users** tree.
2. Right-click on an existing user (or in the blank space) to add a new user, or right-click on the "Users" object and select **Add New User** from the menu.
3. The [New] user tab displays. Enter the user's information.
4. The **User Id** is required, but the **Password** is optional for a user account. SAM's startup menu is 'Main Menu'.
5. The **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.

Assign Menus or Roles to the User

6. In the **Roles / Menus** table are menus that come from the Roles/Menus tree. To assign a menu, click on the search icon and check the menu to be added, then click **OK**.



7. Selected menus for the user will display in the User Menu Tree.

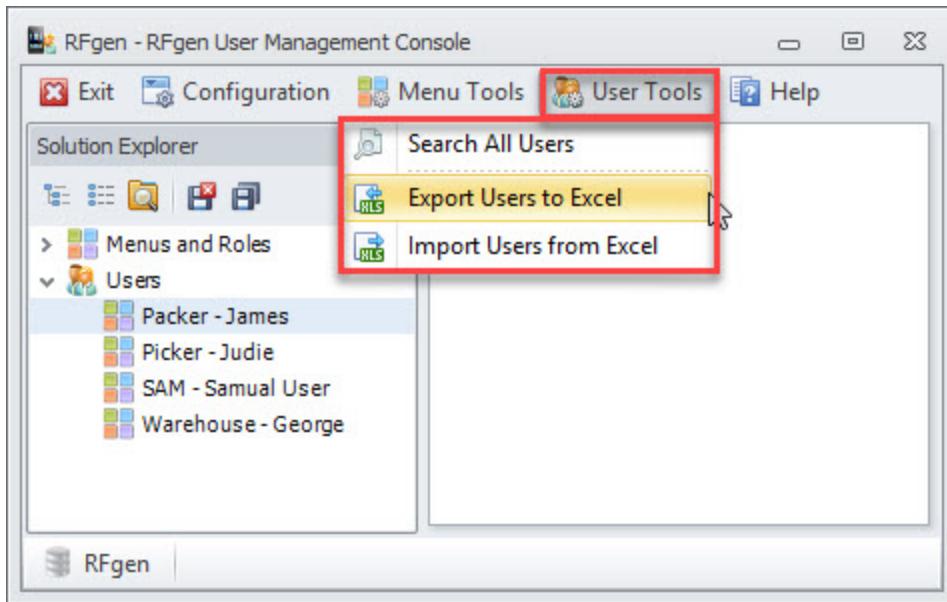


8. The **User Settings / Properties** allows the administrator to include any property and value for the selected user for the purposes of retrieving that data at run time. This has no effect on the user logging in.
9. When done, click the **Save**.

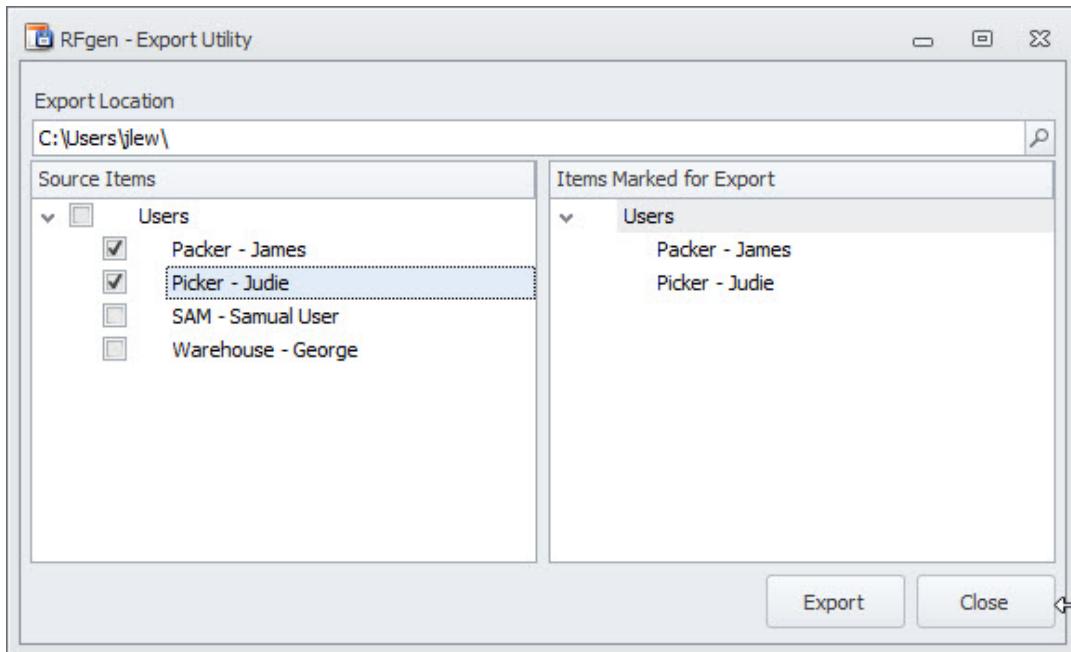
To Remove a User

You can remove a user by right-clicking on the user and selecting Delete.

User Tools: Export or Import Users

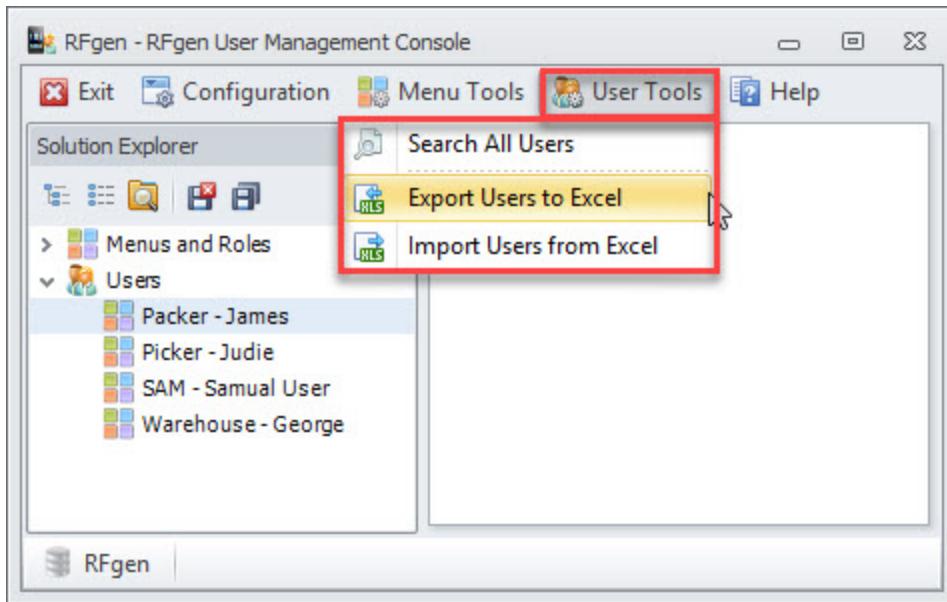


1. To export or import users to a Excel spreadsheet, click on the **User Management Console > User Tool > Export (or Import) to Excel** option. A **Export Utility screen** displays.
2. Enter the destination for the export (or import) in the **Export Location** (or **Import Location**) box.
3. Check the users you want to include and click **Export** (or **Import**) button.



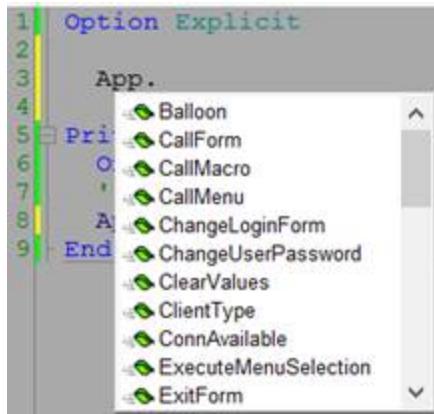
4. A confirmation screen displays. Click **OK**. Your list should appear in the location specified.

To Search Users



1. To search for a user/users, click on the **Users > Tool** button. The Search All Users Utility screen displays.
2. Click in the **Search Mode** and select the desired search type from the drop down menu. This filter allows you to search users if you know their name, or security level etc.
3. Enter your search information in the **Text to Find** box. You can also select the option to match the upper or lower case text and by the whole word. Click **Find**.
4. If the information is found, it displays in the area below. From here, you can then copy the list to Excel.
5. Click the Excel button and enter the destination location. Click **Save** when done.

VBA Language Extensions



Visual Basic Assembly (VBA) Language Extensions are comprised of other categories of functions which are used to manage more aspects of how an application looks or behaves. For example, a VBA Language Extension could be used to "Call the Form" of a specific application, or control a client device data display, manipulate database records, communicate with other databases, control stored procedures, communicate with attached or Windows-based printers and much more.

Some extensions work with the majority of prompts (also called "controls") while others will only work with specific types of prompts.

The extensions provided have been grouped into several categories as follows:

Database Related	Prompt-Specific	Soft Input Panel
Enterprise Resource Planning	Printer	System Error
Menu Strip	Screen Display	System Level
Mobile Device	Screen Mapping	Transaction Management
	Server-Based	

RFgen provides a VBA Language Extension .Net style dropdown menu that helps you complete a script at design time and view the VBA language extension options that are available.

You can also refer to the User Guide or Help for detailed descriptions of the parameters associated with the various VBA extensions.

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

Example:

```
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

Example:

```
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 record-set 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")
```

LogOff

This function is used to logoff a database 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 database connection is the default Source value.

Syntax:

[bValue =] db.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 = db.LogOff(1)
db.LogOff("SQL")
```

LogOn

This is used to logon the database connection and specify a user / password sequence for a database connection. Use of this feature is optional.

Note: the user does not always require this function as the server calls it automatically when a session starts.

Syntax:

[bValue =] db.LogOn ([vSource], [vUserId], [vUserPwd])

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 database system

vUserPwd (Variant) Optional – The login password to be used to connect to the database system

Example:

```
Dim bValue As Boolean
bValue = db.LogOn("SQLServer", "SQLUser", "SQLPass1")
bValue = db.LogOn(1, "SQLUser", "SQLPass1", "CLIENT=800|TYPE=3")
bValue = db.LogOn("Access", "AccessUser", "AccessPass1")
```

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.

Example:

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 adLock-BatchOptimistic.

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

Example:

```
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.

Example:

```
DB.UseDataSource ("RFSample")  
DB.UseDataSource (0)
```

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.

Example:

```
txtBox.Align = AlignCenter  
RFPrompt ("txtBox").Align = AlignCenter  
RFPrompt (2).Align = AlignCenter
```

AutoSize

This property is only used on the Label, Image, and MenuList controls. The options automatically size the control based on the content.

Fill - Fills the screen from its starting location on down.

Horizontal - Spans the content based on the width of the screen.

None - Disallows auto sizing.

Vertical - Stretches the content to match the vertical height of the control.

Syntax:

PromptID.Autosize = enValue

enValue (enSizeModes) an enumeration that contains AutosizeContent, AutosizeFill, AutosizeHorizontal, AutosizeNone and AutosizeVertical

Example:

```
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: lValue = PromptID.BackColor1

nValue (Long) is the color.

vValue (Variant) sets the color.

Example:

```
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: lValue = PromptID.BackColor2

nValue (Long) is the color.

vValue (Variant) sets the color.

Example:

```
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

Example:

```
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:

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

Example:

```
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.

Example:

```
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.

Example:

```

chkPrint.Checked = True
bValue = chkPrint.Checked
RFPrompt("chkPrint").Checked = False
RFPrompt(2).Checked = False

```

Children

This property is used for objects with parent-child relationships. If the child and parent have a property in common but with different values (i.e. background color, forecolor, fontsize etc.) the child's property value trumps the parent property value. There are four methods that are used with the Children property: Children.Append, Children.Count, Children.Item, and Children.Remove.

Children.Append

Use this method to clone objects contained by a parent control (Layout, Panel or TabControl). Children.Append takes a single parameter which is the control that you wish to clone and appends it to a VBA Prompt Array.

Syntax:

PromptID.Children.Append(ByVal pCtl As vbaPrompt, Optional Top As Long, Optional Left As Long)

ByVal pCtl is the prompt you want copied.

ByVal (Long) Top and Left is the location.

Example:

In the Cloning App below, the panel control (pnlChild) is cloned and will be appended to the end of the child controls of pChildren. The new controls will have the same parent that the pChildren control came from.

```

Dim pChildren As vbaPromptArray
Set pChildren = pnlParent.Children
pChildren.Append(pnlChild)

```

Children.Count

This method will return the number of child prompts that exist in the child collection. Its generally used when looping through the prompts to set properties.

Syntax:

PromptID.Children.Count

nCount = Children.Count

Example:

```

Dim nCount As Integer
Dim pParent As vbaPrompt
nCount = pParent.Children.Count

```

Children.Item

This method is used to access the array elements of the vbaPromptArray collection.

Syntax: PromptID.Children.Item(i)

ByVal (Variant) sets the item identifier for the child prompt.

i The index ID of a cloned control in a vbaprompt array.

Note: The variable "i" can also be used to reference the name of a prompt when given in string format.

Example:

In this example, pnlParent is the Panel Control.

```
Dim pChildren as vbaPromptArray  
Set pChildren = pnlParent.Children  
pChildren.Item(3).Caption = "Enter"  
pChildren.Item("TextBox1").Caption = "Enter"
```

Children.Remove

This method is used to remove a child object from the vbaPromptArray collection.

Syntax:

PromptID.Children.Remove(i)

ByVal (Variant) Index is the identifier

ByVal (Boolean) True if its successfully removed; False if it fails.

Example:

```
Dim pChildren As vbaPromptArray  
Set pChildren = pnlParent.Children  
pChildren.Remove (3)
```

Cloning

See "Children.Append"

To clone child objects, see Children.Append method.

For information on how to access a cloned object, refer to "**How to access cloned objects.**"

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.

Example:

```
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.

Example:

```
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.Edits = vValue

Alternate: sValue = PromptID.Edits

sValue (String) is the edit conditions for the prompt.

vValue (Variant) sets the edit conditions for the prompt.

Example:

```
txtPart.Edits = "2N" ' only accept 2 numbers  
RFPrompt("txtPart").Edits = "2N"
```

```
RFPrompt(2).Edits = "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.

Example:

```
txtPart.Edits = "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.

Example:

```
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.

Example:

```
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.

Example:

```
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.

Example:

```
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.

Example:

```
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.

Example:

```
txtPart.ForeColor = RGB(255,255,0) 'yellow
txtPart.ForeColor = &HFF0000      'blue
txtPart.ForeColor = QBColor(5)    'magenta
RFPrompt("txtPart").ForeColor = vbWhite
RFPrompt(2).ForeColor = vbWhite
```

Form.PageNo

This command returns the page number of the active page (the page that is displaying at runtime.)

Syntax:

vPage = Form.PageNo

vPage (Variant) is the page number.

Example:

```
If MenuAction = "SysBack" Then
    MenuAction = ""
    'Get the page number of the active page and
    'make it the Case number
```

```

Select Case Form.PageNo
Case 1: App.ExitForm
Case 2: Call BackFromPage2
Case 3: Call BackFromPage3
End Select
End If

Public Sub BackFromPage2
'1stLine is on page 1
lstLine.SetFocus
End Sub

Public Sub BackFromPage3
'Txt2Mat is on page 2
Txt2Mat.SetFocus
End Sub

```

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.

Example:

```

txtTime.Format = "hh:mm"
RFPrompt("txtTime").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.

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.

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.

Example:

```
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 Image 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

Example:

```

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

Example:

```

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.

Example:

```

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.

Example:

```
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.

Example:

```
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

Example:

```
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.

Example:

```
txtPart.Label.BackColor1 = vbWhite
txtPart.Label.BackColor1 = RGB(255,255,0) 'yellow
txtPart.Label.BackColor1 = &HFF0000      '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:

Syntax: PromptID.Label.BackColor2 = vValue

Alternate: nValue = PromptID.Label.BackColor2

vValue (Variant) sets the color.

nValue (Long) is the color.

Example:

```
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

Example:

```
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

Example:

```
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.

Example:

```
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.

Example:

```
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.

Example:

```
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.

Example:

```
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.

Example:

```
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.

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.

Example:

```
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.

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.

Example:

```
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

Example:

```
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.

Syntax:

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.

Example:

```
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.

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.

Example:

```
Dim vValue As Variant
```

```
vValue = txtPart.Label.Width  
vValue = RFPrompt("txtPart").Label.Width  
RFPrompt(2).Label.Width = 10
```

Layout.GetColSizeType

This method returns the SizeType value for the Layout control Column property.

Layout.GetColVisible

This method returns the value of *True* or *False* for the Visible property that's defined in the graphical, Layout control's Column Property. It can be used with the Layout.SetColVisible extension to dynamically change the property value in the graphical Layout control Columns property.

Layout.GetColWidth

This method returns the width value for the column that's defined in the graphical, Layout control's Columns property. It can be used with the Layout.SetColWidth extension to dynamically change the property value in the graphical Layout control Columns property.

Layout.GetRowHeight

This method returns the value for the height of row that's defined in the graphical Layout control's Rows property. It can be used with the Layout.SetRowHeight extension to dynamically change the property value in the graphical Layout control Rows property.

Layout.GetRowSizeType

This method returns the SizeType value for the row that's defined in the graphical Layout control's Row property. It can be used with the Layout.SetRowSizeType extension to dynamically change the SizeType property value in the graphical Layout control Rows property.

Layout.SetColSizeType

This method sets the SizeType value for the row that's defined in the graphical Layout control's Column property. It can be used with the Layout.GetColSizeType extension to dynamically change the SizeType property value in the graphical Layout control Rows property.

Layout.SetColVisible

This method sets the Visible value as *True* or *False* for the column that's defined in the graphical Layout control Columns property. It can be used with the Layout.GetColVisible extension to dynamically change the Visible property value in the graphical Layout control Columns property.

Layout.SetColWidth

This method sets the width value for the column that's defined in the graphical, Layout control's Columns property. It can be used with the Layout.GetColWidth extension to dynamically change the property value in the graphical Layout control Columns property.

Layout.SetRowHeight

This method sets the Height value for the row that's defined in the graphical, Layout control's Rows property. It can be used with the Layout.GetRowHeight extension to dynamically change the property value in the graphical Layout control Rows property.

Layout.SetRowSizeType

This method sets the SizeType value for the row that's defined in the graphical, Layout control's Rows property. It can be used with the Layout.GetRowSizeType extension to dynamically change the property value in the graphical Layout control Rows property.

Layout.SetRowVisible

This method sets the value as True or False for the Visible property that's defined in the graphical, Layout control's Rows property. It can be used with the Layout.GetRowVisible extension to dynamically change the Visible property value in the graphical Layout control Rows property.

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.

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.

Example:

```
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 the ListBox, MenuList and Combobox.

List.AddColSet

This method is used to add a set of columns in an empty ListBox control, or to replace an existing set of columns in a ListBox. This method is also used to add a new panel to a PanelList control in the event you want the arrangement of data to be different from the previous panel.

Syntax:

Prompt.List.AddColSet()

Example:

```
'Builds your first set of columns
ListBox1.List.AddColSet(1)

'Makes the columns easier to see
ListBox1.List.ColSet(1).LineStyle = LineBoth

'Adds a column and designates it as the first in column in the set (group) of columns.
ListBox1.List.ColSet(1).AppendColumn

'Adds a caption to the column heading to the first column in a set of columns
ListBox1.List.ColSet(1).Column(1).Caption = "Coll"
```

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.

Example:

```
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: lValue = PromptID.List.Cell(x,y).BackColor1

lValue (Long) is the color.

vValue (Variant) sets the color.

Example:

```
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: lValue = PromptID.List.Cell(x,y).BackColor2

lValue (Long) is the color.

vValue (Variant) sets the color.

Example:

```
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

Example:

```
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.

Example:

```
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).Expanded

This property expands or collapses all child nodes of the row specified for a Tree control. The column number is typically 1.

Syntax:

PromptID.List.Cell(x,y).Expanded = bValue

Alternate: bValue = PromptID.List.Cell(x,y).Expanded

bValue (Boolean) is True or False for expanding or collapsing all child nodes of the specified row

Example:

```
lstCars.List.Cell(2, 1).Expanded = True
lstCars.List.Cell(2, 1).Expanded = False
```

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: lValue = PromptID.List.Cell(x,y).ForeColor

lValue (Long) is the color.

vValue (Variant) sets the color.

Example:

```
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).Indent

This property gets or sets the level of the specified node within the tree. This property is only used for the Tree control.

Syntax:

PromptID.List.Cell(x,y).Indent = nValue

Alternate: nValue = PromptID.List.Cell(x,y).Indent

nValue (Long) is the node level within the tree

Example:

```
lstCars.List.Cell(5, 1).Indent = 4
```

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.

Example:

```
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.

Example:

```

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.

Example:

```

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:

IstCars.List.Clear(True)

RFPrompt("IstCars").List.Clear

RFPrompt(2).List.Clear

List.Column(x)

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>

Column(x) (Long) specifies the column number in the grid

Examples:

```
IstCars.List.Column(2).Width = 10
```

```
RFPrompt(2).List.Column(2).Width = 10
```

List.Column(x).Align

This property aligns the text or object of the column within a control that contains columns. For text types use the Text enumerations. For Image or Checkbox types use the general enumerations. The options are: BottomCenter, BottomLeft, BottomRight, CenterCenter, CenterLeft, CenterRight, TextCenter, TextLeft, TextRight., TopCenter, TopLeft, TopRight

Syntax: PromptID.List.Column(x).Align = enValue

enValue (enColAlign) an enumeration that contains BottomCenter, BottomLeft, BottomRight, CenterCenter, CenterLeft, CenterRight, TextCenter, TextLeft, TextRight., TopCenter, TopLeft, TopRight

Examples:

```
IstCars.List.Column(2).Align = TextCenter
```

```
RFPrompt("IstCars").List.Column(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:

```
IstCars.List.Column(2).Autosize = True
```

```
RFPrompt("IstCars").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: lValue = PromptID.List.Column(x).BackColor1

lValue (Long) is the color.

vValue (Variant) sets the color.

Examples:

```
IstCars.List.Column(2).BackColor1 = RGB(255,255,0) 'yellow
IstCars.List.Column(2).BackColor1 = &HFF0000      'blue
IstCars.List.Column(2).BackColor1 = QBColor(5)    'magenta
RFPrompt("IstCars").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: lValue = PromptID.List.Column(x).BackColor2

lValue (Long) is the color.

vValue (Variant) sets the color.

Examples:

```
IstCars.List.Column(2).BackGradient = GradientVertical
IstCars.List.Column(2).BackColor2 = RGB(255,255,0) 'yellow
IstCars.List.Column(2).BackColor2 = &HFF0000      'blue
IstCars.List.Column(2).BackColor2 = QBColor(5)    'magenta
RFPrompt("IstCars").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:

```
IstCars.List.Column(2).BackColor1 = RGB(0,0,255)
```

```
IstCars.List.Column(2).BackColor2 = vbWhite  
IstCars.List.Column(2).BackGradient = GradientVertical  
RFPrompt("IstCars").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:

```
IstCars.List.Column(2).Bold = True  
RFPrompt("IstCars").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:

```
IstCars.List.Column(2).Caption = "Model"  
RFPrompt("IstCars").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:

```
IstCars.List.Column(2).DisplayOnly = True
```

```
RFPrompt("txtPart").List.Column(2).DisplayOnly = False
```

```
RFPrompt(2).List.Column(2).DisplayOnly = False
```

List.Column(x).FontSize

This property accesses the font size parameter associated with a specified column in a control that supports columns like a Listbox or Combobox.

Syntax: PromptID.List.Column(x).FontSize = lValue

Alternate: vValue = PromptID.List.Column(x).FontSize

vValue (Variant) is the font size.

lValue (Long) sets the font size.

Example:

```
IstCars.List.Column(2).FontSize = 15
```

List.Column(x).ForeColor

This property accesses the column's fore color property associated with the specified column.

Syntax: PromptID.List.Column(x).ForeColor = lValue

Alternate: vValue = PromptID.List.Column(x).ForeColor

vValue (Variant) is the color.

lValue (Long) sets the color.

Examples:

```
IstCars.List.Column(2).ForeColor = RGB(255,255,0) 'yellow
```

```
IstCars.List.Column(2).ForeColor = &HFF0000 'blue
```

```
IstCars.List.Column(2).ForeColor = QBColor(5) 'magenta
```

```
RFPrompt("IstCars").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:

IstCars.List.Column(2).Format = "hh:mm"

RFPrompt("IstCars").List.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.

List.Column(x).ImageHeight

This property sets the images in this column to a specific height with prompts that support column.

Syntax: PromptID.List.Column(x).ImageHeight = IValue

Alternate: IValue = PromptID.List.Column(x).ImageHeight

IValue (Long) is the pixel height size for images in this column

Example:

IstCars.List.Column(1).ImageHeight = 10

List.Column(x).ImageWidth

This property sets the images in this column to a specific width with prompts that support column.

Syntax: PromptID.List.Column(x).ImageWidth = lValue

Alternate: lValue = PromptID.List.Column(x).ImageWidth

lValue (Long) is the pixel width size for images in this column

Example:

```
IstCars.List.Column(1).ImageWidth = 10
```

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:

```
IstCars.List.Column(2).Italic = True
```

```
RFPrompt("IstCars").List.Column(2).Italic = True
```

```
RFPrompt(2).List.Column(2).Italic = True
```

List.Column(x).MarginBottom

This property pads the bottom of all the cells in a specified column. It also contributes to the overall height of the entire row.

Syntax: PromptID.List.Column(x).MarginBottom = lValue

Alternate: lValue = PromptID.List.Column(x).MarginBottom

lValue (Long) the padding in pixels between the contents of a cell and the bottom of the cell.

Examples:

```
IstCars.List.Column(2).MarginBottom = 5
```

```
RFPrompt("IstCars").List.Column(2).MarginBottom = 5
```

```
RFPrompt(2).List.Column(2).MarginBottom = 5
```

List.Column(x).MarginLeft

This property pads the left of all the cells in a specified column.

Syntax: PromptID.List.Column(x).MarginLeft = IValue

Alternate: IValue = PromptID.List.Column(x).MarginLeft

IValue (Long) the padding in pixels between the contents of a cell and the left of the cell.

Examples:

IstCars.List.Column(2).MarginLeft = 5

RFPrompt("IstCars").List.Column(2).MarginLeft = 5

RFPrompt(2).List.Column(2).MarginLeft = 5

List.Column(x).MarginRight

This property pads the right of all the cells in a specified column.

Syntax: PromptID.List.Column(x).MarginRight = IValue

Alternate: IValue = PromptID.List.Column(x).MarginRight

IValue (Long) the padding in pixels between the contents of a cell and the right of the cell.

Examples:

IstCars.List.Column(2).MarginRight = 5

RFPrompt("IstCars").List.Column(2).MarginRight = 5

RFPrompt(2).List.Column(2).MarginRight = 5

List.Column(x).MarginTop

This property pads the top of all the cells in a specified column. It also contributes to the overall height of the entire row.

Syntax: PromptID.List.Column(x).MarginTop = IValue

Alternate: IValue = PromptID.List.Column(x).MarginTop

IValue (Long) the padding in pixels between the contents of a cell and the top of the cell.

Examples:

IstCars.List.Column(2).MarginTop = 5

RFPrompt("IstCars").List.Column(2).MarginTop = 5

RFPrompt(2).List.Column(2).MarginTop = 5

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:

IstCars.List.Column(1).ScaleDecimals = 2

RFPrompt("IstCars").List.Column(1).ScaleDecimals = 2

RFPrompt(8).List.Column(1).ScaleDecimals = 2

List.Column(x).Style

This property gets or sets the type of column. The values are Text, Image, and Check box.

Syntax: PromptID.List.Column(x).Style = enStyle

Alternate: enStyle = PromptID.List.Column(x).Style

enStyle (enColumnStyle) Contains ColumnStyleCheck, ColumnStyleImage, and ColumnStyleText

Examples:

IstCars.List.Column(1).Style = ColumnStyleCheck

RFPrompt("IstCars").List.Column(1).Style = ColumnStyleText

RFPrompt(8).List.Column(1).Style = ColumnStyleImage

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:

IstCars.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:

```
IstCars.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:

```
IstCars.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:

```
IstCars.List.Column(2).Width = 25  
RFPrompt("IstCars").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 = IstCars.List.Columns
```

```
iCnt = RFPrompt("IstCars").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 = IstCars.List.Count
```

```
nValue = RFPrompt("IstCars").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", "", "Cor-
vette","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.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 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 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(nCol, sHeading, vDefWidth, [eValue], [bTrimSpaces], [nScaleDecimals])

nCol (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 characters. If -1 is used the column will dynamically size to the widest value in the column.

eValue (enColAlign) Optional – how to align the column; either BottomCenter, BottomLeft, BottomRight, CenterCenter, CenterLeft, CenterRight, TextCenter, TextLeft, TextRight., TopCenter, TopLeft, TopRight

bTrimSpaces (Boolean) Optional – formats the values in the specified column by deleting the leading and trailing spaces in the data.

nScaleDecimals (Long) 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.

Example:

```
cMyList.SQL = "select * from PLANTS"
```

```

oMyList.SetColumn 1, "ID", 5, TextLeft, True
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.RemoveColSet

This method is used to manually remove a ColSet from a list object prompt.

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

Example:

```

lstParts.List.RemoveItem(3) ' Removes the third item
RFPrompt ("lstParts") .List.RemoveItem(3)
RFPrompt (2) .List.RemoveItem(3)

```

List.RowSelector

This method activates or deactivates row highlighting during row selection for list box controls

Syntax: PromptID.List.RowSelector (bValue)

bValue(Boolean) is the display state for the row selector

Example:

```
lstParts.List.RowSelector = False 'Turns off Row Highlighting
```

List.SetDefaultColSet

This method sets the default as either a one- or zero-based collection. For example, if you have a group of columns, you can set the default to identify the first column as Col 0 (zero-based collection) and the next one, Col 1 and the last one as Col 3.

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)
```

Map.CenterMap

Use this method to specify the center of a map/area using the Latitude and Longitude. This extension only functions with the Map control.

Syntax: bValue = PromptID.Map.CenterMap(sLat, sLng)

sLat: (String) is the latitude of the center of the map

sLong: (String) is the longitude of the center of the map

Example:

```
Dim SLat As String
```

```
Dim sLng As String
```

```
sLat = "38.575764"
```

```
sLng = "121.478851"  
mMapArea.Map.CenterMap(sLat,sLng)
```

Map.GetAddress

This method returns an address converted from the longitude and latitude geo-coordinates. This extension only works with the Map control.

Syntax: bValue = PromptID.Map.GetAddress(sLat, sLng, sAddrs)

bValue: (Boolean) True returns a value if successful; False if it fails

sLat: (String) is the Latitude of the address

sLng: (String) is the Longitude of the address

sAddrs: (String) is the address converted from the Latitude and Longitude

Example:

```
sLat = "38.575764"
```

```
sLng = "121.478851"
```

```
mMapArea.Map.GetAddress(sLat,sLng,sAddrs)
```

Map.GetDirections

This returns the driving directions between a start and destination. This extension only works with the Map control.

Syntax: bVal = mapArea.Map.GetDirections(sOrigin, sDest, sDir, bGetMap, sDur, sDist)

bVal: (Boolean) Returns True if the conversion was retrieved; False if it failed.

sOrigin: (String) is the origination or start of the route

sDest: (String) is the route destination

sDir: (String Array) are the directions -- the list of roads between origination and destination

bGetMap: (Boolean) Returns True if the map was retrieved; False if it failed.

sDur: (String) is the duration of the distance traveled

sDist: (String) is the distance traveled

Example:

```
DIM sDir as String
```

```
mapArea.Map.GetDirections(txtOrig.Text, txtDest.Text, sDir, True)
```

Map.GetLatLng

Use this method to return the Latitude and Longitude of a physical address. This extension only works with the Map control.

Syntax: bVal = PromptID.Map.GetLatLng (sAddr, sLat, sLng)

bValue (Boolean) True means the conversion was retrieved; False means it failed.

sAddr: (String) is the address of the location

sLat: (String) is the latitude of the location

sLng: (String) is the longitude of the location

Example:

```
MapArea.Map.GetLatLng(txtAddress.Text, textLat.Text, textLng.Text, True)
```

Map.PlanRoute

Use this method to obtain the most efficient driving route for multiple locations. This extension only works with the Map control.

Syntax: bVal = PromptID.Map.PlanRoute(sOrigin, sDest, sPoints, bGetMap)

bVal: (Boolean) True means the conversion was retrieved; False means it failed

sOrigin: (String) is the origination or start of the planned route

sDest: (String) is the destination of the planned route

sPoints: (String Array) Points are an array of delivery addresses and once its returned, the points will contain the sorted addresses.

bGetMap: (Boolean) Returns True if the map was retrieved; False if it failed.

Example:

```
Dim sPoints(3)As String
```

```
Dim sSrc as String
```

```
Dim sDst as String
```

```
sSrc = "Services, 200 Plaza Dr, Folsom, CA 95630, USA"
```

```
sDst = "Services, 200 Plaza Dr, Folsom, CA 95630, USA"
```

```
sPoints(0) = "Services, 200 Plaza Dr, Folsom, CA 95630, USA"
```

```
sPoints(1) = "3321 Durock Rd, Cameron Park, CA 95682, USA"  
sPoints(2) = "300 Automall Dr, Roseville, CA 95661, USA"  
If not mapRoute.Map.PlanRoute(sSrc,sDst, sPoints, True) Then  
App MsgBox "Your route plan needs to be revised."
```

Map.Zoom

Use this method to zoom in/out of the map area. This extension only functions works with the Map control.

Syntax: PromptID.Map.Zoom = nValue

nValue (Long) sets the Zoom size

Example:

```
Dim sDirs() As String  
Dim sLeg As String  
mapDir.Map.Zoom = 10
```

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

The RFPrompt

This control has been deprecated since 5.0 but is still used in legacy code.

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
```

SetFocus

This method is equivalent to the App.SetFocus command but in this case the syntax check will catch errors because the prompt name must exist. This method is the preferred approach.

Syntax: [bValue =] PromptID.SetFocus([bSaveRSP], [bValidateRSP])

bValue (Boolean) Optional – Returns True or False depending on the success of the command

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:

PartNo.SetFocus ' Cursor will go to prompt PartNo

PartNo.SetFocus(True, True) ' Cursor will go to "PartNo", save current Text property and execute edits and OnEnter event

Tab

This property is only used with the **TabControl** prompt which enables users to toggle between pages or groups of prompts at run time. At runtime, the **TabControl** determines which tab is active and which one is inactive. The following provides the methods for styling (coloring, shaping of the bezel and borders) for the active tab.

The **Tab.Caption** resets the caption of the specified tab to the text assigned to it. The remainder of the functions Tab.Curxxx or Tab.Altxxx set values for the tabs when they are active or inactive respectively. If multiple tabs are used, RFgen automatically knows which are active or inactive, and applies these property values accordingly.

Syntax: PromptID.Tab <method or property>

bVal (Varient) sets the identifier for the active or inactive tab prompt in a TabControl.

PromptID.Tab. AltBtnBackColor1 = vValue

Alternate: nValue = PromptID.Tab. AltBtnBackColor1

vValue (Varient) sets the color

nValue (Long) is the color

Examples:

```
TabControl.Tab.AltBtnBackColor1 = vbRed  
TabControl.Tab. AltBtnBackColor2 = vbYellow  
TabControl.Tab. AltBtnBackGradient = GradientVertical  
TabControl.Tab. AltBtnBevel = 5  
TabControl.Tab. AltBtnBorderColor = vbRed  
TabControl.Tab. AltBtnBorderStyle = DisplayNoBorder  
TabControl.Tab. AltBtnForeColor = vbBlue  
TabControl.Tab.CurBtnBackColor1 = vbRed  
TabControl.Tab.CurBtnBackColor2 = vbYellow  
TabControl.Tab.CurBtnBackStyle = GradientVertical  
TabControl.Tab.CurBtnBevel = 5  
TabControl.Tab.CurBtnBorderColor = vbRed  
TabControl.Tab.CurBtnBorderStyle = DisplayNoBorder  
TabControl.Tab.CurBtnForeColor = vbBlue
```

Tab.AltBtnBackColor1

This method accesses the inactive tab button's background color in a TabControl prompt, and is the primary background color for the button. Refer to "Tab" for syntax and code examples for this property.

Tab.AltBtnBackColor2

This property accesses the inactive tab button's background color in a TabControl prompt, and is the secondary background color for the button. Refer to "Tab" for syntax and code examples for this property.

Tab.AltBtnBackStyle

This property sets the background coloring style for the inactive tab button(s) on the graphical TabControl. The style can be set to use just the AltBtnBackColor1 for a solid background, or both AltBtnBackColor1 and AltBtnBackColor2 to create a gradient (blended) color in several directions.

Syntax: PromptID.Tab.AltBtnBackStyle = enGradients

enGradients: GradientDefault, GradientDiagonalLeft, GradientDiagonalRight, GradientHorizontal, GradientNone, GradientVertical, and GradientVerticalSplit

Tab.AltBtnBevel

This method sets the roundness of inactive tab button's edges in a TabControl prompt. Refer to the syntax and examples listed under "Tab".

Tab.AltBtnBorderColor

This method accesses the inactive tab button's boarder color in a TabControl prompt. Refer to the syntax and examples listed under "Tab".

Tab.AltBtnBorderStyle

This method accesses the inactive tab button's boarder style in a TabControl prompt. Refer to the syntax and examples listed under "Tab".

Tab.AltBtnForeColor

This method accesses the inactive tab button's data field fore color in a TabControl prompt. Refer to the syntax and examples listed under "Tab".

Tab.Caption

This method resets the caption of the specified tab to the text assigned to it. Refer to the syntax and examples listed under "Tab". Note: This method can be applied to active an inactive tab.

Syntax: PromptID.Tab.Caption = vValue

Alternate: sValue = PromptID.Tab.Caption

sValue (String) is the caption of the prompt's tab

vValue (Varient) is the caption to display for the prompt.

Example:

```
tabPlant.Tab.Caption(1) = "Changed"
```

Tab.CurBtnBackColor1

This method accesses the active tab button's background color in a TabControl prompt, and is the primary background color for the button. Refer to "Tab" for syntax and code examples for this property.

Tab.CurBtnBackColor2

This method accesses the inactive tab button's background color in a TabControl prompt, and is the secondary background color for the button. Refer to "Tab" for syntax and code examples for this property.

Tab.CurBtnBackStyle

This method accesses the active tab button's coloring style in a TabControl prompt, and uses either just the BackColor1 for solid backgrounds or both BackColor properties to create gradients in one of several directions. Refer to the examples listed under "Tab".

Syntax: PromptID.Tab.CurBtnBackStyle = enGradients

enGradients: GradientDefault, GradientDiagonalLeft, GradientDiagonalRight, GradientHorizontal, GradientNone, GradientVertical, and GradientVerticalSplit

Tab.CurBtnBevel

This method sets the roundness of active tab button's edges in a TabControl prompt. Refer to the syntax and examples listed under "Tab".

Tab.CurBtnBorderColor

This method accesses the active tab button's boarder color in a TabControl prompt. Refer to the syntax and examples listed under "Tab".

Tab.CurBtnBorderStyle

This method accesses the active tab button's boarder style in a TabControl prompt. Refer to the syntax and examples listed under "Tab".

Tab.CurBtnForeColor

This method accesses the inactive tab button's data field fore color in a TabControl prompt. Refer to the syntax and examples listed under "Tab".

Tab.OpenTab

This method switches the focus to the active tab in a TabControl prompt.

Syntax:

iVal (Index) the numeric identifier of the tab in the graphical TabControl. The tab index is one-based (not zero-based).

Example

```
'Make my second tab in my TabControl the active tab.  
TabControl.Tab.OpenTab(2)
```

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.

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
```

TreeView

This is not a property; Its a graphical prompt that lists data in a tree in a hierachial, nested tree format. Its methods are similar to the ListBox controla. Use the List property to associate data with this graphical prompt.

Example

```
Dim Assembly_A As vbaRow
```

```
Dim Assembly_B As vbaRow
```

```
Dim SubAssembly_A As vbaRow
```

```
Dim SubAssembly_B As vbaRow

Set Assembly_A = TreeView1.List.AddItem(1, "Assembly A: Glass")
Set SubAssembly_A = Assembly_A.AddItem(2, "Location: A01")
Set Assembly_B = TreeView1.List.AddItem(1, "Assembly B: Frame")
Set SubAssembly_B = Assembly_B.AddItem(2, "Location: B01")
```

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

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.

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 pop-up 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:

```
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.

Example:

```
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.

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

Reload

This command performs the same logic as exiting the form and reselecting it from the menu.

Syntax: App.Reload

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 -	vbKeyNumpad0 –
vbKeyF16	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.

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.

Example:

```
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)
```

SQLNum

This command has been obsoleted.

Example:

```
Dim sValue As String
```

```
sValue = SQLNum("123.456,78") ' returns "123,456.78"
```

Theme

This command provides a quick way to change the active theme in code. It only affects the local emulator, and doesn't change anything on the device.

Syntax: `App.Theme`

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"
```

Update

This function will invoke RFgen's normal last-prompt update cycle which will call the Form_OnUpdate event and, if successful, clear the form and reset the focus to the first input prompt. It will return a Boolean value if the update processing was successful.

Syntax: App.Update

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"
```

Menu Strip Extensions

The Menu Strip is a panel of commands that can appear at the top, bottom, or sides of a screen, turned on/ off by the user, or be locked on/off by design. These behaviors can be scripted or set in Solution Explorer > Mobile Themes > Menu Strip and Configuration > Environment Properties > Menu Strip, and/or set using F-Key options. (i.e. 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:

- Clear – clears the data entered for the current prompt
- Exit – exits the current process (same as F4 normally does if configured to do so).
- Options – allows you to associate a system action to this command. For example, launch the barcode scanner function of a mobile device.
- Refresh – refreshes the display screen
- Search – executes the OnSearch event if one exists for the prompt

- Show Menu – shows a secondary Menu if one exists for the prompt.
- Submit – means to enter the data appearing for the current 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
```

Enable

This command will enable or disable a button on the menu strip by referencing its Display value. The code should check if the value is an INT and if so, treat it as an index. Otherwise, it should look up the parameter by name.

Syntax: MenuStrip.Enable(vDisplay, [bEnable])

vDisplay (Variant) is the button's string name (sDisplay) or index number.

bEnable (Boolean) Set to True or False to enable or disable the button.

Example:

```
Menustrip.Enable("Search",false);
Menustrip.Enable(1,true);
```

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.

Refresh

This command will redraw the buttons on the Menustrip in case the buttons have been changed in code.

Syntax: MenuStrip.Refresh

RemoveItem

This command will remove a button from the menu strip by referencing its index. Removing an 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

Example:

```
MenuStrip.Show(False) ' hides the menu strip  
MenuStrip.Show(True) ' shows the menu strip
```

Mobile Device Extensions

These commands are used specifically on a mobile device installations. They provide methods for synchronizing with the server, sending and receiving data, queuing transactions on the host, etc. For queuing commands, the server's Transaction Manager and the client's Transaction Manager must be enabled.

For information about loading and executing COM objects on the Windows CE devices, refer to [Mobile Device Extensions - Windows CE](#).

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

GetTimeInfo

This command will retrieve the Offset value which is the difference of the device time and UTC in minutes. (The offset is NOT the difference between the device time and server time in UTC.)

You can call the function from an event such as a button click.

Syntax: [bOK =] Device.GetTimeInfo([vDeviceTime],[MinutesFromUTC])

bOK (Boolean) – returns a True if the mobile device successfully obtains the offset value.
 False if it fails.

vDateTime (Variant) – Returns the value in minutes.

Example of a pop up message box showing the device's time and UTC offset:

```
Dim DeviceTime AsDate
Dim MinutesFromUTC AsInteger

Device.GetTimeInfo(DeviceTime, MinutesFromUTC)

App MsgBox("Device Time = " & DeviceTime & ", which is " & MinutesFromUTC & " minutes from UTC")
```

DeviceTime can use the normal date formatting functions, but by default it will be formatted like "11/1/2017 8:52:12 AM".

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)
```

Id

This returns the name of a given device for Android, iOS, CE, and Windows Desktop clients. If testing with no client connection, it will return "RFgen Development Studio."

Syntax: Device.ID

Example:

```
Dim sMyDevice as String  
sMyDevice = Device.ID
```

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 where the file exists.

- For Windows CE, the path is the location of the Windows CE device.
- For Android and iOS, only .wav files are supported, and a copy of the source file MUST BE PLACED on the RFgen server under \ProgramData\RFgen51.

When the sound is played at runtime, the .wav file is first copied from the server to the client when the client connects to the server.

You can also use this command to vibrate an Android or iOS device. For details see [Vibrate](#).

Syntax:

Device.PlaySound(sPath), Device.PlaySound("Number. Vibrate")

sPath (String) the full path where the file is stored on the RFgen server

.wav The sound wav file to be played. The duration is the duration of the .wav file.

Example for playing sound on Android or iOS:

Device.PlaySound("\ProgramData\RFgen51\ERROR.WAV")

Example for playing sound on Windows CE:

Device.PlaySound("\Program Files\RFgen51\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 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)
```

ScanEnable

This command enables or disables the user's ability to scan a barcode. This command applies only for a subset of Intermec and Motorola handheld devices. One example might be to disable the scanner just before displaying a message box so that additional scans do not accidentally acknowledge the message box and the user is unaware there was an error causing future scans to fail as well.

Syntax: Device.ScanEnable(bScanOn)

bScanOn (Boolean) when set to True enables the laser scanner

Example:

```
Device.ScanEnable(True)
```

ScanTrigger

This command turns on the laser scanner for a subset of Intermec and Zebra (Motorola) handheld devices for Window CE. It is not supported on Android or iOS devices.

Syntax: Device.ScanTrigger = nTimeout

nTimeout (Long) is the timeout period in milliseconds. The default is 10,000 milliseconds (10 seconds).

Example:

```
Device.ScanTrigger = 10000
```

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)

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)

Shell

This function gives the user the ability to launch a program that is supported by the device's shell in the scripting environment. For example, it can be used to execute a program that opens a webpage in a specific browser.

Syntax: Device.Shell(ByVal File as String, ByVal Options As enDeviceShow)

File as String	This first parameter contains the name of the file or shell program stored on the device. Use closing quotes around the file name or command.
----------------	---

Options As

enDeviceShow This second parameter controls the window display for the program called in the first parameter "File as String".

Device_SHOWMAXIMIZED = Activates the window and displays it as a maximized window.

Device_SHOWMINIMIZED = Activates the window and displays it as a minimized window.

Example:

```
Device.Shell("https://www.rfgen.com",Device_SHOWMINIMIZED)
```

```
Device.Shell("https://www.rfgen.com",Device_SHOWMAXIMIZED)
```

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.Image.Bitmap = bImage
```

```
DB.SaveBitmap(sTableName, sField, bImage, sWhere)
```

Vibrate

This command option must be used with the Device.PlaySound command to vibrate an Android or iOS device. Not all devices support this option on Android or iOS devices. It is not supported on Windows CE devices.

Syntax:

```
Device.PlaySound("Number, Vibrate")
```

Number How long (duration) of a vibration in seconds. Default is 1 second.

Vibrate This command will vibrate the device.

Example

```
Device.PlaySound("1,Vibrate")
```

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)
```

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
```

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*)

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 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(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.

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 more 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 drop-down 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 Soft Input Panel (SIP).

Syntax: [bOK =] SIP.Show(bShow)

bOK (Boolean) – Optional – Returns True if the command was successful.

iShow (Long) – Sets the keyboard mode (no keyboard, show or hide) for the specified RFgen keyboard types.

Long (Long) The enumeration for the RFgen keyboard type. Values are:

KBRD_NONE The default value with SIP.Show to prevent a keyboard/soft input panel from displaying. Is not the same as hiding a keyboard.

KBRD_SIP Use this value with SIP.Show to display the keyboard/soft input panel.

KBRD_CUSTOM Use this value with SIP.Show to hide a keyboard/soft input panel.

KBRD_FULL_ALPHA

KBRD_FULL_ALPHA_NUMERIC

KBRD_ALPHA

KBRD_ALPHA_NUMERIC

KBRD_CUSTOM1

KBRD_CUSTOM2

KBRD_CUSTOM3

KBRD_CUSTOM4

Example:

```
Dim bOK as Boolean
bOK = SIP.Show (KBRD_FULL_ALPHA)
```

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.

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.

GeoGetAddress

This method returns an address converted from the longitude and latitude geo-coordinates. This extension only works with the Map control.

Syntax:

bValue = PromptID.Server.GeoGetAddress(sLat, sLng, sAddrs)

bValue: (Boolean) True returns a value if successful; False if it fails

sLat: (String) is the Latitude of the address

sLng: (String) is the Longitude of the address

sAddrs: (String) is the address converted from the Latitude and Longitude

Example:

```
sLat = "38.575764"  
sLng = "121.478851"  
mMapArea.Server.GeoGetAddress (sLat,sLng,sAddrs)
```

GeoGetDirections

This method to display the driving directions between a start and destination. This extension is used with the Map control

Syntax:

bVal = promptID.Server.GetDirections(sOrg, sDst, sRoads, [Map], [Duration], [Dist])

bVal: (Boolean) Returns True if the directions were retrieved; False if it failed.

sOrg: (String) is the origination or start of the route

sDst: (String) is the route destination

sRoads: (String Array) is the list of roads between origination and destination

[Map]: Optional - is an image of the directions

[Dur]: Optional - is the duration of the distance traveled by car

[Dist]: Optional – is the distance traveled

Example:

```
DIM sDir as String  
server.GeoGetDirections(txtOrig.Text, txtDest.Text, sRoads, True)
```

GeoGetLatLng

Use this method to return the Latitude and Longitude of a physical address. This extension is used with the Map control.

Syntax:

MapControlPromptID.map.

bVal = Server.GeoGetLatLng (sAddr, sLat, sLng)

bValue (Boolean) True means the conversion was retrieved; False means it failed.

sAddr: (String) is the address of the location

sLat: (String) is the latitude of the location

sLng: (String) is the longitude of the location

Example:

```
Server.GeoGetLatLng (txtAddress.Text, textLat.Text, textLng.Text, True)
```

GeoGetMap

This method returns a map representing the last Geo command from Google maps. For instance, if you execute GeoGetDirections, GeoGetMap will get a bitmap image representing those directions from Google Maps. This extension is only used with the Map control.

Syntax:

promptID.GeoGetMap

Example:

```
Server.GeoGetMap( )
```

GeoPlanRoute

This method returns the best route when there are multiple stops.

Use this method to obtain the most efficient driving route for multiple locations. This extension is used with the Map control.

Syntax:

bVal = Server.GeoPlanRoute(sOrg, sDst, sPoints,[Map])

bVal: (Boolean) True means the route was retrieved; False means it failed

sOrg: (String) is the origination or start of the planned route

sDst: (String) is the destination of the planned route

sPoints: (String Array) Points are an array of delivery addresses and once its returned, the points will contain the sorted addresses.

Map: (Boolean) Optional - returns True if the map from Google Maps was retrieved; False if it failed.

Example:

```
DIM sPlanRoute as String
server.GeoPlnRoute (txtOrig.Text, txtDest.Text, sPoints(), Map, True)
```

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 of 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,On-hand".

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, [vVPNNName], [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
vVPNNName	(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")
```

SyncAppsEx

This command is similar to Server.SyncApps except that it includes a parameter to set an error message.

Syntax: [bOK =] Server.SyncApps([vProfile])

bOK (Boolean) Optional – returns True or False for the success of the command

vProfile (String) Optional – is the name of the profile used to compare what is on the device with the server list of objects.

[ErrMsg] Optional – if an error occurs this parameter will be sent to a detailed error message

Examples:

```
Dim bOK As Boolean
```

```

Dim sErr As String
bOK = Server.SyncAppsEx("CEProfile1", sErr)
If bOK = False Then
    App MsgBox(sErr)
End If

```

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.

CheckBoxNoClickFromCode

This property is a global internal flag, when set to True, will prevent the Click event from executing when code is used to toggle the Checkbox. The Click event will still execute if a user toggles the Checkbox from the user interface. This would typically be used to initialize a form, check or uncheck the Checkboxes based on database data and not trigger the Click event.

Syntax: [bValue] = SYS.CheckBoxNoClickFromCode = bValue

bValue (Boolean) Optional – returns the state of this function

Example:

```
SYS.CheckBoxNoClickFromCode = True
```

```
bValue = SYS.CheckBoxNoClickFromCode
```

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.
dcConnString	(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
dcIgnoreCert	(Web)	Ignore Invalid Certificates Boolean check box field
dcIndexDatabase	(DB)	Index Selected Databases Boolean check box field for all database connector types.
dcIndexList	(DB)	Tables Specified for Indexing list of values all database connector types.
dcIsJDE	(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
dcLoginMode	(All)	Returns Login Mode option on the Login Options tab (0 = Automatic, 1 = Manual)
dcMsgServer	(SAP)	Message Server field
dcOnCCE	(SAP)	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 Conn 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")
```

SelectKeyboard

This function enables the end-user to open a specific keyboard.

Syntax: enKyBoardMode = SYS.SelectKeyboard

eKeyBoard = (enKeyBoardMode) The keyboard modes available.

kbAlph = Alphabetical keyboard, no punctuation or numbers

kbAlphNum = Alphabetical and Numeric keyboard, no punctuation symbols

kbCustom() = the customized keyboard created from the RFgen Mobile Development Studio > Soft Keyboards node

kbFullAlpha = the keyboard with all 26 alphabetical characters and punctuation symbols

kbFullAlpha = the keyboard with all 26 alphabetical characters, punctuation symbols, and numbers

kbNone = prevents/suppress the system keyboards. Disables use of any system keyboards

kbNumeric = numbers only keyboard

kbSIP = the Soft Input Panel keyboard (characters are arranged on keyboard to provide a smaller keyboard)

Example:

```
SYS.SelectKeyboard = kbFullAlpha
```

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.")
```

SetProcOptionFields

This function can be used to change the table and fields used by the JD Edwards JDEProcOpt objects. The default table and fields are: **F983051**, VRPID, VRVERS, and VRPODATA.

Syntax: Sys.SetProcOptionFields(ByVal F983051 As String, [ByVal VRPID], [ByVal VRVERS], [ByVal VRPODATA])

ByVal F983051 is the JD Edwards Table that contains the specified fields and/or processing options.

ByVal VRPID is field that contains Program IDs as a string

ByVal VRVERS is the field that contains the version of the JDE program as a string

ByVal VRPODATA is the field that contains the processing data option as binary (BLOB)

Examples:

'Reset the table and field names to the defaults

```
Sys.SetProcOptionFields("F983051", "VRPID", "VRVERS", "VRPODATA")
```

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)
```

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.

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:
	tmAllItems all macros in all queues, in the queue, failed, or completed
	tmCompleted all macros in all queues that were completed successfully
	tmFailed all macros in all queues that were completed unsuccessfully
	tmInProcess 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:

Field Name	Data Type	Description
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

Field Name	Data Type	Description
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
SYSNR	- 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, 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: bIsConnected = SM.Connected

bIsConnected (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 the host and waits for a 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 retrieve 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 retrieve 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 command 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 retrieve 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 number 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	chtTypeBar	chtTypeBubble
chtTypeCandle	chtTypeFilledRadar	chtTypeHiLo
chtTypeHiLoOpenClose	chtTypePie	chtTypePlot
chtTypePolar	chtTypeRadar	chtTypeStackingBar
chtTypeSurface		

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

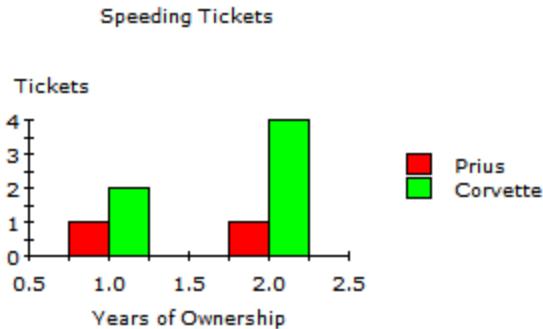
```

```

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

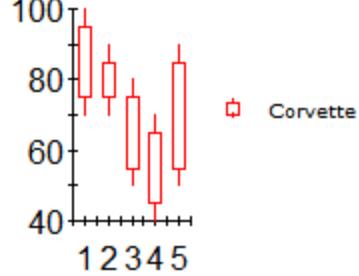
```

```
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
```

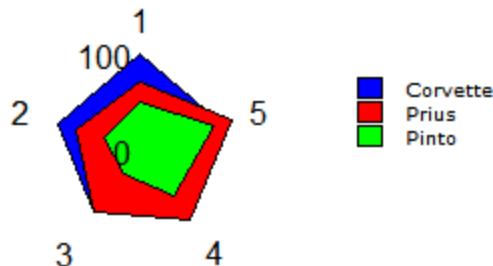
```
oCandle.AddDataPoint 4, 5, 5, 85
```

```
img1.Bitmap = oCandle.Image
```

```
End Sub
```

FilledRadar

Speeding Tickets



```
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

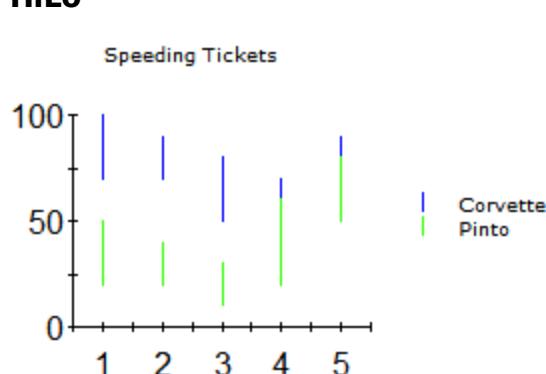
oFRadar.SeriesColor(2) = vbRed
oFRadar.SeriesLabel(2) = "Prius"
oFRadar.AddDataPoint 2, 1, 1, 70
oFRadar.AddDataPoint 2, 2, 2, 70
oFRadar.AddDataPoint 2, 3, 3, 80
oFRadar.AddDataPoint 2, 4, 4, 90
oFRadar.AddDataPoint 2, 5, 5, 100

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

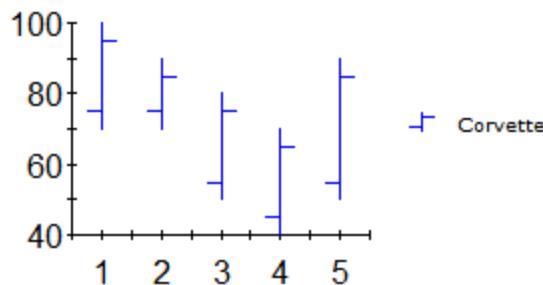
```

```
img1.Bitmap = oHiLo.Image
```

```
End Sub
```

HiLo OpenClose

Speeding Tickets



```

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

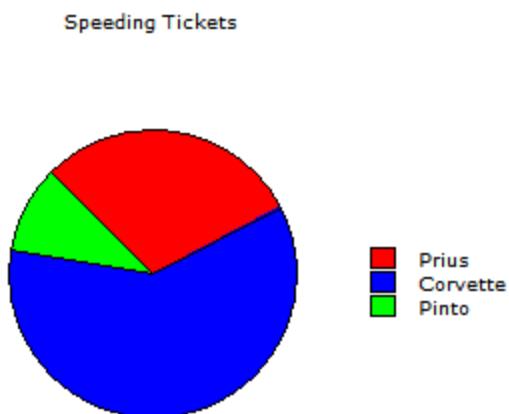
' Low Open points
oHiLoOC.AddDataPoint 3, 1, 1, 75
oHiLoOC.AddDataPoint 3, 2, 2, 75
oHiLoOC.AddDataPoint 3, 3, 3, 55
oHiLoOC.AddDataPoint 3, 4, 4, 45
oHiLoOC.AddDataPoint 3, 5, 5, 55

' 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



```
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

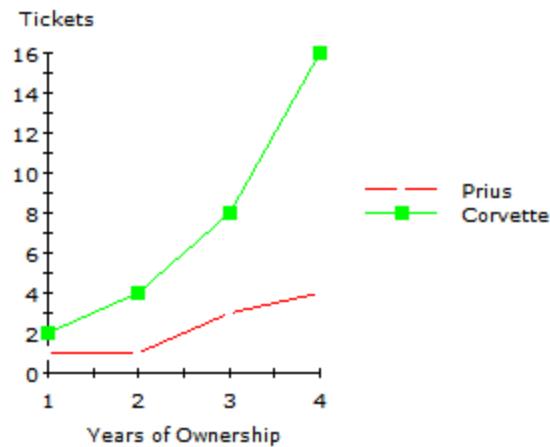
img1.Bitmap = oPie.Image

End Sub

```

Plot

Speeding Tickets



```
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
```

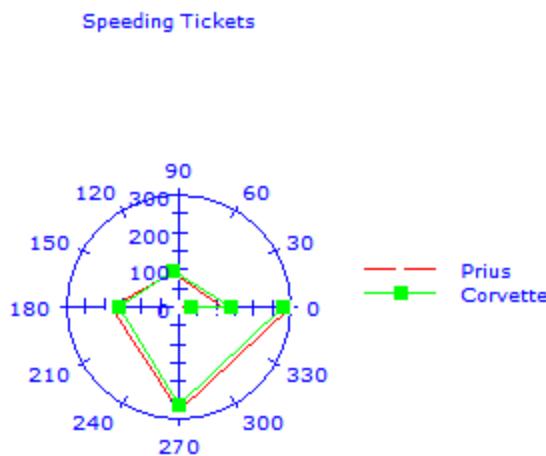
```
oPlot.Height = 300
oPlot.Header = "Speeding Tickets"
oPlot.Type = chtTypePlot
oPlot.AxesLabel("X") = "Years of Ownership"
oPlot.AxesLabel("Y") = "Tickets"
oPlot.AxesTitleFont "X", sFontName, dFontSize
oPlot.AxesTitleFont "Y", sFontName, dFontSize
oPlot.AxesFont "X", sFontName, dFontSize
oPlot.AxesFont "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



```

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

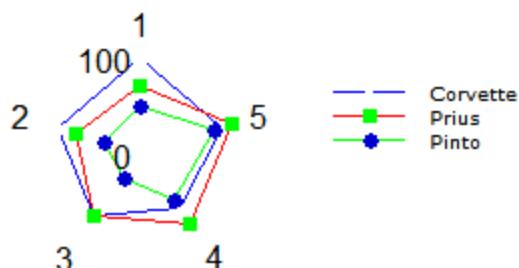
'Corvette
oPolar.SeriesColor(2) = vbGreen
oPolar.SeriesLabel(2) = "Corvette"
oPolar.AddDataPoint 2, 1, 30, 30
oPolar.AddDataPoint 2, 2, 30, 140
oPolar.AddDataPoint 2, 3, 140, 100
oPolar.AddDataPoint 2, 4, 10, -160
oPolar.AddDataPoint 2, 5, 250, 260
oPolar.AddDataPoint 2, 6, 10, 280

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

```

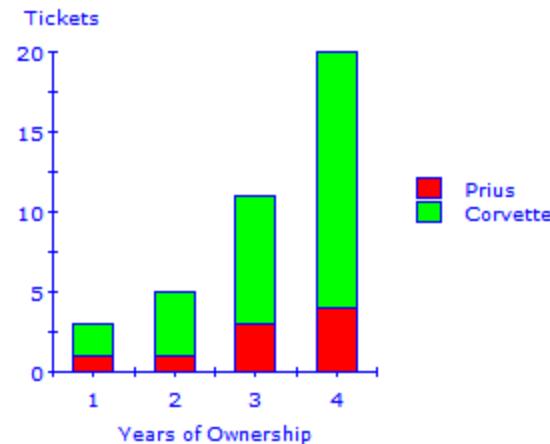
```

img1.Bitmap = oRadar.Image
End Sub

```

StackingBar

Speeding Tickets



```

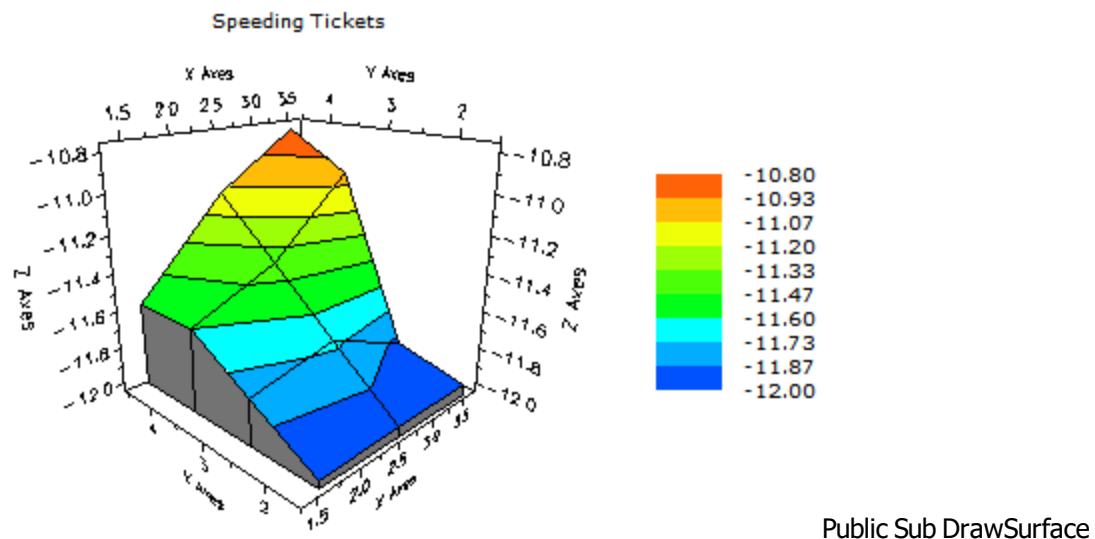
Public Sub DrawStackingBar
On Error Resume Next
'
Dim oSBar As New Chart
Const sFontName As String = "Verdana"
Const dFontSize As Double = 6.5

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 SearchList 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 ListBox is different in that it displays selected items in the original prompt space allocated for the data in the application.

Alternatively, a SearchList can also be designed to display selected items in a "freestyle format". For example, the data entry device screen is replaced by a series of panels as opposed to a traditional table format.

SearchList

This object is an advanced method of creating a scrolling list of items or a scrolling list of panels containing child objects 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: oList.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.AddItem(123, "1st Col Description", "2nd Col Description")
```

BindControl

The method connects a control within a panel to a column number. See also *SetBind*.

Syntax: oMyList.BindControl(vControl, vColumn)

vControl (String) The value to be returned when this is chosen.

vColumn (String) The value to be returned when this is chosen.

Example:

```
Dim oMyList As New SearchList
oList.SQL = "select * from RFgen.cyclecounts"
' SetBind() connects the SearchList to a panel within a form
oMyList.SetBind("SearchListTest", "Panel1")
' BindControl() connects a control within the panel to a column number
oMyList.BindControl("Label1", 1)
oMyList.BindControl("Label2", 2)
oMyList.BindControl("Label3", 3)
oMyList.BindControl("Label4", 4)
oMyList.ShowList
```

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.

Example:

```
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.

Example:

```
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 forecolor 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.

Example:

```
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 within the Searchlist object.

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: BottomCenter, BottomLeft, BottomRight, CenterCenter, CenterLeft, CenterRight, TextCenter, TextLeft, TextRight., TopCenter, TopLeft, TopRight

Syntax: oMyList.Column(x).Align = enValue

enValue (enColAlign) an enumeration that contains BottomCenter, BottomLeft, BottomRight, CenterCenter, CenterLeft, CenterRight, TextCenter, TextLeft, TextRight., TopCenter, TopLeft, TopRight

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: lValue = oMyList.Column(x).BackColor1

lValue (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.

Example:

`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.

Example:

`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.

Example:

`oMyList.Column(2).DisplayOnly = True`

Column(x).Expanded

This property is intended for the Tree control only and is not to be used with the SearchList object.

Column(x).FontSize

This property accesses the font size of the entire search list display. The Default (set in the Theme) is 10.

Syntax: oMyList.Column(x).FontSize = lValue

Alternate: lValue = oMyList.Column(x).FontSize

lValue (Long) is the font size for the search list display

Example:

```
oMyList.Column(2).FontSize = 16
```

Column(x).ForeColor

This property accesses the column's fore color property associated with the specified column.

Syntax: oMyList.Column(x).ForeColor = vValue

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

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.

Column(x).ImageHeight

This property sets the images in this column to a specific height.

Syntax: oMyList.Column(x).ImageHeight = IValue

Alternate: IValue = oMyList.Column(x).ImageHeight

IValue (Long) is the pixel height size for images in this column

Example:

oMyList.Column(1).ImageHeight = 5

Column(x).ImageWidth

This property sets the images in this column to a specific width.

Syntax: oMyList.Column(x).ImageWidth = IValue

Alternate: IValue = oMyList.Column(x).ImageWidth

IValue (Long) is the pixel width size for images in this column

Example:

oMyList.Column(1).ImageWidth = 5

Column(x).Indent

This property is intended for the Tree control only and is not to be used with the SearchList object.

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.

Example:

```
oMyList.Column(2).Italic = True
```

Column(x).MarginBottom

This property pads the bottom of all the cells in a specified column. It also contributes to the overall height of the entire row.

Syntax: oMyList.Column(x).MarginBottom = lValue

Alternate: lValue = oMyList.Column(x).MarginBottom

lValue (Long) the padding in pixels between the contents of a cell and the bottom of the cell.

Example:

```
oMyList.Column(1).MarginBottom = 2
```

Column(x).MarginLeft

This property pads the left of all the cells in a specified column.

Syntax: oMyList.Column(x).MarginLeft = lValue

Alternate: lValue = oMyList.Column(x).MarginLeft

lValue (Long) the padding in pixels between the contents of a cell and the left of the cell.

Example:

```
oMyList.Column(1).MarginLeft = 2
```

Column(x).MarginRight

This property pads the right of all the cells in a specified column.

Syntax: oMyList.Column(x).MarginRight = lValue

Alternate: lValue = oMyList.Column(x).MarginRight

lValue (Long) the padding in pixels between the contents of a cell and the right of the cell.

Example:

oMyList.Column(1).MarginRight = 2

Column(x).MarginTop

This property pads the top of all the cells in a specified column. It also contributes to the overall height of the entire row.

Syntax: oMyList.Column(x).MarginTop = lValue

Alternate: lValue = oMyList.Column(x).MarginTop

lValue (Long) the padding in pixels between the contents of a cell and the top of the cell.

Example:

oMyList.Column(1).MarginTop = 2

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.

Example:

oMyList.Column(1).ScaleDecimals = 2

Column(x).Style

This property gets or sets the type of column. The values are Text, Image, and Check box.

Syntax: oMyList.Column(x).Style = enStyle

Alternate: enStyle = oMyList.Column(x).Style

enStyle (enColumnStyle) Contains ColumnStyleCheck, ColumnStyleImage, and ColumnStyleText

Examples:

```
oMyList.Column(1).Style = ColumnStyleCheck  
oMyList.Column(1).Style = ColumnStyleText
```

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.

Example:

```
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.

Example:

```
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.

Example:

```
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`

SetBind

This method connects the SearchList object to a panel within a form. See also, *BindControl*.

Syntax: `List.Set(vForm, vControl)`

`vForm` (String) The value to be returned when this is chosen.

`vControl` (String) The value to be returned when this control is chosen.

Example:

Dim oMyList As New SearchList

`oList.SQL = "select * from RFgen.cyclecounts"`

' SetBind() connects the SearchList to a panel within a form

`oMyList.SetBind("SearchListTest", "Panel1")`

' BindControl() connects a control within the panel to a column number

```
oMyList.BindControl("Label1", 1)
oMyList.BindControl("Label2", 2)
oMyList.BindControl("Label3", 3)
oMyList.BindControl("Label4", 4)

oMyList.ShowList
```

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

Database Table – used to generate Insert SQL statements

Stored Procedures – found in ODBC compliant databases

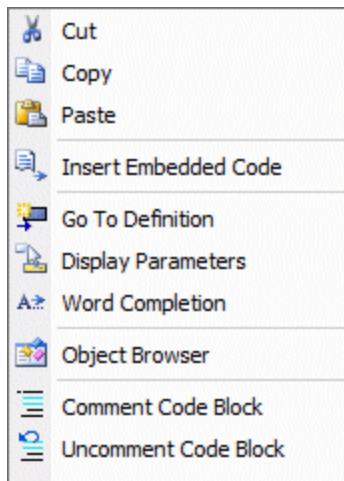
Transaction Macros – to be queued or executed

Vocollect Recordsets – defined Vocollect resources

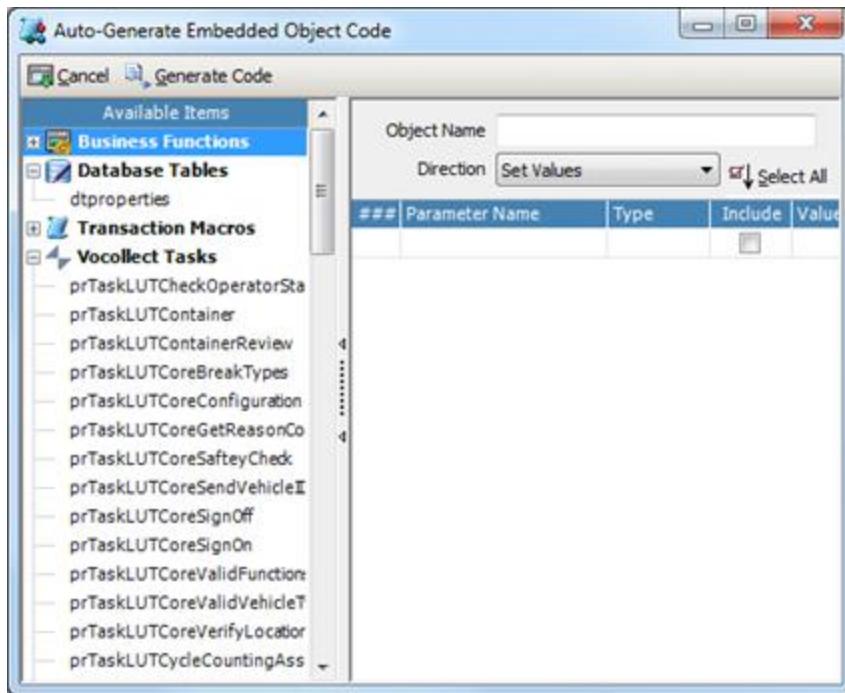
Web Service Objects – based on the Web Service Connector

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".



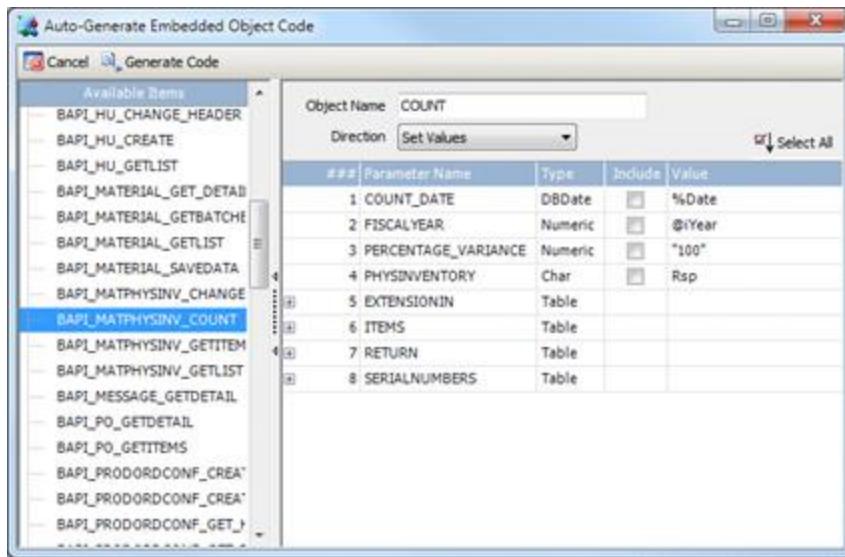
Next, select the type of object to embed from the list on the left.



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.

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.ColumnNames("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: [Value =] emProc.ExecuteMethod(Name)

Value (Boolean) Optional – contains a True / False based upon its success.

Name (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

Example:

```
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")
```

SetKeyFields

For a given table, this function will contain the list of keys to be used in the internal Where clause when updating a table.

Syntax: [bValue] = emProc.SetKeyFields(ParamArray)

bValue (Boolean) Optional – returns the success or failure of the command

ParamArray (param array of Variants) the name of each key for the table in quotes and separated by a comma

Example:

```
Dim emCUSTOMERS As New EmbeddedProc  
emCUSTOMERS.Name = "CUSTOMERS"  
emCUSTOMERS.DataSource = "RFSample"  
emCUSTOMERS.Param("CUSTID") = 1001  
emCUSTOMERS.Param("HOSTID") = "A1"
```

```

emCUSTOMERS.Param("NAME") = "Microsoft"
emCUSTOMERS.SetKeyFields("CUSTID", "HOSTID")
emCUSTOMERS.ExecuteMethod("update")

```

JDE Processing Option

This function is available for JD Edwards systems only, and requires the downloaded JDE processing options to the form where you plan to use this object.

JDEProcOpt

Use this object to retrieve and interact with processing options for specified versions of the JDE application.
(Requires the JDE Processing Option be downloaded to your form.)

The properties of the function are values sent and received from the JDE Processing Options database schema.

Count

The number of processing option values in the JDEProcOpt object.

Syntax: Object value, Object property = JDECount

Example

```

Dim oJDE As New JDEProcOpt
Dim iCnt As Integer
oJDE.ProgramId = "P4113"
oJDE.Version = "ZJDE0001"
iCnt = oJDE.Count

```

This will return total count 19

#	Parameter Name	Page	Script	Description
1	DocumentType			[0.0] DocumentType
2	DefaultFROMPrimLocation	1		[0.1] DefaultFROMPrimLocation
3	DefaultTOPrimLocation	2		[0.2] DefaultTOPrimLocation
4	ProtectCosts	2		[2.0] ProtectCosts
5	SummaryMode	2	1	[2.1] SummaryMode
6	AllowHeldLots	2	2	[2.2] AllowHeldLots
7	AllowOverQtyAvailable	2	4	[2.4] AllowOverQtyAvailable
8	JournalEntriesVersion	1	1	[1.1] JournalEntriesVersion
9	ItemLedgerVersion	1	2	[1.2] ItemLedgerVersion
10	OutInteroperabilityType	3		[3.0] OutInteroperabilityType
11	AgreementAssignProcess	4		[4.0] AgreementAssignProcess
12	PODefaultLotStatus	2	5	[2.5] PODefaultLotStatus
13	LotGroup	2	3	[2.3] LotGroup
14	LPNGenerationMethod	5		[5.0] LPNGenerationMethod
15	BuildStructure	5	1	[5.1] BuildStructure
16	LicenseRateWindow	5	2	[5.2] LicenseRateWindow
17	SearchAndSelect		3	[0.3] SearchAndSelect
18	PCW10Version	1	3	[1.3] PCW10Version
19	ProductionNoConsumption	2	6	[2.6] ProductionNoConsumption

ParamName

The name of a processing option at a specified index.

Syntax:

Object value, Object property = JDEparamname

Example:

```
Dim sDOCTYPE As String
Dim oJDE As New JDEProcOpt
oJDE.ProgramID = "P4113"
oJDE.Version    = "ZJDE0001"
sDOCTYPE = oJDE.ParamName(1)
```

The result of sDOCTYPE will be "Documenttype"

P4113.ZJDE0001				
#	Parameter Name	Page	SeqNo	Description
1	Documenttype			[0.0] Documenttype
2	DefaultF0OMPrimLocation	1	1	[0.1] DefaultF0OMPrimLocation
3	DefaultT0PrimLocation	2	2	[0.2] DefaultT0PrimLocation
4	ProtectCosts	2		[2.0] ProtectCosts
5	SummaryMode	2	1	[2.1] SummaryMode
6	AllowHeldLots	2	2	[2.2] AllowHeldLots
7	AllowOverQtyAvailable	2	4	[2.4] AllowOverQtyAvailable
8	JournalEntriesVersion	1	1	[1.1] JournalEntriesVersion
9	ItemLedgerVersion	1	2	[1.2] ItemLedgerVersion
10	OutInteroperabilityType	3		[3.0] OutInteroperabilityType
11	AgreementAssignProcess	4		[4.0] AgreementAssignProcess
12	PODefaultLotStatus	2	5	[2.5] PODefaultLotStatus
13	LotGroup	2	3	[2.3] LotGroup
14	LPNGenerationMethod	5		[5.0] LPNGenerationMethod
15	BuildStructure	5	1	[5.1] BuildStructure
16	LicensePlateWindow	5	2	[5.2] LicensePlateWindow
17	SearchAndSelect		3	[0.3] SearchAndSelect
18	PCW10Version	1	3	[1.3] PCW10Version
19	ProductionNoConsumption	2	6	[2.6] ProductionNoConsumption

ProgramId

The programID of the JDE processing option.

Syntax:

Object value, Object property = JDEprogramID

Example:

```
Private moProcOPT As New JDEProcOpt
```

```
' Get Proc.Opt. Version from Menu
```

```
msPgm  = App.GetValue("Pgm")
```

```

msVersion = App.GetValue("Vers")
moProcOPT.ProgramId = "P4113"
moProcOPT.Version = "ZJDE0001"

msDOCTYPE = moProcOPT.Value("Documenttype")

' get DocType from Proc.Opt.

msDOCTYPE = GetProcOpt(msPgm,msVersion,"1;1",sHeader)

```

TemplateId

The ID of the template used for the Processing Object.

Syntax: Object value, Object property = JDETemplateID

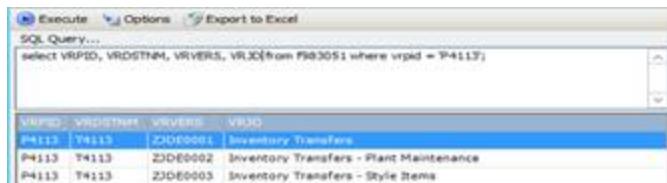
Example

```

Dim sTemplateId As String
Dim oJDE As New JDEProcOpt
oJDE.ProgramID = "P4113"
oJDE.Version = "ZJDE0001"
sTemplateId = oJDE.TemplateId

```

The result of sTemplateId is 'T4113'



A screenshot of a software interface showing a SQL query window. The window has tabs for 'Execute', 'Options', and 'Export to Excel'. The SQL query is:

```
SQL Query...
select VRPBD, VRDSTNM, VRVERS, VRD0 from F903051 where VRPBD = 'P4113';
```

The results table has columns: VRPBD, VRDSTNM, VRVERS, VRD0. The data is:

VRPBD	VRDSTNM	VRVERS	VRD0
P4113	T4113	ZJDE0001	Inventory Transfers
P4113	T4113	ZJDE0002	Inventory Transfers - Plant Maintenance
P4113	T4113	ZJDE0003	Inventory Transfers - Style Items

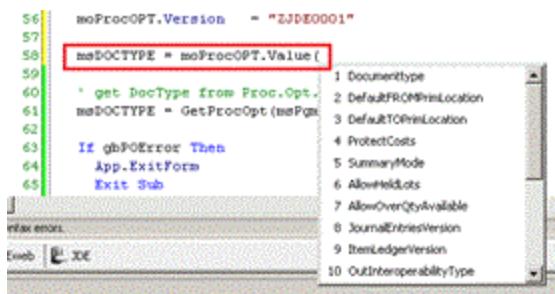
Value

This property returns the value specified in the process option table.

Syntax:

Object value, Object property = ProcOPT.value()

Note: To view Value property parameters, hit CTRL+Space after you open the parenthesis of a value property. Alternatively, you can right-click in the scripting window, and select "embed code" which gives you an option to generate some of this code.

**Example:**

```
Private moProcOPT As New JDEProcOpt
```

```
' Get Proc.Opt. Version from Menu
```

```
msPgm = App.GetValue("Pgm")
```

```
msVersion = App.GetValue("Vers")
```

```
moProcOPT.ProgramId = "P4113"
```

```
moProcOPT.Version = "ZJDE0001"
```

```
msDOCTYPE = moProcOPT.Value("Documenttype")
```

```
' get DocType from Proc.Opt.
```

```
msDOCTYPE = GetProcOpt(msPgm,msVersion,"1;1",sHeader)
```

Version

Syntax: Object value, Object property = JDEversion

Example:

```
' Get Proc.Opt. Version from Menu
```

```
msPgm = App.GetValue("Pgm")
```

```
msVersion = App.GetValue("Vers")
```

```
moProcOPT.ProgramId = "P4113"
```

```
moProcOPT.Version = "ZJDE0001"
```

```
msDOCTYPE = moProcOPT.Value("Documenttype")
```

```
' get DocType from Proc.Opt.
```

```
msDOCTYPE = GetProcOpt(msPgm,msVersion,"1;1",sHeader)
```

Smtp Extension

This language extension gives the user the ability to send e-mails from the scripting environment. In the Application Services configuration window is a place where some defaults may be set so the script environment doesn't need to repeat static information.

Attach

Specify a path and a file you wish to attach. If you would like to attach more than one file just separate the path + file names with a comma like:

"C:\Log.txt, C:\DebugLog.txt"

Syntax: Smtp.Attach = sPath

sPath (String) is the path and the name of the file to be attached to the outgoing message.

Example – see Smtp.Send

Bcc

This property is used to specify a recipient to be "Blind Carbon Copied".

Syntax: Smtp.Bcc = sEmail

sEmail (String) is the email address of the recipient

Example – see Smtp.Send

Cc

This property is used to specify a recipient to be "Carbon Copied".

Syntax:

Smtp.Cc = sEmail

sEmail (String) is the email address of the recipient

Example:

See Smtp.Send

Clear

This property is used to clear the contents of a smtp object.

Syntax: Smtp.Clear()

Example – see Smtp.Send

From

This property is used to specify the sending person's email address.

Syntax: `Smtp.From = sEmail`

`sEmail` (String) is the email address of the sender

Example – see `Smtp.Send`

Host

This property is the host email server that will process the request. In the Application Services configuration there is a place to default this value so it isn't necessary in the script.

Syntax: `Smtp.Host = sServer`

`sServer` (String) is the IP address or DNS name of the outgoing mail server

Example – see `Smtp.Send`

Message

This property contains the message body of the email. To include Carriage Return + Line Feed (Enter keys) in the text use the convention /r/n in the text itself.

Syntax: `Smtp.Message = sBody`

`sBody` (String) is all the text in the body of the email

Example – see `Smtp.Send`

Port

This property is the host email port that will be used to communicate with the host email server. In the Application Services configuration there is a place to default this value so it isn't necessary in the script. Be sure that this port isn't being blocked by firewalls or virus protection.

Syntax: `Smtp.Port = nPort`

`nPort` (Long) is the port number of the outgoing email server

Example – see `Smtp.Send`

Send

This method submits the Smtp properties to the mail server for processing. Note that the Host and Port properties are optional if already specified in the Application Services configuration screen.

Syntax: `Smtp.Send()`

Example:

```
Smtp.Host = "smtp.company.com"  
Smtp.Port = 25  
Smtp.From = "MySelf@company.com"  
Smtp.Cc = "User@company.com"  
Smtp.Bcc = "Boss@company.com"  
Smtp.To = "Ex-Employee@company.com"  
Smtp.Subject = "I'm so sorry"  
Smtp.Message = "Good bye./r/nCall me!"  
Smtp.Attach = "C:\PinkSlip.PDF, C:\GetWell.PDF"  
Smtp.Send  
Smtp.Clear()
```

Subject

This property contains the subject of the email.

Syntax: Smtp.Subject = sText

sText (String) is the subject of the email

Example – see Smtp.Send

To

This property is used to specify the recipient's email address.

Syntax: Smtp.To = sEmail

sEmail (String) is the email address of the recipient

Example – see Smtp.Send

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 commands 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
sMacorName = oData.Param(6)
sMacorName = 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 3rd 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 "123AAAAAA" 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 3rd 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 1st record and 3rd 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"

RFField

The RFField function searches a string from the right to extract a sub-string by using a specified delimiter character.

Syntax: vVAL= RFField(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 = RFField(VAR, "|", 1) returns VAL = '333'

VAL = RFField(VAR, "|", 3) returns VAL = '111'

Note: If StartField = 1, then the RFField 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 values as well. They are sck-TCPProtocol 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
```

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
```

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
```

EndPoint

The Endpoint property returns or sets a string for the WebServices SOAP endpoint. If a EndPoint is not specified the language extension will use the first Endpoint that is Returned as a WEB connection.

Syntax: vValue = oWeb.EndPoint

Alternate: oWeb.EndPoint = vValue

vValue (Variant) the number or name of the data connection object.

Examples:

```
Dim oWeb As Web  
oWeb.EndPoint = "http://www.myEndPoint.com"
```

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") = """http://xml.namespaces.xerox.com/im /xip/services/xcllmswebservice/wsdl/getDistributionRefurbInfo"""
```

InParam

For future use.

OutParam

For future use.

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 command is OBSOLETE. It relates specifically to stored procedures and has been replaced with the Embedded Procedures structure.

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 command is OBSOLETE. It relates specifically to stored procedures and has been replaced with the Embedded Procedures structure.

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 command is OBSOLETE. It relates specifically to stored procedures and has been replaced with the Embedded Procedures structure.

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.

[StoredProcedure Properties and Methods](#)

CommandText

Sets or returns a value containing a provider command such as a stored procedure call.

Syntax: `StoredProc.CommandText = sValue`

Alternate: `sValue = StoredProc.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:

`StoredProc.CommandTimeout = nValue`

Alternate: `nValue = StoredProc.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>
<code>dbCmdText</code>	Evaluates CommandText as a textual definition of a command.
<code>dbCmdStoredProc</code>	Evaluates CommandText as a stored procedure that will return records.
<code>dbCmdExecuteNoRecords</code>	Evaluates CommandText as a stored procedure that will return no records.
<code>dbCmdUnknown</code>	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

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

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 Object

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>
dbParamInput	Default, indicates an input parameter
dbParamOutput	Indicates an output parameter
dbParamInputOutput	Indicates both an input and output parameter
dbParamReturnValue	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.

The files are stored in the %AppData% / ProgramData / RFgen50 or %AppData% / ProgramData / RFgen51 folder. The parameters for your configuration settings belong under the [Environment] global setting.

RFgen.ini

The RFgen.ini file provides configuration parameters for the following features. In all cases, the headings and settings are case sensitive.

Note: The RFgen.ini file must be created using Notepad and stored in the RFgen directory under ProgramData\RFgen5.0 or ProgramData\RFgen5.1 in order for it to work.

LoadBalanced Server Parameters

[Environment]

LBSRedirect=0	This will allow servers to share client licenses but will prevent the clients from being redirected.
LBSLog=1	This will allow the server to log some information from load balancing. It should be used in conjunction with [Trace].
RFSVR510=1	This will cause the server to start generating a log file in the c:\ProgramData\RFgen51\Logs directory. This is currently limited to include load balancing information from the LBSLog.

Routing Queues Between Servers and Mobile Clients

[Environment]

MobileQueue=Queue1,Queue2,Queue3,...

RFgen configuration settings are used if a server needs to push queued items from a local RFgen server to a different RFgen server for processing. Mobile devices also have scenarios where they need to push queued items to a RFgen server, and the "Server.SendQueue" transfers items to the RFgen server. If you have multiple servers and one of them is where the data is collected and queued, but the other server is the one that has the ERP connector, the "MobileQueue" command performs the same transfer. Here is an example. Server A receives the queued transactions from the handheld but doesn't have the ERP connector to process them. Server B has the ERP connector. Server A does not need any queue configured at all for a queue name identified in the MobileQueue setting. Server A keeps the records internally. Server B must have a queue configured and set to process the name mentioned in the MobileQueue setting. On a timed event, most likely, Server A must perform a Server.SendQueue, just like a handheld would, and the contents of the internal queue will be sent to Server

B. This concept eliminates the need for Server A to have a data connector to Server B for the Server.MoveQueue command.

Purge Data From Reports

[Environment]

PurgeAfterNbrDays=30 This command purges the performance data that is visible from Reports > Performance Monitoring.

[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.

Override Transaction Macro Internal Timeout Limit

[Environment]

TMTtimeout=nnn nnn = number of seconds

TMTtimeout allows a Transaction Macro to execute longer than the built-in, internal limit of 30-seconds. The built-in timer terminates any macro that runs longer than 30 seconds. Even though a macro is terminated, it stays in the queue. Should a long running macro get half done, terminate, and re-run, you could end up with a continuous loop. To prevent this problem, use the TMTout command to lengthen the timeout period.

You might also set the timeout to 120 seconds or higher if downloading a large volume of data from a server to a mobile device. After making this change the services should be restarted.

Reset Transaction Queues

[Environment]

TranLimit=n or TranLimit=queue name

The TranLimit command enables the Transaction Manager to process a specified number of transactions in the queue before it resets the queue and recreates the connection. (A reset destroys and restarts the client process in charge of the processing queue.) This allows the system to release the memory for the process. This includes thin client device connections, ODBC, SM, and ERP connections.

You can either specify a limit for all queues by using the command TranLimit=n by itself, or as a limit for individual queues by queue name. If multiple queue limits are defined, only the queue assigned the limit will reset if its limit is triggered.

Note: Pooled data connections ARE NOT reset. See the ResetUDCx=y command in the online help for information on resetting pooled connections.

Example:

[Environment]
TranLimit =1000

Example:

[Environment]
TranLimit=RFQueue01, TWDXX0100, TCCQueue200

[RFQueue01]
TranLimit=100

[TWDXX0200]
TranLimit=200

[TCCQueue200]
TranLimit=200

Reset Pooled Connections

[Environment]
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.

The ResetUDCx=y command will make 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.

Make DB2 Tables Visible

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

Check JDE Connection Before Executing a Business Function

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 cal
CompanyKey=00001	parameter for X0010 call
DocumentType=IA	parameter for X0010 call

Reset JDE Child Processes

RFgen will monitor the JDE.log file for designated strings and then perform a series of resets 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.

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").

Example:

```
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, OK  
2, ,to Activate the component and correct the 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 General Packet Radio Service (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
or
C:\Users\username\AppData\Roaming\RFgen51
```

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\rfgen51\rfgen.xdb
```

WWB.NET: Overview

WinWrap Basic (WWB) is a third-party resource and component that enables RFgen to provide its customers with additional scripting features when coding an application in the Mobile Development Studio. The WWB help is provided for .NET and COM.

WWB.NET	Use '#Language "WWB.NET" for Visual Basic.NET(TM) compatibility.
Declaration	'#Language, '#Reference, '#Uses, Attribute, Class Module, Code Module, Const, Declare, Delegate, Dim, Enum...End Enum, Event, Function...End Function, Imports, Object Module, Option, Private, Property...End Property, Public, ReDim, Static, Structure...End Structure, Sub...End Sub, WithEvents.
Data Type	Any, Boolean, Byte, Char, Date, Decimal, Double, Integer, Long, Object, SByte, Short, Single, String, UInteger, ULong, UShort, obj type, user dialog, user enum, user structure.
Assignment	Assign: (=, +=, -=, *=, /=, \=, ^=, <<=, >>=), Erase
Flow Control	AddHandler, Call, CallByName, Do...Loop, End, Exit, For...Next, For Each...Next, GoTo, If...ElseIf...Else...End If, MacroCheck, MacroCheckThis, MacroRun, MacroRunThis, ModuleLoad, ModuleLoadThis, RaiseEvent, RemoveHandler, Return, Select Case...End Select, Stop, While...End While.
Error Handling	Err, Error, ErrorToString, On Error, Resume, Throw, Try, Using
Conversion	CBool, CByte, CChar, CDate, CDec, CDbl, CInt, CLng, CObj, CSByte, CShort, CSng, CStr, CType, CUInt, CULng, CUShort, CVErr, DirectCast, TryCast, Val.
Variable Info	IsArray, IsDate, IsDBNull, IsError, IsNothing, IsNumeric, IsReference, LBound, SystemTypeName, TypeName, UBound, VarType, VbTypeName.
Constant	False, Nothing, True, Win16, Win32, Win64.
Math	Abs, Atn, Cos, Exp, Fix, Int, Log, Randomize, Rnd, Round, Sgn, Sin, Sqr, Tan.
String	Asc, AscW, Chr, ChrW, Format, GetChar, Hex, InStr, InStrRev, Join, LCASE, Left, Len, LSet, LTrim, Mid, Oct, Replace, Right, RSet, RTrim, Space, Split, Str, StrComp, StrConv, StrDup, StrReverse, Trim, UCASE.
Object	CreateObject, GetObject, Me, With...End With.
Time/Date	DateAdd, DateDiff, DatePart, DateSerial, DateValue, Day, Hour, Minute, Month, MonthName, Now, Second, Timer, TimeSerial, TimeValue, Weekday, WeekdayName, Year.
File	ChDir, ChDrive, CurDir, Dir, EOF, FileAttr, FileClose, FileCopy, FileDateTime, FileLen, FileOpen, FreeFile, Get, GetAttr, Input,

	InputString, Kill, Line Input, LineInput, Loc, Lock, LOF, MkDir, Print, PrintLine, Put, Reset, Rename, RmDir, Seek, Seek, SetAttr, Unlock, Write, WriteLine.
User Input	Dialog, GetFilePath, InputBox, MsgBox, ShowPopupMenu
User Dialog	Begin Dialog...End Dialog, CancelButton, CheckBox, ComboBox, DropDownListBox, GroupBox, ListBox, MultiListBox, OKButton, OptionButton, OptionGroup, Picture, PushButton, Text, TextBox.
Dialog Function	DialogFunc, DlgControlId, DlgCount, DlgEnable, DlgEnd, DlgFocus, DlgListBoxArray, DlgName, DlgNumber, DlgSetPicture, DlgText, DlgType, DlgValue, DlgVisible.
DDE	DDEExecute, DDEInitiate, DDEPoke, DDERequest, DDETerninate, DDETerninateAll.
Settings	DeleteSetting, GetAllSettings, GetSetting, SaveSetting
Miscellaneous	AboutWinWrapBasic, AppActivate, Assign, Attribute, Beep, CallersLine, Choose, Clipboard, Command, Decode64, Decode64B, Decrypt64, Decrypt64B, Debug.Print, DoEvents, Encode64, Encode64B, Encrypt64, Encrypt64B, Environ, Eval, IIf, GetLocale, KeyName, Mac-roDir, QBColor, Rem, RGB, SendKeys, SetLocale, Shell, Wait.
Operator	Operators: +, -, ^, *, /, \, Mod, +, -, <<, >>, &, =, <>, <, >, <=, >=, Like, New, TypeOf, Not, And, AndAlso, Or, OrElse, Xor, Eqv, Imp, Is, IsNot, GetType, AddressOf.

#Language Special Comment

Syntax	'#Language "WWB.NET"
Group	Declaration
Description	Selects Visual Basic.NET(TM) compatibility.

If this special comment is not included then the macro/module is parsed using '#Language "WWB-COM" rules without the enhancements.

Language reference by **group**:

- Declaration, Data Type, Assignment
- Flow Control, Error Handling
- Conversion, Variable Info, Constant
- Math, String, Object, Time/Date, File
- User Input, User Dialog, Dialog Function
- DDE, Settings, Miscellaneous
- Operator

These VB.NET keywords are recognized, but not supported: FileGet, FileGetObject, FilePut, FilePutObject, Inherits, Interface, MyBase, MyClass, MustInherit, MustOverride, Namespace, Narrowing, NotInheritable, NotOverridable,

Operator, Overloads, Overridable, Overrides, Partial, Protected, SyncLock, Widening.

Version Available for WinWrap Basic version 9.1 or higher. (Not supported in Windows CE Applications.)

#Reference Special Comment

Syntax

```
'#Reference {uuid}#vermajor.verminor#lcid#[path[#name]]  
-or-  
'#Reference #assembly
```

Description The Reference comment indicates that the current *macro/module* references the COM type library (or .NET assembly) identified. Reference comment lines must be the first lines in the macro/module (following the global **Attributes**). Reference comments are in reverse priority (from lowest to highest). The IDE does not display the reference comments.

Parameter	Description
uuid	Type library's universally unique identifier.
vermajor	Type library's major version number.
verminor	Type library's minor version number.
lcid	Type library's locale identifier.
path	Type library's path.
name	Type library's name.
assembly	Fully qualified assembly name.

Examples

```
'#Reference {00025E01-0000-0000-C000-000000000046}#4.0#0#C:\PROGRAM  
FILES\COMMON FILES\MICROSOFT SHARED\DAO\DAO350.DLL#Microsoft DAO  
3.5 Object Library
```

For a .Net assembly determine the fully qualified name first using the GACUTIL.exe /lr <assembly name> or other means and then:

```
'#Reference #Class1, Version=1.0.0.0, Culture=neutral, PublicKeyToken-  
n=3d01b60ac60ef95e, processorArchitecture=MSIL
```

Note that the status line at the bottom of the VBA Modules window will display an error if something is not correct. The Reference line entered will disappear after pressing enter at the end of the line.

The GACUTIL.exe /i can be used to install an assembly into the Global Assembly Cache. To verify, find the DLL's name in \Windows\Assembly. Changes to the DLL must be re-installed to the Global Assembly Cache to take effect. If RFgen is already running it may need to be restarted to see the change.

#Uses Special Comment

Syntax	'#Uses "module" [Only:[Win16 Win32 Win64]] ... -or- '\$Include: "module"
Group	Declaration
Description	The Uses comment indicates that the current <i>macro/module</i> uses public and friend symbols from the <i>module</i> . The Only option indicates that the module is only loaded for that Windows platform.
<hr/>	
Parameter	Description
<i>module</i>	Public and Friend symbols from this module are accessible. If the module name is a relative path then the path is relative to the macro/module containing the Uses comment. For example, if module "A:\B\CD.BAS" has this uses comment: '#Uses "E.BAS" then it uses "A:\B\CD\E.BAS".
<hr/>	
See Also	Class Module, Code Module, Object Module.
Example	<pre>'Macro A.WWB '#Language "WWB.NET" '#Uses "B.BAS" Sub Main Debug.Print BFun'("Hello") ""HELLO" End Sub 'Module B.BAS '#Language "WWB.NET" Public Function BFun'() BFun' = UCASE() End Function</pre>

AboutWinWrapBasic Instruction

Syntax	AboutWinWrapBasic [Timeout]
Group	Miscellaneous
Description	Show the WinWrap Basic about box.
<hr/>	
Parameter	Description
<i>Timeout</i>	This numeric value is the maximum number of seconds to show the about box. A value less than or equal to zero displays the about box until the user closes it. If this value is omitted then a three second timeout is used.
<hr/>	
Example	<pre>'#Language "WWB.NET" Sub Main AboutWinWrapBasic End Sub</pre>

Abs Function

Syntax `Abs(Num)`

Group Math

Description Return the absolute value.

Parameter	Description
<code>Num</code>	Return the absolute value of this numeric value.'

See Also [Sgn.](#)

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Abs(9) ' 9
    Debug.Print Abs(0) ' 0
    Debug.Print Abs(-9) ' 9
End Sub
```

AddHandler Instruction

Syntax `AddHandler expr.name, user delegate`

Group Flow Control

Description Add a *user delegate* to the *expr.name*'s list of handlers.

Parameter	Description
<i>expr</i>	This is an expression which returns an object reference.
<i>name</i>	This is an event name.
<i>user delegate</i>	This is an expression which returns a delegate.

See Also [RemoveHandler.](#)

Example

```
'#Reference #System.Windows.Forms, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089, processorArchitecture=MSIL
'#Language "WWB.NET"
Imports System.Windows.Forms
```

```
Sub Main
    Using t As New Timer
        AddHandler t.Tick, AddressOf OnTick
        t.Interval = 1000
        t.Enabled = True
        Wait 5
        RemoveHandler t.Tick, AddressOf OnTick
    End Using
End Sub

Private Sub OnTick
    Debug.Print Now
End Sub
```

AddressOf Operator

Syntax	AddressOf [expr.]name
Group	Operator
Description	Return a delegate for the <i>expr macro/module</i> 's Sub or Function matching <i>name</i> .

A delegate's Sub or Function is called using the delegate's Invoke method.

Note: The AddressOf operator returns an object which holds a weak reference to the macro/module. If no other references to the macro/module exist then the macro/module reference is deleted and using the result of the AddressOf operator causes a run-time error.

Parameter	Description
expr	This is a macro/module reference. If omitted then the current macro/module's reference (Me) is used.
name	Return a delegate for this Sub or Function.

See Also

Delegate.

Example

```
#Language "WWB.NET"
Delegate Function OpType(ByVal v1 As Object, ByVal v2 As Object) As Object

Sub Main
    Debug.Print DoOp(AddressOf Add,1,2) '3
    Debug.Print DoOp(AddressOf Subtract,1,2) '-1
End Sub

Function DoOp(ByVal op As OpType, _
             ByVal v1 As Object, ByVal v2 As Object) As Object
    DoOp = op.Invoke(v1,v2)
End Function

Function Add(ByVal v1 As Object, ByVal v2 As Object) As Object
    Add = v1+v2
End Function

Function Subtract(ByVal v1 As Object, ByVal v2 As Object) As Object
    Subtract = v1-v2
End Function
```

Any Data Type

Syntax	Declare ...(... v As Any ...)...
Group	Data Type
Description	Any variable expression (Declare only).

AppActivate Instruction

Syntax	AppActivate <i>Titl'</i> -or- AppActivate <i>TaskID</i>
Group	Miscellaneous
Description	<p>Form 1: Activate the application top-level window titled <i>Titl'</i>. If no window by that title exists then the first window with at title that starts with <i>Titl'</i> is activated.</p> <p>If no window matches then an error occurs.</p> <p>Form 2: Activate the application top-level window for task <i>TaskID</i>. If no window for that task exists then an error occurs.</p>
See Also	SendKeys, Shell().
Example	<pre>'#Language "WWB.NET" Sub Main ' make ProgMan the active application AppActivate "Program Manager" End Sub</pre>

Asc Function

Syntax	Asc()
Group	String
Description	Return the ASCII value.
	Note: A similar function, AscW, returns the Unicode value.
See Also	Ch'().
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print Asc("A") ' 65 End Sub</pre>

Assign Instruction

Syntax	Assign <i>lhs</i> , <i>rhs</i> [, <i>Depth</i>]
---------------	--

Group	Miscellaneous
Description	Assign the value of the <i>rhs</i> to the <i>lhs</i> .
Parameter	Description
<i>lhs</i>	This string represents the variable expression to be set.
<i>rhs</i>	Evaluate this string value and assign it to the <i>lhs</i> expression.
<i>Depth</i>	This integer value indicates how deep into the stack to locate the local variables. If Depth = 0 then use the current <i>procedure</i> . If this value is omitted then the depth is 0.
See Also	Eval.
Example	<pre>'#Language "WWB.NET" Sub Main Dim X As String Assign "X", """Hello""" Debug.Print X 'Hello A Debug.Print X 'Bye End Sub Sub A Dim X As String Assign "X", """Welcome""" Assign "X", """Bye""", 1 Debug.Print Eval("X") 'Welcome Debug.Print Eval("X",1) 'Bye End Sub</pre>

Assign Operators

Syntax	<i>var</i> [Op] <i>expr</i>
Group	Assignment
Description	Assign a value to <i>var</i> .
Op	Description
=	Assign the value of <i>expr</i> to <i>var</i> .
+=	The same as: <i>var</i> = <i>var</i> + (<i>expr</i>).
-=	The same as: <i>var</i> = <i>var</i> - (<i>expr</i>).
*=	The same as: <i>var</i> = <i>var</i> * (<i>expr</i>).
/=	The same as: <i>var</i> = <i>var</i> / (<i>expr</i>).
\=	The same as: <i>var</i> = <i>var</i> \ (<i>expr</i>).
^=	The same as: <i>var</i> = <i>var</i> ^ (<i>expr</i>).
<<=	The same as: <i>var</i> = <i>var</i> << (<i>expr</i>).
>>=	The same as: <i>var</i> = <i>var</i> >> (<i>expr</i>).
&=	The same as: <i>var</i> = <i>var</i> & (<i>expr</i>).
Example	<pre>'#Language "WWB.NET" Sub Main X = 1 X = X*2</pre>

```
Debug.Print X ' 2
End Sub
```

Atn Function

Syntax	<code>Atn(Num)</code>				
Group	Math				
Description	Return the arc tangent. This is the number of radians. There are 2*Pi radians in a full circle.				
	<hr/>				
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">Parameter</th><th style="text-align: left; padding: 2px;">Description</th></tr> </thead> <tbody> <tr> <td style="padding: 2px;"><code>Num</code></td><td style="padding: 2px;">Return the arc tangent of this numeric value.</td></tr> </tbody> </table> <hr/>	Parameter	Description	<code>Num</code>	Return the arc tangent of this numeric value.
Parameter	Description				
<code>Num</code>	Return the arc tangent of this numeric value.				
See Also	Cos, Sin, Tan.				
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print Atn(1)*4 ' 3.1415926535898 End Sub</pre>				

Attribute Definition/Statement

Syntax	<pre>Attribute attributename = value Attribute varname.attributename = value Attribute procname.attributename = value</pre>
Group	Declaration
Description	<p>All attribute definitions and statements are ignored except for:</p> <ul style="list-style-type: none"> • Form 1: Module level attribute <pre>Attribute VB_Name = "name" Attribute VB_GlobalNameSpace = bool Attribute VB_Creatable = bool Attribute VB_PredeclaredId = bool Attribute VB_Exposed = bool Attribute VB_HelpID = int Attribute VB_Description = "text"</pre> <p>VB_Name - Declares the name of the class module or object module. VB_GlobalNameSpace - Declares the class module as a global class. (ignore) VB_Creatable - Declares the module as creatable (True), non-creatable (False). (ignore) VB_PredeclaredId - Declares the module as a predeclared identifier (True). (ignore) VB_Exposed - Declares the module as public (True). (ignore) VB_HelpID - Declares the module's help context displayed by the object browser.</p>

VB_Description - Declares the module's help text displayed by the object browser.

- Form 2: Macro/Module level variable attribute

```
Public varname As type
Attribute varname.VB_VarUserMemId = 0
Attribute varname.VB_VarHelpID = int
Attribute varname.VB_VarDescription = "text"
```

VB_VarUserMemID - Declares **Public** varname as the default property for a **class module** or **object module**.

VB_VarHelpID - Declares the variable's help context displayed by the object browser.

VB_VarDescription - Declares the variable's help text displayed by the object browser.

- Form 3: User defined procedure attribute

```
[Sub | Function | Property [Get|Let|Set]] procname ...
Attribute procname.VB_UserMemId = 0
Attribute procname.VB_HelpID = int
Attribute procname.VB_Description = "text"
```

...

End [Sub | Function | Property]

VB_UserMemID - Declares **Property** procname as the default property for a **class module** or **object module**.

VB_HelpID - Declares the procedure's help context displayed by the object browser.

VB_Description - Declares the procedure's help text displayed by the object browser.

HelpFile

Each macro/module can define the HelpFile for the object browser:

```
'#HelpFile "helpfile"
```

where "helpfile" is a full path to the help file associated with the help text and help context.

Beep Instruction

Syntax Beep

Group Miscellaneous

Description Sound the bell.

Example '#Language "WWB.NET"

Sub Main

 Beep ' beep the bell

End Sub

Begin Dialog Definition

Syntax	<pre>Begin Dialog <i>name</i> [<i>X</i>, <i>Y</i>] <i>DX</i>, <i>DY</i>[, <i>Titl</i>]_ [<i>, dialogfunc</i>] User Dialog Item [User Dialog Item]... End Dialog</pre>																
Group	User Dialog																
Description	Define a <i>user dialog</i> type to be used later in a Dim As <i>name</i> statement.																
	<hr/> <table border="0"> <thead> <tr> <th style="text-align: left;">Parameter</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td><i>X</i></td> <td>This numeric value is the distance from the left edge of the screen to the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font. If this is omitted then the dialog will be centered.</td></tr> <tr> <td><i>Y</i></td> <td>This numeric value is the distance from the top edge of the screen to the top edge of the dialog box. It is measured in 1/12 ths of the average character width for the dialog's font. If this is omitted then the dialog will be centered.</td></tr> <tr> <td><i>DX</i></td> <td>This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.</td></tr> <tr> <td><i>DY</i></td> <td>This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.</td></tr> <tr> <td><i>Titl</i></td> <td>This string value is the title of the user dialog. If this is omitted then there is no title.</td></tr> <tr> <td><i>dialogfunc</i></td> <td>This is the function name that implements the DialogFunc for this <i>user dialog</i>. If this is omitted then the UserDialog doesn't have a dialogfunc.</td></tr> <tr> <td>User Dialog Item</td> <td>One of: CancelButton, CheckBox, ComboBox, DropListBox, GroupBox, ListBox, MultiListBox, OKButton, OptionButton, OptionGroup, PushButton, Text, TextBox.</td></tr> </tbody> </table> <hr/>	Parameter	Description	<i>X</i>	This numeric value is the distance from the left edge of the screen to the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font. If this is omitted then the dialog will be centered.	<i>Y</i>	This numeric value is the distance from the top edge of the screen to the top edge of the dialog box. It is measured in 1/12 ths of the average character width for the dialog's font. If this is omitted then the dialog will be centered.	<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.	<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.	<i>Titl</i>	This string value is the title of the user dialog. If this is omitted then there is no title.	<i>dialogfunc</i>	This is the function name that implements the DialogFunc for this <i>user dialog</i> . If this is omitted then the UserDialog doesn't have a dialogfunc.	User Dialog Item	One of: CancelButton , CheckBox , ComboBox , DropListBox , GroupBox , ListBox , MultiListBox , OKButton , OptionButton , OptionGroup , PushButton , Text , TextBox .
Parameter	Description																
<i>X</i>	This numeric value is the distance from the left edge of the screen to the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font. If this is omitted then the dialog will be centered.																
<i>Y</i>	This numeric value is the distance from the top edge of the screen to the top edge of the dialog box. It is measured in 1/12 ths of the average character width for the dialog's font. If this is omitted then the dialog will be centered.																
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.																
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.																
<i>Titl</i>	This string value is the title of the user dialog. If this is omitted then there is no title.																
<i>dialogfunc</i>	This is the function name that implements the DialogFunc for this <i>user dialog</i> . If this is omitted then the UserDialog doesn't have a dialogfunc.																
User Dialog Item	One of: CancelButton , CheckBox , ComboBox , DropListBox , GroupBox , ListBox , MultiListBox , OKButton , OptionButton , OptionGroup , PushButton , Text , TextBox .																
Example	<pre>'#Language "WWB.NET" Sub Main Begin Dialog UserDialog 200,120 Text 10,10,180,15,"Please push the OK button" OKButton 80,90,40,20 End Dialog Dim dlg As UserDialog Dialog dlg ' show dialog (wait for ok) End Sub</pre>																

Boolean Data Type

Syntax	Dim <i>v</i> As Boolean
Group	Data Type
Description	A True or False value.

Byte Data Type

Syntax	Dim <i>v</i> As Byte
---------------	------------------------------------

Group	Data Type
Description	An 8 bit unsigned integer value.

Call Instruction

Syntax	Call <i>name</i> [(<i>arglist</i>)] -or- <i>name</i> [<i>arglist</i>]
Group	Flow Control
Description	Evaluate the <i>arglist</i> and call subroutine (or function) <i>name</i> with those values. Sub (or function) <i>name</i> must be previously defined by either a Sub , Function or Property definition. If <i>name</i> is a function then the result is discarded. If Call is omitted and <i>name</i> is a subroutine then the <i>arglist</i> must not be enclosed in parens.
See Also	Declare , Sub .
Example	<pre>'#Language "WWB.NET" Sub Show(Titl',Value) Debug.Print Titl'; "="; Value End Sub Sub Main Call Show("2000/9",2000/9) ' 222.222222222 Show "1<2",1<2 True End Sub</pre>

CallByName Instruction

Syntax	CallByName(<i>Obj</i> , <i>ProcName</i> , <i>CallType</i> ,[<i>expr</i> [, ...]])
Group	Flow Control
Description	Call an <i>Obj</i> 's method/property, <i>ProcName</i> , by name. Pass the <i>exprs</i> to the method/property.

Parameter	Description
<i>Obj</i>	Call the method/property for this object reference.
<i>ProcName</i>	This string value is the name of the method/property to be called.
<i>CallType</i>	Type of method/property call. See table below.
<i>expr</i>	These expressions are passed to the <i>obj</i> 's method/property.

CallType	Value	Effect
vbMethod	1	Call or evaluate the method.
vbGet	2	Evaluate the property's value.
vbLet	4	Assign the property's value.
vbSet	8	Set the property's reference.

Example

```
'#Language "WWB.NET"
Sub Main
    On Error Resume Next
    CallByName Err, "Raise", vbMethod, 1
    Debug.Print CallByName(Err, "Number", vbGet) ' 1
End Sub
```

CallersLine Function

Syntax

CallersLine[(*Depth*)]

Group

Miscellaneous

Description

Return the caller's line as a text string.

The text format is: "[macroname|subname#linenum] linetext".

Parameter	Description
<i>Depth</i>	This integer value indicates how deep into the stack to get the caller's line. If Depth = -1 then return the current line. If Depth = 0 then return the calling subroutine's current line, etc.. If Depth is greater than or equal to the call stack depth then a null string is returned. If this value is omitted then the depth is 0.

Example

```
'#Language "WWB.NET"
Sub Main
    A
End Sub
Sub A
    Debug.Print CallersLine "[(untitled 1)|Main# 2] A"
End Sub
```

CancelButton Dialog Item Definition

Syntax

CancelButton *X*, *Y*, *DX*, *DY*[, *Field*]

Group

User Dialog

Description

Define a cancel button item. Pressing the Cancel button from a **Dialog** instruction causes a run-time error. (**Dialog()** function call returns 0.)

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Field</i>	This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i> . If this is omitted then the field name is "Cancel".

See Also

Begin Dialog.

Example

```
'#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120
        Text 10,10,180,30,"Please push the Cancel button"
        OKButton 40,90,40,20
        CancelButton 110,90,60,20
    End Dialog
    Dim dlg As UserDialog
    Dialog dlg 'show dialog (wait for cancel)
    Debug.Print "Cancel was not pressed"
End Sub
```

CBool Function

Syntax

CBool(*expr*)

Group

Conversion

Description

Convert to a **Boolean** value. Zero converts to **False**, while all other values convert to **True**.

Parameter	Description
<i>expr</i>	Convert a number or string value to a boolean value.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print CBool(-1) 'True
    Debug.Print CBool(0) 'False
    Debug.Print CBool(1) 'True
End Sub
```

CByte Function

Syntax

CByte(*expr*)

Group

Conversion

Description

Convert to an 8 bit unsigned integer **Byte** value.

Parameter	Description
<i>expr</i>	Convert a number or string value to an unsigned byte value.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print CByte(1.6) '2
End Sub
```

CChar Function

Syntax

CChar(*expr*)

Group	Conversion
Description	Convert to a Char value.
<hr/>	
Parameter	Description
<hr/>	
expr	Convert a number or string value to an single character value. This is the first character of the string conversion.
<hr/>	
Example	'#Language "WWB.NET" Sub Main Debug.Print CChar(33) "!" End Sub

CDate Function

Syntax	CDate(expr)
Group	Conversion
Description	Convert to a Date value.
<hr/>	
Parameter	Description
<hr/>	
expr	Convert a number or string value to a date value.
<hr/>	
Example	'#Language "WWB.NET" Sub Main Debug.Print CDate(2) ' 1/1/00 End Sub

CDbl Function

Syntax	CDbl(expr)
Group	Conversion
Description	Convert to a Double precision real.
<hr/>	
Parameter	Description
<hr/>	
expr	Convert a number or string value to a double precision real.
<hr/>	
Example	'#Language "WWB.NET" Sub Main Debug.Print CDbl("1E6") ' 1000000 End Sub

CDec Function

Syntax	CDec(expr)
Group	Conversion
Description	Convert to a Decimal (96 bit scaled real).

Parameter	Description
expr	Convert a number or string value to a 96 bit scaled real.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print CDec("1E16") + 0.1 ' 1000000000000000.1
End Sub
```

Char Data Type

Syntax Dim v As Char**Group** Data Type**Description** A one character value.

ChDir Instruction

Syntax ChDir *Di'***Group** File**Description** Change the current directory to *Di'*.**Pocket PC** Not supported.

Parameter	Description
<i>Di'</i>	This string value is the path and name of the directory.

See Also [ChDrive, CurDi'\(\)](#).**Example**

```
'#Language "WWB.NET"
Sub Main
    ChDir "C:\"
    Debug.Print CurDi() "C:\""
End Sub
```

ChDrive Instruction

Syntax ChDrive *Driv'***Group** File**Description** Change the current drive to *Driv'*.**Pocket PC** Not supported.

Parameter	Description
<i>Driv'</i>	This string value is the drive letter.

See Also [ChDir, CurDi'\(\)](#).

Example

```
'#Language "WWB.NET"
Sub Main
    ChDrive "B"
    Debug.Print CurDi() ""B:\"
End Sub
```

CheckBox Dialog Item Definition

Syntax CheckBox *X, Y, DX, DY, Titl', .Field[, Options]*

Group User Dialog

Description Define a checkbox item.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Field</i>	The value of the check box is accessed via this field. Unchecked is 0, checked is 1 and grayed is 2.
<i>Options</i>	This numeric value controls the type of check box. Choose one value from following table. (If this numeric value omitted then zero is used.)

Option	Description
0	Check box is either check or unchecked.
1	Check box is either check, unchecked or grayed, and it switches between checked and unchecked when clicked.
2	Check box is either check, unchecked or grayed, and it cycles through all three states as the button is clicked.

See Also

Begin Dialog.

Example

```
'#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120
        Text 10,10,180,15,"Please push the OK button"
        CheckBox 10,25,180,15,"&Check box",Check
        OKButton 80,90,40,20
    End Dialog
    Dim dlg As UserDialog
    dlg.Check = 1
    Dialog dlg ' show dialog (wait for ok)
    Debug.Print dlg.Check
End Sub
```

Choose Function

Syntax	Choose(<i>Index</i> , <i>expr</i> [, ...])
Group	Flow Control
Description	Return the value of the <i>expr</i> indicated by <i>Index</i> .
<hr/>	
Parameter	Description
<i>Index</i>	The numeric value indicates which <i>expr</i> to return. If this value is less than one or greater than the number of <i>exprs</i> then System.DBNull.Value is returned.
<i>expr</i>	All expressions are evaluated.
<hr/>	
See Also	If , Select Case , IIf() .
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print Choose(2,"Hi","there") "there" End Sub</pre>

Ch' Function

Syntax	Ch'(<i>Num</i>)
Group	String
Description	Return a one char string for the ASCII value.
<p>Note: A similar function, ChrW, returns a single char Unicode string.</p>	
Parameter	Description
<i>Num</i>	Return one char string for this ASCII numeric value.
<hr/>	
See Also	Asc().
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print Ch'(48) "0" End Sub</pre>

CInt Function

Syntax	CInt(<i>expr</i>)
Group	Conversion
Description	Convert to an Integer . If <i>expr</i> is too big (or too small) to fit then an overflow error occurs.
<hr/>	
Parameter	Description
<i>expr</i>	Convert a number or string value to an Integer .
<hr/>	

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print CInt(1.6) ' 2
End Sub
```

Class Module

Group	Declaration
Description	<p>A class <i>module</i> implements an object.</p> <p>Note: The Class statement is not supported. Use a class module instead.</p> <ul style="list-style-type: none"> • Has a set of Public procedures accessible from other <i>macros</i> and <i>modules</i>. • These public symbols are accessed via an object variable. • Has an optional set of Events that can be raised. • Public Consts, Structures, arrays, fixed length strings are not allowed. • Has an optional Private Sub Class_Initialize which is called when an instance is created. • Has an optional Private Sub Class_Terminate which is called when an instance is destroyed. • A class module is similar to a object module except that no instance is automatically created. • To create an instance use: <code>Dim Obj As classname Obj = New classname</code>
See Also	Code Module, Object Module, Uses, Class_Initialize, Class_Terminate.
Example	<pre>'A.WWB '#Language "WWB.NET" '#Uses "File.CLS" Sub Main Dim File As New File File.Attach "C:\AUTOEXEC.BAT" Debug.Print File.ReadLine End Sub 'File.CLS 'File New Module Class Module >Edit Properties Name=File '#Language "WWB.NET" Option Explicit Dim FN As Integer Public Sub Attach(FileName As String) FN = FreeFile FileOpen FN, FileName, OpenMode.Input End Sub Public Sub Detach() If FN <> 0 Then FileClose'FN</pre>

```

FN = 0
End Sub
Public Function ReadLine() As String
    ReadLine = LineInput(FN)
End Function

Private Sub Class_Initialize()
    Debug.Print "Class_Initialize"
End Sub

Private Sub Class_Terminate()
    Debug.Print "Class_Terminate"
    Detach
End Sub

```

Class_Initialize Sub

Syntax	Private Sub Class_Initialize() ...
Group	Declaration
Description	Class module initialization subroutine. Each time a new instance is created for a class module the Class_Initialize sub is called. If Class_Initialize is not defined then no special initialization occurs.
See Also	Code Module , Class_Terminate .

Class_Terminate Sub

Syntax	Private Sub Class_Terminate() ...
Group	Declaration
Description	Class module termination subroutine. Each time an instance is destroyed for a class module the Class_Terminate sub is called. If Class_Terminate is not defined then no special termination occurs.
See Also	Code Module , Class_Initialize .

Clipboard Instruction/Function

Syntax	Clipboard Tex' -or- Clipboar'[()]
Group	Miscellaneous

Description Form 1: Set the clipboard to *Tex'*. This is like the Edit|Copy menu command.

Form 2: Return the text in the clipboard.

Parameter	Description
<i>Tex'</i>	Put this string value into the clipboard.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Clipboar()
    Clipboard "Hello"
    Debug.Print Clipboar() "Hello"
End Sub
```

CLng Function

Syntax CLng(*expr*)

Group Conversion

Description Convert to a **Long**. If *expr* is too big (or too small) to fit then an overflow error occurs.

Parameter	Description
<i>expr</i>	Convert a number or string value to a Long .

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print CLng(1.6) ' 2
End Sub
```

CObj Function

Syntax CObj(*expr*)

Group Conversion

Description Convert to an **Object**. The object contains the value.

Parameter	Description
<i>expr</i>	Convert to an object. If the parameter is already an object, return it.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print CObj(Sqr(2)) ' 1.4142135381699
End Sub
```

Code Module

Group Declaration

Description	A Code <i>module</i> implements a code library. Note: The Module statement is not supported. Use a code module instead.
	<ul style="list-style-type: none"> • Has a set of Public <i>procedures</i> accessible from other <i>macros</i> and <i>modules</i>. • These public symbols are accessed directly. • May be initialized by a Private Sub Main.

See Also	Class Module, Object Module, Uses, Main.
-----------------	---

Example	<pre>'A.WWB '#Language "WWB.NET" '#Uses "Module1.BAS" Sub Main Debug.Print Value "Hello" End Sub 'Module1.BAS 'File New Module Code Module >Edit Properties Name=Module1 '#Language "WWB.NET" Option Explicit Private mValue As String Property Get Value() As String Value = mValue End Property 'this sub is called when the module is first loaded Private Sub Main mValue = "Hello" End Sub</pre>
----------------	---

ComboBox Dialog Item Definition

Syntax	ComboBox <i>X, Y, DX, DY, StrArra'(), .Fiel[], Options</i>
Group	User Dialog
Description	Define a combobox item. Combo boxes combine the functionality of an edit box and a list box.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>StrArra'()</i>	This one-dimensional array of strings establishes the list of choices. All the non-null elements of the array are used.
<i>Fiel'</i>	The value of the combo box is accessed via this field. This is the text in the edit box.
<i>Options</i>	This numeric value controls the type of combo box. Choose one value from following table. (If this numeric value omitted then zero is used.)

Option	Description
0	List is not sorted.
2	List is sorted.

See Also**Begin Dialog.****Example**

```
'#Language "WWB.NET"
Sub Main
    Dim combo'(3)
    combo'(0) = "Combo 0"
    combo'(1) = "Combo 1"
    combo'(2) = "Combo 2"
    combo'(3) = "Combo 3"
    Begin Dialog UserDialog 200,120
        Text 10,10,180,15,"Please push the OK button"
        ComboBox 10,25,180,60,combo'(),.comb'
        OKButton 80,90,40,20
    End Dialog
    Dim dlg As UserDialog
    dlg.comb' = "none"
    Dialog dlg ' show dialog (wait for ok)
    Debug.Print dlg.comb'
End Sub
```

Comman' Function

Syntax

Comman'

Group

Miscellaneous

DescriptionContains the value of the **MacroRun** parameters.**See Also****MacroRun.****Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print "Command line parameter is: """;
    Debug.Print Comman';
    Debug.Print """"
End Sub
```

Const Definition

Syntax

```
[ | Private | Public ]_
Const name[type] [As Type] = expr[, ...]
```

Group

Declaration

Description

Define *name* as the value of *expr*. The *expr* may be refer other constants or built-in functions. If the type of the constants is not specified, the type of *expr* is used. Constants defined outside a **Sub**, **Function** or **Property** block are available in the entire *macro/module*.

Private is assumed if neither **Private** or **Public** is specified.

Note: Const statement in a **Sub**, **Function** or **Property** block may not use **Private** or **Public**.

Example

```
'#Language "WWB.NET"
Sub Main
    Const Pi = 4*Atn(1), e = Exp(1)
    Debug.Print Pi ' 3.14159265358979
    Debug.Print e ' 2.71828182845905
End Sub
```

Cos Function

Syntax

`Cos(Num)`

Group

Math

Description

Return the cosine.

Parameter	Description
<code>Num</code>	Return the cosine of this numeric value. This is the number of radians. There are 2*Pi radians in a full circle.

See Also

Atn, Sin, Tan.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Cos(1) ' 0.54030230586814
End Sub
```

CreateObject Function

Syntax

`CreateObject(Clas')`

Group

Object

Description

Create a new object of type `Clas'`.

Parameter	Description
<code>Clas'</code>	This string value is the application's registered class name. If this application is not currently active it will be started.

See Also

Objects.

Example

```
'#Language "WWB.NET"
Sub Main
    Dim App As Object
    'App = CreateObject("WinWrap.CppDemoApplication")
    App.Move 20,30 ' move icon to 20,30
    'App = Nothing
```

```
App.Quit      ' run-time error (no object)
End Sub
```

CSByte Function

Syntax CSByte(*expr*)

Group Conversion

Description Convert to an 8 bit signed integer **SByte** value.

Parameter	Description
<i>expr</i>	Convert a number or string value to a signed byte value.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print CSByte(1.6) ' 2
End Sub
```

CShort Function

Syntax CShort(*expr*)

Group Conversion

Description Convert to a 16 bit signed integer **Short** value. If *expr* is too big (or too small) to fit then an overflow error occurs.

Parameter	Description
<i>expr</i>	Convert a number or string value to a 16 bit signed integer.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print CShort(1.6) ' 2
End Sub
```

CSng Function

Syntax CSng(*expr*)

Group Conversion

Description Convert to a **Single** precision real. If *expr* is too big (or too small) to fit then an overflow error occurs.

Parameter	Description
<i>expr</i>	Convert a number or string value to a single precision real.

Example

```
'#Language "WWB.NET"
Sub Main
```

```
Debug.Print CSng(Sqr(2)) ' 1.4142135381699
End Sub
```

CStr Function

Syntax CStr(*expr*)

Group Conversion

Description Convert to a **String**.

Parameter	Description
<i>expr</i>	Convert a number or string value to a string value using the current locale (GetLocale).

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print CStr(Sqr(2)) "1.4142135623731" - US English locale
End Sub
```

CurDi' Function

Syntax CurDi'([*Driv*])

Group File

Description Return the current directory for *Driv'*.

Pocket PC Not supported.

Parameter	Description
<i>Driv'</i>	This string value is the drive letter. If this is omitted or null then return the current directory for the current drive.

See Also ChDir, ChDrive.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print CurDi'()
End Sub
```

CType Function

Syntax CType(*expr*, *objtype*)

Group Conversion

Description Convert an *expr* to the specified *objtype* type. Conversion include inheritance, implementation and value conversion. If the *expr* can't be converted, a "type mismatch error" will occur.

Parameter	Description
-----------	-------------

<i>expr</i>	Convert the value of this expression.
<i>objtype</i>	Convert to this type.

See Also**DirectCast, TryCast.****Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print CType(1.1, Integer) ' 1
    Dim V As Object
    V = Err
    Debug.Print TypeName(CType(V, ErrObject)) ' ErrObject
End Sub
```

CUInt Function

SyntaxCUInt(*expr*)**Group**

Conversion

Description

Convert to an **UInteger**. If *expr* is too big (or too small) to fit then an overflow error occurs.

Parameter	Description
<i>expr</i>	Convert a number or string value to an UInteger .

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print CUInt(1.6) ' 2
End Sub
```

CULng Function

SyntaxCULng(*expr*)**Group**

Conversion

Description

Convert to a **ULong**. If *expr* is too big (or too small) to fit then an overflow error occurs.

Parameter	Description
<i>expr</i>	Convert a number or string value to a ULong .

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print CULng(1.6) ' 2
End Sub
```

CUShort Function

SyntaxCUShort(*expr*)

Group	Conversion
Description	Convert to a 16 bit unsigned integer UShort value. If <i>expr</i> is too big (or too small) to fit then an overflow error occurs.
<hr/>	
Parameter	Description
<i>expr</i>	Convert a number or string value to a 16 bit unsigned integer.
<hr/>	
Example	'#Language "WWB.NET" Sub Main Debug.Print CUShort(1.6) ' 2 End Sub

Date Data Type

Syntax	<code>Dim v As Date</code>
Group	Data Type
Description	A 64 bit real value. The whole part represents the date, while the fractional part is the time of day. (December 30, 1899 = 0.) Use #date# as a literal date value in an expression.

DateAdd Function

Syntax	<code>DateAdd(interval, number, dateexpr)</code>
Group	Time/Date
Description	Return a Date value a number of intervals from another date.
<hr/>	
Parameter	Description
<i>interval</i>	This string value indicates which kind of interval to add.
<i>number</i>	Add this many intervals. Use a negative value to get an earlier date.
<i>dateexpr</i>	Calculate the new date relative to this date value.'
<hr/>	
Interval	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week
h	Hour
n	Minute
s	Second
<hr/>	

See Also [DateDiff](#), [DatePart](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print DateAdd("yyyy",1,#1/1/2000#) '1/1/2001
End Sub
```

DateDiff Function

Syntax

DateDiff(*interval*, *dateexpr1*, *dateexpr2*)

Group

Time/Date

Description

Return the number of intervals between two dates.

Parameter	Description
<i>interval</i>	This string value indicates which kind of interval to subtract.
<i>dateexpr1</i>	Calculate the from this date value to <i>dateexpr2</i> . '
<i>dateexpr2</i>	Calculate the from <i>dateexpr1</i> to this date value. '

Interval	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week
h	Hour
n	Minute
s	Second

See Also

DateAdd, **DatePart**.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print DateDiff("yyyy",#1/1/1990#,#1/1/2000#) ' 10
End Sub
```

DatePart Function

Syntax

DatePart(*interval*, *dateexpr*)

Group

Time/Date

Description

Return the number from the date corresponding to the interval.

Parameter	Description
<i>interval</i>	This string value indicates which kind of interval to extract.
<i>dateexpr</i>	Get the interval from this date value. '

Interval	Description (return value range)
yyyy	Year (100-9999)

q	Quarter (1-4)
m	Month (1-12)
y	Day of year (1-366)
d	Day (1-31)
w	Weekday (1-7)
ww	Week (1-53)
h	Hour (0-23)
n	Minute (0-59)
s	Second (0-59)

See Also**DateAdd**, **DateDiff**.**Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print DatePart("yyyy",#1/1/2000#) ' 2000
End Sub
```

DateSerial Function

Syntax DateSerial(*Year*, *Month*, *Day*)**Group** Time/Date**Description** Return a **Date** value.

Parameter	Description
<i>Year</i>	This numeric value is the year (0 to 9999). (0 to 99 are interpreted by the operating system.)
<i>Month</i>	This numeric value is the month (1 to 12).
<i>Day</i>	This numeric value is the day (1 to 31).

See Also**DateValue**, **TimeSerial**, **TimeValue**.**Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print DateSerial(2000,7,4) ' 7/4/2000
End Sub
```

DateValue Function

Syntax DateValue(*Dat'*)**Group** Time/Date**Description** Return the day part of the date encoded as a string.

Parameter	Description
<i>Dat'</i>	Convert this string value to the day part of date it represents.

See Also**DateSerial**, **TimeSerial**, **TimeValue**.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print DateValue("1/1/2000 12:00:01 AM")
    '1/1/2000
End Sub
```

Day Function

Syntax Day(*dateexpr*)

Group Time/Date

Description Return the day of the month (1 to 31).

Parameter	Description
<i>dateexpr</i>	Return the day of the month for this date value.'

See Also **Date(), Month(), Weekday(), Year().**

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Day(#1/1/1900#) ' 1
    Debug.Print Day(#1/2/1900#) ' 2
End Sub
```

DDEExecute Instruction

Syntax DDEExecute *ChanNum*, *Command*[, *Timeout*]

Group DDE

Description Send the DDE Execute *Command*' string via DDE *ChanNum*.

Pocket PC Not supported.

Parameter	Description
<i>ChanNum</i>	This is the channel number returned by the DDEInitiate function. Up to 10 channels may be used at one time.
<i>Command</i> '	Send this command value to the server application. The interpretation of this value is defined by the server application.
<i>Timeout</i>	The command will generate an error if the number of seconds specified by the timeout is exceeded before the command has completed. The default is five seconds.

Example

```
'#Language "WWB.NET"
Sub Main
    ChanNum = DDEInitiate("PROGMAN","PROGMAN")
    DDEExecute ChanNum, "[CreateGroup(XXX)]"
    DDETernate ChanNum
End Sub
```

DDEInitiate Function

Syntax	<code>DDEInitiate(<i>Ap'</i>, <i>Topi'</i>)</code>
Group	DDE
Description	Initiate a DDE conversation with <i>Ap'</i> using <i>Topi'</i> . If the conversation is successfully started then the return value is a channel number that can be used with other DDE instructions and functions.
Pocket PC	Not supported.
<hr/>	
Parameter	Description
<i>Ap'</i>	Locate this server application.
<i>Topi'</i>	This is the server application's topic. The interpretation of this value is defined by the server application.
<hr/>	
Example	<pre>'#Language "WWB.NET" Sub Main ChanNum = DDEInitiate("PROGMAN","PROGMAN") DDEExecute ChanNum,"[CreateGroup(XXX)]" DDETerninate ChanNum End Sub</pre>

DDEPoke Instruction

Syntax	<code>DDEPoke <i>ChanNum</i>, <i>Ite'</i>, <i>Dat'</i>[, <i>Timeout</i>] </code>
Group	DDE
Description	Poke <i>Dat'</i> to the <i>Ite'</i> via DDE <i>ChanNum</i> .
Pocket PC	Not supported.
<hr/>	
Parameter	Description
<i>ChanNum</i>	This is the channel number returned by the DDEInitiate function. Up to 10 channels may be used at one time.
<i>Ite'</i>	This is the server application's item. The interpretation of this value is defined by the server application.
<i>Dat'</i>	Send this data value to the server application. The interpretation of this value is defined by the server application.
<i>Timeout</i>	The command will generate an error if the number of seconds specified by the timeout is exceeded before the command has completed. The default is five seconds.
<hr/>	
Example	<pre>'#Language "WWB.NET" Sub Main ChanNum = DDEInitiate("PROGMAN","PROGMAN") DDEPoke ChanNum,"Group","XXX" DDETerninate ChanNum End Sub</pre>

DDEReques' Function

Syntax	<code>DDEReques'(ChanNum, Ite[, Timeout])</code>
Group	DDE
Description	Request information for <i>Ite</i> '. If the request is not satisfied then the return value will be a null string.
Pocket PC	Not supported.

Parameter	Description
<i>ChanNum</i>	This is the channel number returned by the DDEInitiate function. Up to 10 channels may be used at one time.
<i>Ite'</i>	This is the server application's item. The interpretation of this value is defined by the server application.
<i>Timeout</i>	The command will generate an error if the number of seconds specified by the timeout is exceeded before the command has completed. The default is five seconds.

Example	<pre>'#Language "WWB.NET" Sub Main ChanNum = DDEInitiate("PROGMAN","PROGMAN") Debug.Print DDEReques'(ChanNum,"Groups") DDETerninate ChanNum End Sub</pre>
----------------	---

DDETerninate Instruction

Syntax	<code>DDETerninate ChanNum</code>
Group	DDE
Description	Terminate DDE <i>ChanNum</i> .
Pocket PC	Not supported.

Parameter	Description
<i>ChanNum</i>	This is the channel number returned by the DDEInitiate function. Up to 10 channels may be used at one time.

Example	<pre>'#Language "WWB.NET" Sub Main ChanNum = DDEInitiate("PROGMAN","PROGMAN") DDEExecute ChanNum, "[CreateGroup(XXX)]" DDETerninate ChanNum End Sub</pre>
----------------	---

DDETerninateAll Instruction

Syntax	<code>DDETerninateAll</code>
Group	DDE

Description	Terminate all open DDE channels.
Pocket PC	Not supported.
Example	<pre>#Language "WWB.NET" Sub Main ChanNum = DDEInitiate("PROGMAN","PROGMAN") DDEExecute ChanNum,["CreateGroup(XXX)"] DDETernateAll End Sub</pre>

Debug Object

Syntax	Debug.Clear -or- Debug.Print [<i>expr</i> ; ...][;]
Group	Miscellaneous
Description	<p>Form 1: Clear the output window.</p> <p>Form 2: Print the <i>expr</i>(s) to the output window. Use ; to separate expressions. A <i>num</i> is automatically converted to a string before printing (just like St'()). If the instruction does not end with a ; then a newline is printed at the end.</p>
Example	<pre>#Language "WWB.NET" Sub Main X = 4 Debug.Print "X/2="; X/2 ' 2 Debug.Print "Start..."; ' don't print a newline Debug.Print "Finish" ' print a newline End Sub</pre>

Decimal Data Type

Syntax	Dim <i>v</i> As Decimal
Group	Data Type
Description	<p>A 96 bit scaled real value. A decimal number is of the form: s*m*10^-p where</p> <ul style="list-style-type: none"> • s - sign (+1 or -1) • m - mantissa, unsigned binary value of 96 bits (0 to 79,228,162,514,264,337,593,543,950,335) • p - scaling power (0 to +28)

Declare Definition

Syntax

```
[| Private | Public ]_
Declare Sub name Lib "dll name"_
[Alias "module name"] [(param[, ...])]

-or-

[| Private | Public ]_
Declare Function name[type] Lib "dll name"_
[Alias "module name"] [(param[, ...])] [As type()]
```

Group

Declaration

Description

Interface to a DLL defined subroutine or function. The values of the calling *arglist* are assigned to the *params*.

WARNING! Be very careful when declaring DLL subroutines or functions. If you make a mistake and declare the parameters or result incorrectly then Windows might halt. Save any open documents before testing new DLL declarations.

Err.LastDLLError returns the error code for that last DLL call.

Access

If no access is specified then **Public** is assumed.

Pocket PC

Not supported.

Parameter	Description
<i>name</i>	This is the name of the subroutine or function being defined. If Alias "module name" is omitted then this is the module name, too.
" <i>dll name</i> "	This is the DLL file where the module's code is.
" <i>module name</i> "	This is the name of the module in the DLL file. If this is #number then it is the ordinal number of the module. If it is omitted then <i>name</i> is the module name. The DLL is searched for the specified module name. If this module exists, it is used. All As String parameters are converted from Unicode to ASCII prior to calling the DLL and from ASCII to Unicode afterwards. (Use "Unicode: <i>module name</i> " to prevent ASCII to Unicode conversion.) If the module does not exist, one or two other module names are tried: 1) For Windows NT only: The module name with a "W" appended is tried. All As String parameters are passed as Unicode to calling the DLL. 2) For Windows NT and Windows 95: The module name with an "A" appended is tried. All As String parameters are converted from Unicode to ASCII prior to calling the DLL and from ASCII to Unicode afterwards. If none of these module names is found a run-time error occurs.
<i>params</i>	A list of zero or more <i>params</i> that are used by the DLL subroutine or function. (Note: A ByVal string's value may be modified by the DLL.)

See Also

Function, Sub, Call.

Example

```
#Language "WWB.NET"
Declare Function GetActiveWindow& Lib "user32" ()
Declare Function GetWindowTextLengthA& Lib "user32" _
(ByVal hwnd&)
Declare Sub GetWindowTextA Lib "user32" _
(ByVal hwnd&, ByVal lps$, ByVal cbMax&)
```

```

Function ActiveWindowTitl'()
  ActiveWindow = GetActiveWindow()
  TitleLen = GetWindowTextLengthA(ActiveWindow)
  Titl' = Spac'(TitleLen)
  GetWindowTextA ActiveWindow,Titl',TitleLen+1
  ActiveWindowTitl' = Titl'
End Function

Sub Main
  Debug.Print ActiveWindowTitl'()
End Sub

```

Decode64 Function

Syntax

Decode6'(*Data*)

-or-

Decode64B(*Data*)

Group

Miscellaneous

Description

Return a string using the base 64 decoding algorithm.

Decode64B returns a **Byte** array.

Parameter	Description
<i>Data</i>	Return this string's base 64 decoding.

See Also

Encode64.

Example

```

'#Language "WWB.NET"
Sub Main
  Debug.Print Decode64("SGVsbG8gV29ybGQhIQ==") "Hello World!!"
End Sub

```

Decrypt64 Function

Syntax

Decrypt6'(*Data* [,*Password*])

-or-

Decrypt64B(*Data* [,*Password*])

Group

Miscellaneous

Description

Return a string using the RC4 stream decryption algorithm.

Decrypt64B returns a **Byte** array.

Parameter	Description
<i>Data</i>	Return this string's decryption. (The string is first decoded using base 64 decoding.)
<i>Password</i>	Decrypt using this password.

See Also

Decode64, Encode64, Encrypt64.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Decrypt64("Y4GFrF+k1YUHwjEzsg==", "abc") "Hello World!!"
End Sub
```

Delegate Definition

Syntax	[Private Public]_ Delegate Sub <i>name</i> [([<i>param</i> [, ...]])] -or- [Private Public]_ Delegate Function <i>name</i> [<i>type</i>] _ [([<i>param</i> [, ...]])] [<i>As type</i> (())]]						
Group	Declaration						
Description	Define a new <i>user delegate</i> (subroutine or function pointer type).						
	A delegate's Sub or Function is called using the delegate's Invoke method.						
Access	If no access is specified then Public is assumed.						
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>name</i></td> <td>This is the name of the delegate being defined.</td> </tr> <tr> <td><i>params</i></td> <td>A list of zero or more <i>params</i> that are used by the delegate's subroutine or function.</td> </tr> </tbody> </table>	Parameter	Description	<i>name</i>	This is the name of the delegate being defined.	<i>params</i>	A list of zero or more <i>params</i> that are used by the delegate's subroutine or function.
Parameter	Description						
<i>name</i>	This is the name of the delegate being defined.						
<i>params</i>	A list of zero or more <i>params</i> that are used by the delegate's subroutine or function.						
See Also	AddressOf.						
Example	<pre>'#Language "WWB.NET" Delegate Function OpType(ByVal v1 As Object, ByVal v2 As Object) As Object Sub Main Debug.Print DoOp(AddressOf Add,1,2) ' 3 Debug.Print DoOp(AddressOf Subtract,1,2) '-1 End Sub Function DoOp(ByVal op As OpType, _ ByVal v1 As Object, ByVal v2 As Object) As Object DoOp = op.Invoke(v1,v2) End Function Function Add(ByVal v1 As Object, ByVal v2 As Object) As Object Add = v1+v2 End Function Function Subtract(ByVal v1 As Object, ByVal v2 As Object) As Object Subtract = v1-v2 End Function</pre>						

DeleteSetting Instruction

Syntax

DeleteSetting *AppNam'*, *Sectio*[, *Ke*]

Group	Settings
Description	Delete the settings for <i>Key</i> in <i>Section</i> in project <i>AppName</i> . Win16 and Win32s store settings in a .ini file named <i>AppName</i> . Win32/Win64 store settings in the registration database under "HKEY_CURRENT_USER\ Software\ VB and VBA Program Settings\ <i>AppName</i> \ <i>Section</i> \ <i>Key</i> ". If <i>AppName</i> starts with "..\" then "VB and VBA Program Settings\" is omitted.
<hr/>	
Parameter	Description
<i>AppNam'</i>	This string value is the name of the project which has this <i>Section</i> and <i>Key</i> .
<i>Sectio'</i>	This string value is the name of the section of the project settings.
<i>Ke'</i>	This string value is the name of the key in the section of the project settings. If this is omitted then delete the entire section.

Example

```
'#Language "WWB.NET"
Sub Main
    SaveSetting "MyApp","Font","Size",10
    DeleteSetting "MyApp","Font","Size"
End Sub
```

Dialog Instruction/Function

Syntax	Dialog <i>dialogvar[, default]</i> -or- Dialog(<i>dialogvar[, default]</i>)
Group	User Input
Description	Display the dialog associated with <i>dialogvar</i> . The initial values of the dialog fields are provided by <i>dialogvar</i> . If the OK button or any push button is pressed then the fields in dialog are copied to the <i>dialogvar</i> . The Dialog() function returns a value indicating which button was pressed. (See the result table below.)
<hr/>	
Parameter	Description
<i>dlgvar</i>	This variable that holds the values of the fields in a dialog. Use <i>.field</i> to access individual fields in a dialog variable.
<i>default</i>	This numeric value indicates which button is the default button. (Pressing the Enter key on a non-button pushes the default button.) Use -2 to indicate that there is no default button. Other possible values are shown in the result table below. If this value is omitted then the first PushButton , OKButton or CancelButton is the default button.
<hr/>	
Result	Description
-1	OK button was pressed.
0	Cancel button was pressed.
>0	Nth push button was pressed.

See Also

Begin Dialog.

Example

```
'#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120
```

```

Text 10,10,180,15,"Please push the OK button"
OKButton 80,90,40,20
End Dialog
Dim dlg As UserDialog
Dialog dlg ' show dialog (wait for ok)
End Sub

```

DialogFunc Prototype

Syntax

```

Function dialogfunc(DlgItem$, Action%, SuppValue&) _
As Boolean
Select Case Actio'
Case 1 'Dialog box initialization
...
Case 2 'Value changing or button pressed
...
Case 3 'TextBox or ComboBox text changed
...
Case 4 'Focus changed
...
Case 5 'Idle
...
Case 6 'Function key
...
End Select
End Function

```

Group

Dialog Function

Description

A *dialogfunc* implements the dynamic dialog capabilities.

Parameter	Description
<i>DlgItem</i>	This string value is the name of the user dialog item's <i>field</i> .
<i>Action</i>	This numeric value indicates what action the dialog function is being asked to do.
<i>SuppValue</i>	This numeric value provides additional information for some actions.
Action	Description
1	Dialog box initialization. <i>DlgItem</i> is a null string. <i>SuppValue</i> is the dialog's window handle. Set <i>dialogfunc</i> = True to terminate the dialog.
2	CheckBox , DropListBox , ListBox , MultiListBox or OptionGroup : <i>DlgItem</i> 's value has changed. <i>SuppValue</i> is the new value. CancelButton , OKButton or PushButton : <i>DlgItem</i> 's button was pushed. <i>SuppValue</i> is meaningless. Set <i>dialogfunc</i> = True to prevent the dialog from closing.
3	ComboBox or TextBox : <i>DlgItem</i> 's text changed and losing focus. <i>SuppValue</i> is the number of characters.
4	Item <i>DlgItem</i> is gaining focus. <i>SuppValue</i> is the item that is losing focus. (The first item is 0, second is 1, etc.)
5	Idle processing. <i>DlgItem</i> is a null string. <i>SuppValue</i> is zero. Set <i>dialogfunc</i> = True to continue receiving idle actions. The idle action is called as often as possible. Use <i>Wait .1</i> to reduce the number of idle calls to 10 per second.
6	Function key (F1-F24) was pressed. <i>DlgItem</i> has the focus. <i>SuppValue</i> is the function key number and the shift/control/alt key state.

Regular function keys range from 1 to 24.
 Shift function keys have &H100 added.
 Control function keys have &H200 added.
 Alt function keys have &H400 added.
 (Alt-F4 closes the dialog and is never passed to the Dialog Function.)

See Also**Begin Dialog.****Example**

```
'#Language "WWB.NET"
Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,40,180,15,.Text
    OKButton 30,90,60,20
    PushButton 110,90,60,20,"&Hello"
  End Dialog
  Dim dlg As UserDialog
  Debug.Print Dialog(dlg)
End Sub

Function DialogFunc(DlgItem$, Action%, SuppValue&) As Boolean
  Debug.Print "Action="; Action'
  If Action' <> 1 And Action' <> 5 Then
    Debug.Print Dlgite"; "="; DlgText(Dlgite); """"
  End If
  Debug.Print "SuppValue="; SuppValue&
  Select Case Action'
    Case 1 ' Dialog box initialization
      Beep
    Case 2 ' Value changing or button pressed
      If Dlgite' = "Hello" Then
        MsgBox "Hello"
        DialogFunc = True 'do not exit the dialog
      End If
    Case 4 ' Focus changed
      Debug.Print "DlgFocus=""; DlgFocus(); """"
  End Select
End Function
```

Dim Definition

Syntax

`Dim [WithEvents] vardeclaration [= initialValue][, ...]`

Group

Declaration

Description

Dimension var array(s) using the *dimensions* to establish the minimum and maximum index value for each dimension. If the *dimensions* are omitted then a scalar (single value) variable is defined. A dynamic array is declared using () without any *dimensions*. It must be **ReDimensioned** before it can be used.

See Also [Begin Dialog](#), [Dialog](#), [Option Base](#), [Private](#), [Public](#), [ReDim](#), [Static](#), [WithEvents](#).

Example

```
'#Language "WWB.NET"
Sub Dolt(Size)
    Dim C0, C1(), C2(2,3)
    ReDim C1(Size+1) ' dynamic array
    C0 = 1
    C1(0) = 2
    C2(0,0) = 3
    Debug.Print C0; C1(0); C2(0,0) ' 1 2 3
End Sub

Sub Main
    Dolt 1
End Sub
```

Di' Function

Syntax Di'([*Patter*][, *AttribMask*])

Group File

Description Scan a directory for the first file matching *Patter*.

Parameter	Description
<i>Patter</i> '	This string value is the path and name of the file search pattern. If this is omitted then continue scanning with the previous pattern. Each <i>macro</i> has its own independent search. A path relative to the current directory can be used.
<i>AttribMask</i>	This numeric value controls which files are found. A file with an <i>attribute</i> that matches will be found.

See Also [GetAttr\(\)](#).

Example

```
'#Language "WWB.NET"
Sub Main
    ' = Di'("*.*)"
    While ' <> ""
        Debug.Print '
        ' = Di'()
    End While
End Sub
```

DirectCast Function

Syntax DirectCast(*expr*, *objtype*)

Group Conversion

Description Return *expr*'s type is related to *objtype* type. If it is not, a "type mismatch error" will occur.

Parameter	Description
<i>expr</i>	Cast the value of this expression.

<i>objtype</i>	Cast to this type.
----------------	--------------------

See Also **CType, TryCast.**

Example

```
'#Language "WWB.NET"
Sub Main
    Dim V As Object
    V = Err
    Debug.Print TypeName(DirectCast(V, ErrObject)) ' ErrObject
End Sub
```

DlgControlId Function

Syntax DlgControlId(*DlgItem*)
Group Dialog Function
Description Return the *field's* window id.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
<i>DlgItem</i>	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>fieldname</i> .

Example

```
'#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120,.DialogFunc
        Text 10,10,180,15,"Please push the OK button"
        TextBox 10,40,180,15,.Text
        OKButton 30,90,60,20
        PushButton 110,90,60,20,"&Hello"
    End Dialog
    Dim dlg As UserDialog
    Debug.Print Dialog(dlg)
End Sub

Function DialogFunc(DlgItem$, Action%, SuppValue&) As Boolean
    Debug.Print "Action="; Action'
    Select Case Action'
        Case 1 'Dialog box initialization
            Beep
        Case 2 ' Value changing or button pressed
            If DlgItem' = "Hello" Then
                DialogFunc = True 'do not exit the dialog
            End If
        Case 4 ' Focus changed
            Debug.Print "DlgFocus="""; DlgFocus(); """
            Debug.Print "DlgControlId("; DlgItem'; ")=";
            Debug.Print DlgControlId(DlgItem')
    End Select
End Function
```

DlgCount Function

Syntax	DlgCount()
Group	Dialog Function
Description	Return the number of dialog items in the dialog.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Example	<pre>'#Language "WWB.NET" Sub Main Begin Dialog UserDialog 200,120,.DialogFunc Text 10,10,180,15,"Please push the OK button" TextBox 10,40,180,15,.Text OKButton 30,90,60,20 End Dialog Dim dlg As UserDialog Dialog dlg End Sub Function DialogFunc(DlgItem\$, Action%, SuppValue&) As Boolean Debug.Print "Action="; Action' Select Case Action' Case 1 'Dialog box initialization Beep Debug.Print "DlgCount="; DlgCount() '3 End Select End Function</pre>
----------------	---

DlgEnable Instruction/Function

Syntax	DlgEnable <i>DlgItem</i> [, <i>Enable</i>] -or- DlgEnable(<i>DlgItem</i>)
Group	Dialog Function
Description	Instruction: Enable or disable <i>DlgItem</i> .

Function: Return **True** if *DlgItem* is enabled.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
<i>DlgItem</i>	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>field name</i> . Note: Use -1 to enable or disable all the dialog items at once.
<i>Enable</i>	It this numeric value is True then enable <i>DlgItem</i> . Otherwise, disable it. If this omitted then toggle it.

Example

```

'#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120,.DialogFunc
        Text 10,10,180,15,"Please push the OK button"
        TextBox 10,40,180,15,.Text
        OKButton 30,90,60,20
        PushButton 110,90,60,20,"&Disable"
    End Dialog
    Dim dlg As UserDialog
    Debug.Print Dialog(dlg)
End Sub

Function DialogFunc(DlgItem$, Action%, SuppValue&) As Boolean
    Debug.Print "Action="; Action'
    Select Case Action'
        Case 1 ' Dialog box initialization
            Beep
        Case 2 ' Value changing or button pressed
            Select Case DlgItem'
                Case "Disable"
                    DlgText DlgItem',"&Enable"
                    DlgEnable "Text",False
                    DialogFunc = True 'do not exit the dialog
                Case "Enable"
                    DlgText DlgItem',"&Disable"
                    DlgEnable "Text",True
                    DialogFunc = True 'do not exit the dialog
            End Select
        End Select
    End Function

```

DlgEnd Instruction

SyntaxDlgEnd *ReturnCode***Group**

Dialog Function

DescriptionSet the return code for the **Dialog** Function and close the user dialog.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
<i>ReturnCode</i>	Return this numeric value.

Example

```

'#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 210,120,.DialogFunc
        Text 10,10,190,15,"Please push the Close button"
        OKButton 30,90,60,20
        CheckBox 120,90,60,20,"&Close",.CheckBox1
    End Dialog
    Dim dlg As UserDialog
    Debug.Print Dialog(dlg)

```

```

End Sub

Function DialogFunc(DlgItem$, Action%, SuppValue&) As Boolean
  Debug.Print "Action="; Actio'
  Select Case Actio'
    Case 1 'Dialog box initialization
      Beep
    Case 2 ' Value changing or button pressed
      Select Case Dlgte'
        Case "CheckBox1"
          DlgEnd 1000
        End Select
      End Select
    End Function

```

DlgFocus Instruction/Function

Syntax

DlgFocus DlgItem

-or-

DlgFocu'()

Group

Dialog Function

Description

Instruction: Move the focus to this *DlgItem*.

Function: Return the *field* name which has the focus as a string.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
<i>DlgItem</i>	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>field</i> name.

Example

```

'#Language "WWB.NET"
Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,40,180,15,.Text
    OKButton 30,90,60,20
    PushButton 110,90,60,20,"&Hello"
  End Dialog
  Dim dlg As UserDialog
  Debug.Print Dialog(dlg)
End Sub

Function DialogFunc(DlgItem$, Action%, SuppValue&) As Boolean
  Debug.Print "Action="; Actio'
  Select Case Actio'
    Case 1 'Dialog box initialization
      Beep
    Case 2 ' Value changing or button pressed
      If Dlgte' = "Hello" Then
        MsgBox "Hello"

```

```

    DialogFunc = True 'do not exit the dialog
End If
Case 4 ' Focus changed
  Debug.Print "DlgFocus="""; DlgFocus(); """
End Select
End Function

```

DlgListBoxArray Instruction/Function

Syntax `DlgListBoxArray DlgItem, StrArra'()`

-or-

`DlgListBoxArray(DlgItem[, StrArra'()])`

Group Dialog Function

Description Instruction: Set the list entries for *DlgItem*.

Function: Return the number entries in *DlgItem*'s list.

This instruction/function must be called directly or indirectly from a *dialogfunc*.
 The *DlgItem* should refer to a **ComboBox**, **DropListBox**, **ListBox** or
MultiListBox.

Parameter	Description
<i>DlgItem</i>	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's field name.
<i>StrArra'</i> ()	Set the list entries of <i>DlgItem</i> . This one-dimensional array of strings establishes the list of choices. All the non-null elements of the array are used.

Example

```

'#Language "WWB.NET"
Dim lists()

Sub Main
  ReDim list(0)
  list(0) = "List 0"
  Begin Dialog UserDialog 200,119,.DialogFunc
    Text 10,7,180,14,"Please push the OK button"
    ListBox 10,21,180,63,lists(),list
    OKButton 30,91,40,21
    PushButton 110,91,60,21,"&Change"
  End Dialog
  Dim dlg As UserDialog
  dlg.list = 2
  Dialog dlg ' show dialog (wait for ok)
  Debug.Print dlg.list
End Sub

```

```

Function DialogFunc(DlgItem$, Action%, SuppValue&) As Boolean
Select Case Action
Case 2 'Value changing or button pressed
  If DlgItem$ = "Change" Then
    Dim N As Integer

```

```

N = UBound(list')+1
ReDim Preserve list'(N)
list'(N) = "List " & N
DlgListBoxArray "list",list'()
DialogFunc = True 'do not exit the dialog
End If
End Select
End Function

```

DlgName Function

Syntax DlgNam'(*DlgItem*)

Group Dialog Function

Description Return the *field name* of the *DlgItem* number.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
<i>DlgItem</i>	This numeric value is the dialog item number. The first item is 0, second is 1, etc.

Example

```

'#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120,,DialogFunc
        Text 10,10,180,15,"Please push the OK button"
        TextBox 10,40,180,15,,Text
        OKButton 30,90,60,20
    End Dialog
    Dim dlg As UserDialog
    Dialog dlg
End Sub

Function DialogFunc(DlgItem$, Action%, SuppValue&) As Boolean
    Debug.Print "Action="; Action'
    Select Case Action'
        Case 1 'Dialog box initialization
            Beep
            For I = 0 To DlgCount()-1
                Debug.Print I; DlgName(I)
            Next I
        End Select
    End Function

```

DlgNumber Function

Syntax DlgNumber(*DlgItc*)

Group Dialog Function

Description Return the number of the *DlgItc*. The first item is 0, second is 1, etc.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
DlgIte'	This string value is the dialog item's <i>field name</i> .

Example

```
'#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120,.DialogFunc
        Text 10,10,180,15,"Please push the OK button"
        TextBox 10,40,180,15,.Text
        OKButton 30,90,60,20
    End Dialog
    Dim dlg As UserDialog
    Dialog dlg
End Sub

Function DialogFunc(DlgItem$, Action%, SuppValue&) As Boolean
    Debug.Print "Action="; Action'
    Select Case Action'
        Case 1 ' Dialog box initialization
            Beep
        Case 4 ' Focus changed
            Debug.Print DlgIte'; "="; DlgNumber(DlgIte')
    End Select
End Function
```

DlgSetPicture Instruction

Syntax DlgSetPicture *DlgItem*, *FileName*, *Type***Group** Dialog Function**Description** Instruction: Set the file name for *DlgItem*.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
DlgItem	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>field name</i> .
FileName	Set the file name of <i>DlgItem</i> to this string value.
Type	This numeric value indicates the type of bitmap used. See below.

Type	Effect
0	<i>FileName</i> is the name of the bitmap file. If the file does not exist then "(missing picture)" is displayed.
3	The clipboard's bitmap is displayed. If the clipboard does not contain a bitmap then "(missing picture)" is displayed.
16	Same as 0, but instead of displaying "(missing picture)" a run-time error occurs.
19	Same as 3, but instead of displaying "(missing picture)" a run-time error occurs.

Example

```
'#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120,.DialogFunc
        Picture 10,10,180,75,"",0,.Picture
```

```

OKButton 30,90,60,20
PushButton 110,90,60,20,"&View"
End Dialog
Dim dlg As UserDialog
Debug.Print Dialog(dlg)
End Sub

Function DialogFunc(DlgItem$, Action%, SuppValue&) As Boolean
Debug.Print "Action="; Action'
Select Case Action'
Case 1 'Dialog box initialization
Beep
Case 2 'Value changing or button pressed
Select Case DlgItem'
Case "View"
  FileName = GetFilePath("Bitmap","BMP")
  DlgSetPicture "Picture",FileName,0
  DialogFunc = True 'do not exit the dialog
End Select
End Select
End Function

```

DlgText Instruction/Function

Syntax *DlgText DlgItem, Text*
 -*or*-
 DlgTex'(DlgItem)

Group Dialog Function

Description Instruction: Set the text for *DlgItem*.

Function: Return the text from *DlgItem*.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
<i>DlgItem</i>	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's field name. Note: Use -1 to access the dialog's title.
<i>Text</i>	Set the text of <i>DlgItem</i> to this string value.

Example

```

'#Language "WWB.NET"
Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,40,180,15,.Text
    OKButton 30,90,60,20
    PushButton 110,90,60,20,"&Now"
  End Dialog
  Dim dlg As UserDialog
  Debug.Print Dialog(dlg)
End Sub

```

```

Function DialogFunc(DlgItem$, Action%, SuppValue&) As Boolean
  Debug.Print "Action="; Actio'
  Select Case Actio'
    Case 1 'Dialog box initialization
      Beep
    Case 2 'Value changing or button pressed
      Select Case Dlgte'
        Case "Now"
          DlgText "Text",CStr(Now)
          DialogFunc = True 'do not exit the dialog
        End Select
      End Select
  End Function

```

DlgType Function

Syntax DlgTyp'(DlgItem)

Group Dialog Function

Description Return a string value indicating the type of the *DlgItem*. One of: "**CancelButton**", "**CheckBox**", "**ComboBox**", "**DropDownBox**", "**GroupBox**", "**ListBox**", "**MultiListBox**", "**OKButton**", "**OptionButton**", "**OptionGroup**", "**PushButton**", "**Text**", "**TextBox**".

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
DlgItem	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's field name.

Example

```

'#Language "WWB.NET"
Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,40,180,15,.Text
    OKButton 30,90,60,20
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg
End Sub

Function DialogFunc(DlgItem$, Action%, SuppValue&) As Boolean
  Debug.Print "Action="; Actio'
  Select Case Actio'
    Case 1 'Dialog box initialization
      Beep
      For I = 0 To DlgCount()-1
        Debug.Print I; DlgType(I)
      Next I
    End Select
  End Function

```

DlgValue Instruction/Function

Syntax	DlgValue <i>DlgItem</i> , <i>Value</i> -or- DlgValue(<i>DlgItem</i>)
Group	Dialog Function
Description	Instruction: Set the numeric value(s) <i>DlgItem</i> . Function: Return the numeric value(s) for <i>DlgItem</i> . (A MultiListBox user dialog item returns an array.)

This instruction/function must be called directly or indirectly from a *dialogfunc*.

The *DlgItem* should refer to a **CheckBox**, **ComboBox**, **DropListBox**, **ListBox**, **MultiListBox** or **OptionGroup**.

Parameter	Description
<i>DlgItem</i>	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's field name.
<i>Value</i>	Set the text of <i>DlgItem</i> to this numeric value. (A MultiListBox user dialog item uses an array.)

Example	<pre>'#Language "WWB.NET" Sub Main Begin Dialog UserDialog 150,147,.DialogFunc GroupBox 10,7,130,77,"Direction",.Field1 PushButton 100,28,30,21,"&Up" PushButton 100,56,30,21,"&Dn" OptionGroup .Direction OptionButton 20,21,80,14,"&North",.North OptionButton 20,35,80,14,"&South",.South OptionButton 20,49,80,14,"&East",.East OptionButton 20,63,80,14,"&West",.West OKButton 10,91,130,21 CancelButton 10,119,130,21 End Dialog Dim dlg As UserDialog Dialog dlg MsgBox "Direction=" & dlg.Direction End Sub Function DialogFunc(DlgItem\$, Action%, SuppValue&) As Boolean Select Case Action Case 1 ' Dialog box initialization Beep Case 2 ' Value changing or button pressed Select Case DlgItem Case "Up" DlgValue "Direction",0 DialogFunc = True 'do not exit the dialog Case "Dn" DlgValue "Direction",1 End Select End Select End Function</pre>
----------------	---

```

    DialogFunc = True 'do not exit the dialog
  End Select
End Select
End Function

```

DlgVisible Instruction/Function

Syntax DlgVisible *DlgItem*[, *Visible*]

-or-

DlgVisible(*DlgItem*)

Group Dialog Function

Description Instruction: Show or hide *DlgItem*.

Function: Return **True** if *DlgItem* is visible.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
<i>DlgItem</i>	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>fieldname</i> .
<i>Enable</i>	If this numeric value is True then show <i>DlgItem</i> . Otherwise, hide it. If this omitted then toggle it.

Example

```

'#Language "WWB.NET"
Sub Main
  Begin Dialog UserDialog 200,120,,DialogFunc
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,40,180,15,,Text
    OKButton 30,90,60,20
    PushButton 110,90,60,20,"&Hide"
  End Dialog
  Dim dlg As UserDialog
  Debug.Print Dialog(dlg)
End Sub

Function DialogFunc(DlgItem$, Action%, SuppValue&) As Boolean
  Debug.Print "Action="; Action'
  Select Case Action'
    Case 1 ' Dialog box initialization
      Beep
    Case 2 ' Value changing or button pressed
      Select Case DlgItem'
        Case "Hide"
          DlgText DlgItem',"&Show"
          DlgVisible "Text",False
          DialogFunc = True 'do not exit the dialog
        Case "Show"
          DlgText DlgItem',"&Hide"
          DlgVisible "Text",True
          DialogFunc = True 'do not exit the dialog
      End Select
  End Select
End Function

```

```
End Select
End Function
```

Do Statement

Syntax	<pre>Do <i>statements</i> Loop -or- Do {Until While} <i>condexpr</i> <i>statements</i> Loop -or- Do <i>statements</i> Loop {Until While} <i>condexpr</i></pre>
Group	Flow Control
Description	<p>Form 1: Do <i>statements</i> forever. The loop can be exited by using Exit or Goto.</p> <p>Form 2: Check for loop termination before executing the loop the first time.</p> <p>Form 3: Execute the loop once and then check for loop termination.</p>

Loop Termination:

- Until *condexpr*: Do *statements* until *condexpr* is **True**.
- While *condexpr*: Do *statements* while *condexpr* is **True**.

See Also	For, For Each, Exit Do, While.
-----------------	---------------------------------------

Example	<pre>'#Language "WWB.NET" Sub Main I = 2 Do I = I*2 Loop Until I > 10 Debug.Print I ' 16 End Sub</pre>
----------------	---

DoEvents Instruction

Syntax	DoEvents
Group	Miscellaneous
Description	This instruction allows other applications to process events.
Example	<pre>'#Language "WWB.NET" Sub Main</pre>

```
DoEvents ' let other apps work
End Sub
```

Double Data Type

Syntax	<code>Dim v As Double</code>
Group	Data Type
Description	A 64 bit real value.

DropDownBox Dialog Item Definition

Syntax	<code>DropDownBox X, Y, DX, DY, StrArra'(), .Field[, Options]</code>
Group	User Dialog
Description	Define a drop-down listbox item.

Parameter	Description
X	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
Y	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
DX	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
DY	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
StrArra'()	This one-dimensional array of strings establishes the list of choices. All the non-null elements of the array are used.
Field	The value of the drop-down list box is accessed via this field. It is the index of the StrArra'() var.
Options	This numeric value controls the type of drop-down list box. Choose one value from following table. (If this numeric value omitted then zero is used.)

Option	Description
0	Text box is not editable and list is not sorted.
1	Text box is editable and list is not sorted.
2	Text box is not editable and list is sorted.
3	Text box is editable and list is sorted.

See Also	Begin Dialog .
Example	

```
'#Language "WWB.NET"
Sub Main
    Dim list'(3)
    list'(0) = "List 0"
    list'(1) = "List 1"
    list'(2) = "List 2"
    list'(3) = "List 3"
    Begin Dialog UserDialog 200,120
```

```

Text 10,10,180,15,"Please push the OK button"
DropDownBox 10,25,180,60,list'(),list1
DropDownBox 10,50,180,60,list'(),list2,1
OKButton 80,90,40,20
End Dialog
Dim dlg As UserDialog
dlg.list1 = 2      'list1 is a numeric field
dlg.list2 = "xxx"  'list2 is a string field
Dialog dlg 'show dialog (wait for ok)
Debug.Print list'(dlg.list1)
Debug.Print dlg.list2
End Sub

```

Encode64 Function

Syntax `Encode6'(Data)`
 -or-
 `Encode64'(Data)`

Group Miscellaneous

Description Return a string using the base 64 encoding algorithm.

Parameter	Description
<code>Data</code>	Return this string's base 64 encoding.

See Also [Decode64](#).

Example '#Language "WWB.NET"
Sub Main
 Debug.Print Encode64("Hello World!!") "SGVsbG8gV29ybGQhIQ=="
End Sub

Encrypt64 Function

Syntax `Encrypt6'(Data [,Password])`

Group Miscellaneous

Description Return a string using the RC4 stream encryption algorithm. (The string is also encoded using base 64 encoding.)

Parameter	Description
<code>Data</code>	Return this string's encryption.
<code>Password</code>	Encrypt using this password.

See Also [Decode64](#), [Decrypt64](#), [Encode64](#).

Example '#Language "WWB.NET"
Sub Main
 Debug.Print Encrypt64("Hello World!!", "abc") "Y4GFrF+k1YUHwjEzsg=="
End Sub

End Instruction

Syntax	End
Group	Flow Control
Description	The end instruction causes the <i>macro</i> to terminate immediately. If the macro was run by another macro using the MacroRun instruction then that macro continues on the instruction following the MacroRun .
Example	<pre>'#Language "WWB.NET" Sub DoSub '= UCas'(InputBo("Enter End:")) If '=' = "END" Then End Debug.Print "End was not entered." End Sub Sub Main Debug.Print "Before DoSub" DoSub Debug.Print "After DoSub" End Sub</pre>

Enum Definition

Syntax	[Private Public]_ Enum <i>name</i> <i>elem</i> [= <i>value</i>] [...] End Enum
Group	Declaration
Description	Define a new <i>user enum</i> . Each <i>elem</i> defines an element of the enum. If <i>value</i> is given then that is the element's value. The value can be any constant integer expression. If <i>value</i> is omitted then the element's value is one more than the previous element's value. If there is no previous element then zero is used.
Access	If no access is specified then Public is assumed.
Example	<pre>'#Language "WWB.NET" Enum Days Monday Tuesday Wednesday Thursday Friday Saturday Sunday End Enum Sub Main</pre>

```

Dim D As Days
For D = Days.Monday To Days.Friday
    Debug.Print D ' 0 through 4
Next D
End Sub

```

Environ Function

Syntax	<code>Enviro'(<i>Index</i>)</code> -or- <code>Enviro'(<i>Name</i>)</code>						
Group	Miscellaneous						
Description	Return an environment string.						
Pocket PC	Not supported.						
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">Parameter</th><th style="text-align: left; padding: 2px;">Description</th></tr> </thead> <tbody> <tr> <td style="padding: 2px;"><i>Index</i></td><td style="padding: 2px;">Return this environment string's value. If there is no environment string at this index a null string is returned. Indexes start at one.</td></tr> <tr> <td style="padding: 2px;"><i>Name</i></td><td style="padding: 2px;">Return this environment string's value. If the environment string can't be found a null string is returned.</td></tr> </tbody> </table>	Parameter	Description	<i>Index</i>	Return this environment string's value. If there is no environment string at this index a null string is returned. Indexes start at one.	<i>Name</i>	Return this environment string's value. If the environment string can't be found a null string is returned.
Parameter	Description						
<i>Index</i>	Return this environment string's value. If there is no environment string at this index a null string is returned. Indexes start at one.						
<i>Name</i>	Return this environment string's value. If the environment string can't be found a null string is returned.						
Example	<pre> '#Language "WWB.NET" Sub Main Debug.Print Environ("Path") End Sub </pre>						

EOF Function

Syntax	<code>EOF(<i>StreamNum</i>)</code>				
Group	File				
Description	Return True if <i>StreamNum</i> is at the end of the file.				
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">Parameter</th><th style="text-align: left; padding: 2px;">Description</th></tr> </thead> <tbody> <tr> <td style="padding: 2px;"><i>StreamNum</i></td><td style="padding: 2px;">Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.</td></tr> </tbody> </table>	Parameter	Description	<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
Parameter	Description				
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.				
Example	<pre> '#Language "WWB.NET" Sub Main FileOpen 1, "XXX", OpenMode.Input While Not EOF(1) Dim L As String L = LineInput(1) Debug.Print L End While FileClose'1 End Sub </pre>				

Erase Instruction

Syntax	<code>Erase arrayvar[, ...]</code> -or- <code>Erase userstructurevar.elem[, ...]</code>
Group	Assignment
Description	Reset <i>arrayvar</i> or <i>user defined structure</i> array element to zero. (Dynamic arrays are reset to undimensioned arrays.) String arrays values are set to a null string. <i>arrayvar</i> must be declared as an array. <ul style="list-style-type: none"> • Declare with Dim, Private, Public or Static. • Declare as a parameter of Sub, Function or Property definition.
Example	<pre>#Language "WWB.NET" Sub Main Dim '(2) '(1) = 1 Erase ' Debug.Print '(1) ' 0 End Sub</pre>

Err Object

Syntax	<code>Err</code>
Group	Error Handling
Description	Set Err to zero to clear the last error event. Err in an expression returns the last error code. Add vbObjectError to your error number in ActiveX Automation objects. Use Err.Raise or Error to trigger an error event.
	Err.Number This is the error code for the last error event. Set it to zero (or use Err.Clear) to clear the last error condition. Use Error or Err.Raise to trigger an error event. This is the default property.
	Err.Description This string is the description of the last error event.
	Err.Source This string is the error source file name of the last error event.
	Err.HelpFile This string is the help file name of the last error event.
	Err.HelpContext This number is the help context id of the last error event.

Err.Clear

Clear the last error event.

```
Err.Raise [Number:=]errorcode _
[, [Source:=]source] _
[, [Description:=]errordesc] _
[, [HelpFile:=]helpfile] _
[, [HelpContext:=]context]
```

Raise an error event.

Err.LastDLLError

Returns the error code for the last DLL call (see **Declare**).

Example

```
'#Language "WWB.NET"
Sub Main
    On Error GoTo Problem
    Err = 1 ' set to error #1 (handler not triggered)
    Exit Sub

    Problem: ' error handler
    Error Err ' halt macro with message
End Sub
```

Error Instruction

Syntax *Error ErrorCode*

Group Error Handling

Description Signal error *ErrorCode*. This triggers error handling just like a real error. The current *procedure*'s error handler is activated, unless it is already active or there isn't one. In that case the calling *procedure*'s error handler is tried. (Use **Err.Raise** to provide complete error information.)

Parameter	Description
<i>ErrorCode</i>	This is the error number.

Example

```
'#Language "WWB.NET"
Sub Main
    On Error GoTo Problem
    Error 1 ' simulate error #1
    Exit Sub

    Problem: ' error handler
    Debug.Print "Err.Description="; Err.Description
    Resume Next
End Sub
```

ErrorToString Function

Syntax `ErrorToString(ErrorCode)`

Group Error Handling

Description The `ErrorToString` function returns the error text string.

Parameter	Description
<code>ErrorCode</code>	This is the error number.

Example

```
'#Language "WWB.NET"
Sub Main
    On Error GoTo Problem
    Error 1 ' simulate error #1
    Exit Sub

    Problem: ' error handler
    Debug.Print "Description="; ErrorToString(Err.Num)
    Resume Next
End Sub
```

Eval Function

Syntax `Eval(Expr[, Depth])`

Group Miscellaneous

Description Return the value of the string expression as evaluated.

Parameter	Description
<code>Expr</code>	Evaluate this string value.
<code>Depth</code>	This integer value indicates how deep into the stack to locate the local variables. If <code>Depth = 0</code> then use the current <i>procedure</i> . If this value is omitted then the depth is 0.

See Also

Assign.

Example

```
'#Language "WWB.NET"
Sub Main
    Dim X As String
    X = "Hello"
    Debug.Print Eval("X") 'Hello
    A
End Sub
Sub A
    Dim X As String
    X = "Bye"
    Debug.Print Eval("X") 'Bye
    Debug.Print Eval("X",1) 'Hello
End Sub
```

Event Definition

Syntax

```
[| Public |_
Event name([param[, ...]])]
```

Group

Declaration

Description

User defined event. The event defines a sub that can be defined using **WithEvents**. The values of the calling *arglist* are assigned to the *params*.

Access

If no access is specified then **Public** is assumed.

See Also

RaiseEvent.

Example

```
' Class1
'#Language "WWB.NET"
Event Changing(ByVal OldValue As String, ByVal NewValue As String)
```

Private Value_ As String

```
Property Get Value As String
    Value = Value_
End Property
```

```
Property Let Value(ByVal NewValue As String)
    RaiseEvent Changing(Value_, NewValue)
    Value_ = NewValue
End Property
```

#Uses "Class1.cls"

Dim WithEvents c1 As Class1

```
Sub Main
    'c1 = New Class1
    c1.Value = "Hello"
    c1.Value = "Goodbye"
End Sub
```

```
Sub c1_Changing(ByVal OldValue As String, ByVal NewValue As String) Handles
    c1.Changing
    Debug.Print "OldValue=""" & OldValue & """", NewValue=""" & NewValue & """"
End Sub
```

Exit Instruction

Syntax

Exit {All|Do|For|Function|Property|Sub|Try|While}

Group

Flow Control

Description

The exit instruction causes the *macro* to continue with out doing some or all of the remaining instructions.

Exit	Description
------	-------------

All	Exit all <i>macros</i> .
Do	Exit the Do loop.
For	Exit the For or For Each loop.
Function	Exit the Function block. Note: This instruction clears the Err and sets Error`\$ to null.
Property	Exit the Property block. Note: This instruction clears the Err and sets Error`\$ to null.
Sub	Exit the Sub block. Note: This instruction clears the Err and sets Error`\$ to null.
Try	Exit the Try block.
While	Exit the While loop.

Example

```
'#Language "WWB.NET"
Sub Main
    '= InputBo'("Enter Do, For, While, Sub or All:")
    Debug.Print "Before DoSub"
    DoSub UCas(')
    Debug.Print "After DoSub"
End Sub

Sub DoSub()
    Do
        If ' = "DO" Then Exit Do
        I = I+1
    Loop While I < 10
    If I = 0 Then Debug.Print "Do was entered"

    For I = 1 To 10
        If ' = "FOR" Then Exit For
    Next I
    If I = 1 Then Debug.Print "For was entered"

    I = 10
    While I > 0
        If ' = "WHILE" Then Exit While
        I = I-1
    End While
    If I = 10 Then Debug.Print "While was entered"

    If ' = "SUB" Then Exit Sub
    Debug.Print "Sub was not entered."
    If ' = "ALL" Then Exit All
    Debug.Print "All was not entered."
End Sub
```

Exp Function

Syntax `Exp(Num)`**Group** Math**Description** Return the exponential.

Parameter	Description
Num	Return e raised to the power of this numeric value. The value e is approximately

2.718282.

See Also**Log.****Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Exp(1) ' 2.718281828459
End Sub
```

False Keyword

Group

Constant

Description

A *condexpr* is false when its value is zero. A function that returns False returns the value 0.

FileAttr Function

Syntax`FileAttr(StreamNum, ReturnValue)`**Group**

File

Description

Return *StreamNum*'s open mode or file handle.

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
<i>ReturnValue</i>	1 - return the mode used to open the file: 1=Input, 2=Output, 4=Random, 8=Append, 32=Binary 2 - return the file handle

See Also**Open.****Example**

```
'#Language "WWB.NET"
Sub Main
    FileOpen 1, "XXX", OpenMode.Output
    Debug.Print FileAttr(1,1) ' 2
    FileClose'1
End Sub
```

FileClose Instruction

Syntax`FileClose [StreamNum][, ...]`**Group**

File

Description

Close *StreamNums*.

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros. If this is omitted then all open streams for the current

macro/module are closed.

See Also**FileOpen, Reset.****Example**

```
'#Language "WWB.NET"
Sub Main
    ' read the first line of XXX and print it
    FileOpen 1, "XXX", OpenMode.Input
    Dim L As String
    LineInput 1, L
    Debug.Print L
    FileClose 1
End Sub
```

FileCopy Instruction

Syntax `FileCopy FromNam', ToNam'`**Group** File**Description** Copy a file.

Parameter	Description
<i>FromNam'</i>	This string value is the path and name of the source file. A path relative to the current directory can be used.
<i>ToNam'</i>	This string value is the path and name of the destination file. A path relative to the current directory can be used.

Example

```
'#Language "WWB.NET"
Sub Main
    FileCopy "C:\AUTOEXEC.BAT","C:\AUTOEXEC.BAK"
End Sub
```

FileDateTime Function

Syntax `FileDateTime(Nam')`**Group** File**Description** Return the date and time file *Nam'* was last changed as a **Date** value. If the file does not exist then a run-time error occurs.

Parameter	Description
<i>Nam'</i>	This string value is the path and name of the file. A path relative to the current directory can be used.

Example

```
'#Language "WWB.NET"
Sub Main
    ' = Di'("*.*)'
    While ' <> ""
        Debug.Print '; " "; FileDateTime(')
        ' = Di'()
```

```

End While
End Sub

```

FileLen Function

Syntax	FileLen(<i>Nam</i>)
Group	File
Description	Return the length of file <i>Nam</i> '. If the file does not exist then a run-time error occurs.
	<hr/>
	Parameter Description
	<i>Nam'</i> This string value is the path and name of the file. A path relative to the current directory can be used.
Example	<pre> '#Language "WWB.NET" Sub Main '= Di'("*.*") While ' <> "" Debug.Print ', " "; FileLen(') '= Di'() End While End Sub </pre>

FileOpen Instruction

Syntax	FileOpen <i>StreamNum</i> , <i>Nam</i> ', <i>mode</i> , [<i>access</i>], [<i>lock</i>], [<i>RecordLen</i>]
Group	File
Description	Open file <i>Nam</i> ' for <i>mode</i> as <i>StreamNum</i> .
Reading Unicode Files	To read a Unicode (UTF-16 or UTF-8) file, just open using Input mode. All the input is converted automatically.
Writing Unicode Files	To write a Unicode (UTF-16 or UTF-8) file, open using Output or Append mode and write the appropriate Byte Order Mark (BOM). All the printed output is converted automatically.

To create a Unicode (UTF-16) text file use:

```

FileOpen fn, FileName, OpenMode.Output
Print fn, vbUTF16BOM ' first char is the UTF-16 Byte Order Mark
... ' everything automatically converted to UTF-16
FileClose fn

```

To create a Unicode (UTF-8) text file use:

```

FileOpen fn, FileName, OpenMode.Output
Print fn, vbUTF8BOM ' first three chars are the UTF-8 Byte Order Mark
... ' everything automatically converted to UTF-8
FileClose fn

```

Parameter	Description
	<hr/>

<i>Nam'</i>	This string value is the path and name of the file. A path relative to the current directory can be used.
<i>mode</i>	May be Input (1), Output (2), Append (8), Binary (4) or Random (32).
<i>access</i>	May be Read (1), Write (2) or Read Write (3). If omitted then Read Write is used.
<i>lock</i>	May be Shared (0), Lock Read (2), Lock Write (1) or Lock Read Write (3). If omitted then Shared is used.
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
<i>RecordLen</i>	This numeric value is the record length for Random mode files. Other file modes ignore this value.

See Also**FileClose, FileAttr, FreeFile, Reset.****Example**

```
#Language "WWB.NET"
Sub Main
    FileOpen 1, "XXX", OpenMode.Output
    PrintLine 1, "1,2,""Hello"""
    FileClose 1
End Sub
```

Fix Function

Syntax**Fix(Num)****Group**

Math

Description

Return the integer value.

Parameter	Description
<i>Num</i>	Return the integer portion of this numeric value. The number is truncated. Positive numbers return the next lower integer. Negative numbers return the next higher integer.'

See Also**Int.****Example**

```
#Language "WWB.NET"
Sub Main
    Debug.Print Fix(9.9) '9
    Debug.Print Fix(0) '0
    Debug.Print Fix(-9.9) '-9
End Sub
```

For Statement

Syntax

```
For Num [As type] = First To Last[Step Inc]
    statements
Next [Num]
```

Group

Flow Control

DescriptionExecute *statements* while *Num* is in the range *First* to *Last*.

Parameter	Description
<i>Num</i>	This is the iteration variable.

<i>As type</i>	Optional. The iteration variable can be declared here which eliminates the need for a separate Dim statement.
<i>First</i>	Set <i>Num</i> to this value initially.
<i>Last</i>	Continue looping while <i>Num</i> is in the range. See <i>Step</i> below.
<i>Step</i>	If this numeric value is greater than zero then the for loop continues as long as <i>Num</i> is less than or equal to <i>Last</i> . If this numeric value is less than zero then the for loop continues as long as <i>Num</i> is greater than or equal to <i>Last</i> . If this is omitted then one is used.

See Also [Do, For Each, Exit For, While.](#)

Example

```
'#Language "WWB.NET"
Sub Main
    For I = 1 To 2000 Step 100
        Debug.Print I; I+I; I*I
    Next I
End Sub
```

For Each Statement

Syntax **For Each** *var* [*As type*] In *items*
statements
Next [*var*]

Group Flow Control

Description Execute *statements* for each item in *items*.

Parameter	Description
<i>var</i>	This is the iteration variable.
<i>As type</i>	Optional. The iteration variable can be declared here which eliminates the need for a separate Dim statement.
<i>items</i>	This is the collection of items to be done.

See Also [Do, For, Exit For, While.](#)

Example

```
'#Language "WWB.NET"
Sub Main
    Dim Document As Object
    For Each Document In App.Documents
        Debug.Print Document.Title
    Next Document
End Sub
```

Forma' Function

Syntax **Forma'**(*expr*[, *for*], [*firstday*], [*firstweek*])

Group String

Description Return the formatted string representation of *expr*.

Parameter	Description
-----------	-------------

<i>expr</i>	Return the formatted string representation of this numeric value.
<i>form</i>	Format <i>expr</i> using to this string value. If this is omitted then return the <i>expr</i> as a string.
<i>firstday</i>	Format using this day as the first day of the week. If this is omitted then the vbSunday is used.
<i>firstweek</i>	Format using this week as the first week of the year. If this is omitted then the vbFirstJan1 is used.

firstday	Value	Description
vbUseSystemFirstDay	0	Use the systems first day of the week.
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

firstweek	Value	Description
vbUseSystem	0	Use the systems first week of the year.
vbFirstJan1	1	The week that January 1 occurs in. This is the default value.
2	vbFirstFourDays	The first week that has at least four days in the year.
3	vbFirstFullWeek	The first week that entirely in the year.

See Also

Predefined Date Format, Predefined Number Format, User defined Date Format, User defined Number Format, User defined Text Format.

Format Predefined Date

Description

The following predefined date formats may be used with the **Format** function. Predefined formats may not be combined with user defined formats or other predefined formats.

Form	Description
General Date	Same as user defined date format "c"
Long Date	Same as user defined date format "dddddd"
Medium Date	Not supported at this time.
Short Date	Same as user defined date format "ddddd"
Long Time	Same as user defined date format "tttt"
Medium Time	Same as user defined date format "hh:mm AMPM"
Short Time	Same as user defined date format "hh:mm"

Format Predefined Number

Description

The following predefined number formats may be used with the **Format** function. Predefined formats may not be combined with user defined formats or other

predefined formats.

Form	Description
General Number	Return number as is.
Currency	Same as user defined number format '#,##0.00; '#,##0.00)" Not locale dependent at this time.
Fixed	Same as user defined number format "0.00".
Standard	Same as user defined number format "#,##0.00".
Percent	Same as user defined number format "0.00%".
Scientific	Same as user defined number format "0.00E+00".
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".

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Forma'(2.145,"Standard") ' 2.15
End Sub
```

Format User Defined Date

Description The following date formats may be used with the **Format** function. Date formats may be combined to create the user defined date format. User defined date formats may not be combined with other user defined formats or predefined formats.

Parameter	Description
:	insert localized time separator
/	insert localized date separator
c	insert dddd tttt, insert date only if t=0, insert time only if d=0
d	insert day number without leading zero
dd	insert day number with leading zero
ddd	insert abbreviated day name
dddd	insert full day name
ddddd	insert date according to Short Date format
dddddd	insert date according to Long Date format
w	insert day of week number
ww	insert week of year number
m	insert month number without leading zero insert minute number without leading zero (if follows h or hh)
mm	insert month number with leading zero insert minute number with leading zero (if follows h or hh)
mmm	insert abbreviated month name
mmmm	insert full month name
q	insert quarter number
y	insert day of year number
yy	insert year number (two digits)
yyy	insert year number (four digits, no leading zeros)
h	insert hour number without leading zero

hh	insert hour number with leading zero
n	insert minute number without leading zero
nn	insert minute number with leading zero
s	insert second number without leading zero
ss	insert second number with leading zero
tttt	insert time according to time format
AM/PM	use 12 hour clock and insert AM (hours 0 to 11) and PM (12 to 23)
am/pm	use 12 hour clock and insert am (hours 0 to 11) and pm (12 to 23)
A/P	use 12 hour clock and insert A (hours 0 to 11) and P (12 to 23)
a/p	use 12 hour clock and insert a (hours 0 to 11) and p (12 to 23)
AMPM	use 12 hour clock and insert localized AM/PM strings
\c	insert character c
"text"	insert literal text

Example

Format User Defined Number

Description

The following number formats may be used with the **Format** function. Number formats may be combined to create the user defined number format. User defined number formats may not be combined with other user defined formats or pre-defined formats.

User defined number formats can contain up to four sections separated by ';':

- form - format for non-negative expr, '-format for negative expr, empty and null expr return ""'
- form;negform - negform: format for negative expr
- form;negform;zeroform - zeroform: format for zero expr
- form;negform;zeroform;nullform - nullform: format for null expr

Parameter	Description
#	digit, don't include leading/trailing zero digits (all the digits left of decimal point are returned) eg. Format(19,"##") returns "19" eg. Format(19,"#") returns "19"
0	digit, include leading/trailing zero digits eg. Format(19,"000") returns "019" eg. Format(19,"0") returns "19"
.	decimal, insert localized decimal point eg. Format(19.9,"###.00") returns "19.90" eg. Format(19.9,"###.#") returns "19.9"
,	thousands, insert localized thousand separator every 3 digits "xxx," or "xxx." mean divide expr by 1000 prior to formatting two adjacent commas "," means divide expr by 1000 again eg. Format(1900000,"0,,") returns "2" eg. Format(1900000,"0,,0") returns "1.9"
%	percent, insert %, multiply expr by 100 prior to formatting
:	insert localized time separator
/	insert localized date separator

E+ e+ E- e-	use exponential notation, insert E (or e) and the signed exponent eg. Format(1000,"0.00E+00") returns "1.00E+03" eg. Format(.001,"0.00E+00") returns "1.00E-03"
- +'() space	insert literal char eg. Format(10,'#") returns '10"
\c	insert character c eg. Format(19,"#####\#") returns "#19#"
"text"	insert literal text eg. Format(19,"####/# #####/# #####/#") returns "##19##"

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Forma'(2.145,"#.00") ' 2.15
End Sub
```

Format User Defined Text

Description

The following text formats may be used with the **Format** function. Text formats may be combined to create the user defined text format. User defined text formats may not be combined with other user defined formats or predefined formats.

User defined text formats can contain one or two sections separated by ';':

- form - format for all strings
- form;nullform - nullform: format for empty and null strings

Parameter	Description
@	char placeholder, insert char or space
&	char placeholder, insert char or nothing
<	all chars lowercase
>	all chars uppercase
!	fill placeholder from left-to-right (default is right-to-left)
\c	insert character c
"text"	insert literal text

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Format("123","ab@c") " ab1c23"
    Debug.Print Format("123","!ab@c") " ab3c"
End Sub
```

FreeFile Function

Syntax

FreeFile[()]

Group

File

Description	Return the next unused shared stream number (greater than or equal to 256). Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print FreeFile ' 256 FN = FreeFile FileOpen FN, "XXX", OpenMode.Output Debug.Print FreeFile ' 257 FileClose'FN Debug.Print FreeFile ' 256 End Sub</pre>

Friend Keyword

Group	Declaration
Description	Friend Functions , Properties and Subs in a <i>module</i> are available in all other <i>macros/modules</i> that access it. Friends are not accessible via Object variables.

Function Definition

Syntax	<pre>[Private Public Friend]_ [Default]_ Function name[type][([param[, ...]])]_ [As type()] [Handles var.eventname] statements End Function</pre>
Group	Declaration
Description	User defined function. The function defines a set of <i>statements</i> to be executed when it is called. The values of the calling <i>arglist</i> are assigned to the <i>params</i> . Assigning to <i>name[type]</i> sets the value of the function result.
Access	If no access is specified then Public is assumed.
Handles	The Function provides an event handler for the WithEvents variable's (<i>var</i>) event (<i>eventname</i>).
See Also	Declare , Property , Sub , WithEvents .
Example	<pre>'#Language "WWB.NET" Function Power(X,Y) P = 1 For I = 1 To Y P = P*X Next I Power = P End Function</pre>

```

Sub Main
    Debug.Print Power(2,8) ' 256
End Sub

```

Get Instruction

Syntax	Get' StreamNum, [RecordNum], var										
Group	File										
Description	Get a variable's value from <i>StreamNum</i> .										
<table border="1"> <thead> <tr> <th>Parameter</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>StreamNum</i></td><td>Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.</td></tr> <tr> <td><i>RecordNum</i></td><td>For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1. If this is omitted then the current position (or record number) is used.</td></tr> <tr> <td><i>var</i></td><td>This variable value is read from the file. For a fixed length variable (like Long) the number of bytes required to restore the variable are read. For a Variant variable two bytes are read which describe its type and then the variable value is read accordingly. For a <i>user structure</i> variable each field is read in sequence. For an array variable each element is read in sequence. For a dynamic array variable the number of dimensions and range of each dimension is read prior to reading the array values. All binary data values are read from the file in <i>little-endian</i> format.</td></tr> <tr> <td colspan="2">Note: When reading a string (or a dynamic array) from a Binary mode file the length (or array dimension) information is not read. The current string length determines how much string data is read. The current array dimension determines how many array elements are read.</td></tr> </tbody> </table>		Parameter	Description	<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.	<i>RecordNum</i>	For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1. If this is omitted then the current position (or record number) is used.	<i>var</i>	This variable value is read from the file. For a fixed length variable (like Long) the number of bytes required to restore the variable are read. For a Variant variable two bytes are read which describe its type and then the variable value is read accordingly. For a <i>user structure</i> variable each field is read in sequence. For an array variable each element is read in sequence. For a dynamic array variable the number of dimensions and range of each dimension is read prior to reading the array values. All binary data values are read from the file in <i>little-endian</i> format.	Note: When reading a string (or a dynamic array) from a Binary mode file the length (or array dimension) information is not read. The current string length determines how much string data is read. The current array dimension determines how many array elements are read.	
Parameter	Description										
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.										
<i>RecordNum</i>	For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1. If this is omitted then the current position (or record number) is used.										
<i>var</i>	This variable value is read from the file. For a fixed length variable (like Long) the number of bytes required to restore the variable are read. For a Variant variable two bytes are read which describe its type and then the variable value is read accordingly. For a <i>user structure</i> variable each field is read in sequence. For an array variable each element is read in sequence. For a dynamic array variable the number of dimensions and range of each dimension is read prior to reading the array values. All binary data values are read from the file in <i>little-endian</i> format.										
Note: When reading a string (or a dynamic array) from a Binary mode file the length (or array dimension) information is not read. The current string length determines how much string data is read. The current array dimension determines how many array elements are read.											

See Also

Open, Put.

Example

```

'#Language "WWB.NET"
Sub Main
    Dim V As Variant
    FileOpen 1, "SAVE_V.DAT", OpenMode.Binary, OpenAccess.Read
    Get'1, , V
    FileClose'1
End Sub

```

GetAllSettings Function

Syntax	GetAllSettings(<i>AppNam'</i> , <i>Sectio'</i>)
Group	Settings
Description	Get all of <i>Section</i> 's settings in project <i>AppName</i> . Settings are returned in a Object . Nothing is returned if there are no keys in the section. Otherwise, the Object contains a two dimension array: (I,0) is the key and (I,1) is the setting. Win16 and Win32s store settings in a .ini file named <i>AppName</i> . Win32/Win64 store settings in the registration database under "HKEY_CURRENT_USER\

Software\ VB and VBA Program Settings\ *AppName*\ *Section*\ *Key*". If *AppName* starts with "..\" then "VB and VBA Program Settings\" is omitted.

Parameter	Description
<i>AppNam'</i>	This string value is the name of the project which has this <i>Section</i> and <i>Key</i> .
<i>Sectio'</i>	This string value is the name of the section of the project settings.

Example

```
'#Language "WWB.NET"
Sub Main
    SaveSetting "MyApp","Font","Size",10
    SaveSetting "MyApp","Font","Name","Courier"
    Settings = GetAllSettings("MyApp","Font")
    For I = LBound(Settings) To UBound(Settings)
        Debug.Print Settings(I,0); "="; Settings(I,1)
    Next I
    DeleteSetting "MyApp" "Font"
End Sub
```

GetAttr Function

SyntaxGetAttr(*Nam*)**Group**

File

Description

Return the *attributes* for file *Nam*'. If the file does not exist then a run-time error occurs.

Parameter	Description
<i>Nam'</i>	This string value is the path and name of the file. A path relative to the current directory can be used.

Example

```
'#Language "WWB.NET"
Sub Main
    ' = Di('*. *')
    While ' <> ""
        Debug.Print '; " "; GetAttr(')
        ' = Di(')
    End While
End Sub
```

GetChar Function

SyntaxGetChar(*Str*, *Index*)**Group**

String

Description

Return the character at *Index*. The first char is at Index=1.

Parameter	Description
<i>Str</i>	Index into this string value.
<i>Index</i>	This is the index into the string value.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print GetChar("abcd",2) "b"
End Sub
```

GetFilePat' Function

Syntax

```
GetFilePat'([DefNam], [DefEx], [DefDi], _
    [Tit], [Option])
```

Group

User Input

Description

Put up a dialog box and get a file path from the user. The returned string is a complete path and file name. If the cancel button is pressed then a null string is returned.

Parameter	Description
DefNam'	Set the initial File Name in the to this string value. If this is omitted then *.DefEx' is used.
DefEx'	Initially show files whose extension matches this string value. (Multiple extensions can be specified by using ";" as the separator.) If this is omitted then * is used.
	A "filter" may be specified using "description *.ext ". Multiple descriptions can be used by repeating this format. (e.g. "Bitmap files *.bmp PNG files *.png JPEG files *.jpg")
DefDi'	This string value is the initial directory. If this is omitted then the current directory is used.
Tit'	This string value is the title of the dialog. If this is omitted then "Get File Path" is used.
Option	This numeric value determines the file selection options. If this is omitted then zero is used. See table below.

Option	Effect
0	Only allow the user to select a file that exists.
1	Confirm creation when the user selects a file that does not exist.
2	Allow the user to select any file whether it exists or not.
3	Confirm overwrite when the user selects a file that exists.
+4	Selecting a different directory changes the application's current directory.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print GetFilePat'()
End Sub
```

GetObject Function

Syntax

```
GetObject([Fil][, Clas])
```

Group

Object

Description

Get an existing object of type *Clas'* from *Fil'*.

Pocket PC

Not supported.

Parameter	Description
<i>File'</i>	This is the file where the object resides. If this is omitted then the currently active object for <i>Class</i> ' is returned.
<i>Class'</i>	This string value is the application's registered class name. If this application is not currently active it will be started. If this is omitted then the application associated with the file's extension will be started.

Example

```
'#Language "WWB.NET"
Sub Main
    Dim App As Object
    'App = GetObject("WinWrap.CppDemoApplication")
    App.Move 20,30 ' move icon to 20,30
    'App = Nothing
    App.Quit      ' run-time error (no object)
End Sub
```

GetLocale Function

Syntax

GetLocale

Group

Miscellaneous

Description

Get the locale ID for the current thread.

See Also**SetLocale.****Example**

```
'#Language "WWB.NET"
Sub Main
    SetLocale &H409 ' English, US
    Debug.Print Hex(GetLocale) "409"
End Sub
```

GetSetting Function

SyntaxGetSetting('AppName', *Section*', *Key*[, *Default*'])**Group**

Settings

Description

Get the setting for *Key* in *Section* in project *AppName*. Win16 and Win32 store settings in a .ini file named *AppName*. Win32/Win64 store settings in the registration database under "HKEY_CURRENT_USER\ Software\ VB and VBA Program Settings\ *AppName*\ *Section*\ *Key*". If *AppName* starts with "..\" then "VB and VBA Program Settings\" is omitted.

Parameter	Description
<i>AppName'</i>	This string value is the name of the project which has this <i>Section</i> and <i>Key</i> .
<i>Section'</i>	This string value is the name of the section of the project settings.
<i>Key'</i>	This string value is the name of the key in the section of the project settings.
<i>Default'</i>	Return this string value if no setting has been saved. If this is omitted then a null string is used.

Example

```
'#Language "WWB.NET"
Sub Main
    SaveSetting "MyApp","Font","Size",10
    Debug.Print GetSetting("MyApp","Font","Size") ' 10
End Sub
```

GetType Operator

Syntax `GetType(objtype)`

Group Operator

Description Return the .NET Type object for *objtype*.

This operator does not support user defined **Enums**, **Structures**, **Class Modules** or **Object Modules**.

See Also [Objects](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print GetType(Integer).Name "Integer"
End Sub
```

Goto Instruction

Syntax `GoTo label`

Group Flow Control

Description Go to the *label* and continue execution from there. Only *labels* in the current user defined *procedure* are accessible.

Example

```
'#Language "WWB.NET"
Sub Main
    X = 2
    Loop:
        X = X*X
        If X < 100 Then GoTo Loop
        Debug.Print X ' 256
End Sub
```

GroupBox Dialog Item Definition

Syntax `GroupBox X, Y, DX, DY, Title[, .Field]`

Group User Dialog

Description Define a groupbox item.

Parameter	Description
-----------	-------------

<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Title</i>	This string value is the title of the group box.
<i>Field</i>	This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i> . If this identifier is omitted then the first two words of the title are used.

See Also**Begin Dialog.****Example**

```
'#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120
        Text 10,10,180,15,"Please push the OK button"
        GroupBox 10,25,180,60,"Group box"
        OKButton 80,90,40,20
    End Dialog
    Dim dlg As UserDialog
    Dialog dlg ' show dialog (wait for ok)
End Sub
```

He' Function

Syntax**He'(Num)****Group**

String

Description

Return a hex string.

Parameter	Description
<i>Num</i>	Return a hex encoded string for this numeric value.

See Also**Oc'(), St'(), Val().****Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print He'(15) 'F
End Sub
```

Hour Function

Syntax**Hour(*dateexpr*)****Group**

Time/Date

Description

Return the hour of the day (0 to 23).

Parameter	Description
<i>dateexpr</i>	Return the hour of the day for this date value. '

See Also	Minute(), Second(), Time().
Example	'#Language "WWB.NET" Sub Main Debug.Print Hour(#12:00:01 AM#) ' 0 End Sub

If Statement

Syntax	If <i>condexpr</i> Then [<i>instruction</i>] [Else <i>instruction</i>] -or- If <i>condexpr</i> Then <i>statements</i> [Elseif <i>condexpr</i> Then <i>statements</i>][...] [Else <i>statements</i>] End If
Group	Flow Control
Description	<p>Form 1: Single line if statement. Execute the <i>instruction</i> following the Then if <i>condexpr</i> is True. Otherwise, execute the <i>instruction</i> following the Else. The Else portion is optional.</p> <p>Form 2: The multiple line if is useful for complex ifs. Each if <i>condexpr</i> is checked in turn. The first True one causes the following <i>statements</i> to be executed. If all are False then the Else's <i>statements</i> are executed. The ElseIf and Else portions are optional.</p> <p>Form 3: If <i>objexpr</i>'s type is the same type or a type descended from <i>objtype</i> the Then portion is executed.</p>

See Also	Select Case, Choose(), IIf().
Example	'#Language "WWB.NET" Sub Main S = InputBox("Enter hello, goodbye, dinner or sleep:") S = UCase(S) If S = "HELLO" Then Debug.Print "come in" If S = "GOODBYE" Then Debug.Print "see you later" If S = "DINNER" Then Debug.Print "Please come in." Debug.Print "Dinner will be ready soon." Elseif S = "SLEEP" Then Debug.Print "Sorry." Debug.Print "We are full for the night" End If End Sub

IIf Function

Syntax	<code>IIf(condexpr, TruePart, FalsePart)</code>
Group	Miscellaneous
Description	Return the value of the parameter indicated by <i>condexpr</i> . Both <i>TruePart</i> and <i>FalsePart</i> are evaluated.
<hr/>	
Parameter	Description
<i>condexpr</i>	If this value is True then return <i>TruePart</i> . Otherwise, return <i>FalsePart</i> .
<i>TruePart</i>	Return this value if <i>condexpr</i> is True .
<i>FalsePart</i>	Return this value if <i>condexpr</i> is False .
<hr/>	
See Also	If, Select Case, Choose().
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print IIf(1 > 0,"True","False") "True" End Sub</pre>

Imports Definition

Syntax	Imports [alias =] prefix
Group	Declaration
Description	The Imports statement provides a convenient shorthand for using identifiers provided by references.
<hr/>	
Parameter	Description
<i>alias</i>	Use this identifier as a substitute for "prefix". If omitted, no alternate identifier is defined.
<i>prefix</i>	Use this "prefix" before identifiers when searching references.
<hr/>	
Example	<pre>'#Language "WWB.NET" Imports System.Text Imports StrBuf = System.Text.StringBuffer Sub Main Dim sb As New StringBuffer ' System.Text.StringBuffer Dim sb2 As New StrBuf ' System.Text.StringBuffer End Sub</pre>

Input Instruction

Syntax	<code>Input'StreamNum, var[, ...]</code>
Group	File

Description Get input from *StreamNum* and assign it to *vars*. Input values are comma delimited. Leading and trailing spaces are ignored. If the first char (following the leading spaces) is a quote ("") then the string is terminated by an ending quote. Special values #NULL#, #FALSE#, #TRUE#, #date# and #ERROR number# are converted to their appropriate value and data type.

See Also [Line Input, Print, Write.](#)

Example

```
#Language "WWB.NET"
Sub Main
    Dim A, B, '
    FileOpen 1, "XXX", OpenMode.Input
    Input'1, A, B,
    Debug.Print A, B,
    FileClose'1
End Sub
```

InputBo' Function

Syntax InputBo'(*Promp*[, *Title*[, *Defaul*[, *XPos*, *YPos*]])

Group User Input

Description Display an input box where the user can enter a line of text. Pressing the OK button returns the string entered. Pressing the Cancel button returns a null string.

Parameter	Description
<i>Promp</i> '	Use this string value as the prompt in the input box.
<i>Title</i> '	Use this string value as the title of the input box. If this is omitted then the input box does not have a title.
<i>Defaul</i> '	Use this string value as the initial value in the input box. If this is omitted then the initial value is blank.
<i>XPos</i>	When the dialog is put up the left edge will be at this screen position. If this is omitted then the dialog will be centered.
<i>YPos</i>	When the dialog is put up the top edge will be at this screen position. If this is omitted then the dialog will be centered.

Example

```
#Language "WWB.NET"
Sub Main
    Dim '
    '= InputBo("Enter some text:", _
    "Input Box Example","asdf")
    Debug.Print '
End Sub
```

InputStrin' Function

Syntax InputStrin'(*StreamNum*, *N*)

Group File

Description Return *N* chars from *StreamNum*.

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
<i>N</i>	Read this many chars. If fewer than that many chars are left before the end of file then a run-time error occurs.

Example

```
'#Language "WWB.NET"
Sub Main
    FileOpen 1, "XXX", OpenMode.Input
    Dim L, T
    L = LOF(1)
    T = InputString(1,L)
    FileClose 1
    Debug.Print T;
End Sub
```

InStr Function

Syntax InStr([*Index*,]*S'*, *S*)**Group** String**Description** Return the index where *S'* first matches *S*. If no match is found return 0.

Parameter	Description
<i>Index</i>	Start searching for <i>S</i> at this index in <i>S'</i> . If this is omitted then start searching from the beginning of <i>S'</i> .
<i>S'</i>	Search for <i>S</i> in this string value. '
<i>S</i>	Search <i>S</i> for this string value. '

See Also [InStrRev\(\)](#), [Lef'\(\)](#), [Len\(\)](#), [Mi'\(\)](#), [Replac'\(\)](#), [Righ'\(\)](#).**Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print InStr("Hello", "l") ' 3
End Sub
```

InStrRev Function

Syntax InStrRev(*S*', *S*[, *Index*])**Group** String**Description** Return the index where *S'* last matches *S*. If no match is found return 0.

Parameter	Description
<i>S'</i>	Search for <i>S</i> in this string value. '
<i>S</i>	Search <i>S</i> for this string value. '
<i>Index</i>	Start searching for <i>S</i> ending at this index in <i>S'</i> . If this is omitted then start searching from the end of <i>S'</i> .

See Also [Lef'\(\)](#), [Len\(\)](#), [Mi'\(\)](#), [Replac'\(\)](#), [Righ'\(\)](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print InStrRev("Hello","l") ' 4
End Sub
```

Int Function

Syntax

`Int(Num)`

Group

Math

Description

Return the integer value.

Parameter	Description
<code>Num</code>	Return the largest integer which is less than or equal to this numeric value.'

See Also

Fix.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Int(9.9) ' 9
    Debug.Print Int(0) ' 0
    Debug.Print Int(-9.9) '-10
End Sub
```

Integer Data Type

Syntax

`Dim v As Integer`

Group

Data Type

Description

A 32 bit signed integer value.

Is Operator

Syntax

`expr Is expr`

-or-

`expr IsNot expr`

Group

Operator

Description

Return the **True** if both *exprs* refer to the same object.

The *IsNot* operator inverts the result.

See Also

Objects.

Example

```
'#Language "WWB.NET"
Sub Main
    Dim X As Object
    Dim Y As Object
```

```

Debug.Print X Is Y 'True
Debug.Print X IsNot Y 'False
End Sub

```

IsArray Function

Syntax `IsArray(expr)`

Group Variable Info

Description Return the **True** if *expr* is an array of values.

Parameter	Description
<i>expr</i>	A array variable or a variant var can contain multiple of values.

See Also [TypeName](#), [VarType](#).

Example

```

'#Language "WWB.NET"
Sub Main
    Dim X As Object, Y(2) As Integer
    Debug.Print IsArray(X) 'False
    X = Y
    Debug.Print IsArray(X) 'True
End Sub

```

IsDate Function

Syntax `IsDate(expr)`

Group Variable Info

Description Return the **True** if *expr* is a valid date.

Parameter	Description
<i>expr</i>	A variant expression to test for a valid date.

See Also [TypeName](#), [VarType](#).

Example

```

'#Language "WWB.NET"
Sub Main
    Dim X As Object
    X = 1
    Debug.Print IsDate(X) 'False
    X = Now
    Debug.Print IsDate(X) 'True
End Sub

```

IsDBNull Function

Syntax `IsDBNull(expr)`

Group Variable Info

Description Return the **True** if *expr* is System.DBNull.Value.

Parameter	Description
<i>expr</i>	A variant expression to test for System.DBNull.Value.

See Also **IsDBNull, TypeName, VarType.**

Example

```
'#Language "WWB.NET"
Sub Main
    Dim X As Object
    Debug.Print IsEmpty(X) 'True
    Debug.Print IsDBNull(X) 'False
    X = 1
    Debug.Print IsDBNull(X) 'False
    X = "1"
    Debug.Print IsDBNull(X) 'False
    X = System.DBNull.Value
    Debug.Print IsDBNull(X) True
    X = X*2
    Debug.Print IsDBNull(X) True
End Sub
```

IsError Function

Syntax IsError(*expr*)

Group Variable Info

Description Return the **True** if *expr* is an error code.

Parameter	Description
<i>expr</i>	A variant expression to test for an error code value.

See Also **TypeName, VarType.**

Example

```
'#Language "WWB.NET"
Sub Main
    Dim X As Object
    Debug.Print IsError(X) 'False
    X = CVErr(1)
    Debug.Print IsError(X) True
End Sub
```

IsNot Operator

Syntax *expr* IsNot *expr*

Group Operator

Description Return the **False** if both *exprs* refer to the same object.

See Also **Objects.**

Example

```
'#Language "WWB.NET"
Sub Main
    Dim X As Object
    Dim Y As Object
    Debug.Print X IsNot Y ' False
End Sub
```

IsNothing Function

Syntax

`IsNothing(expr)`

Group

Variable Info

Description

Return the **True** if *expr* contains an object reference which is **Nothing**.

Parameter	Description
<i>var</i>	If <i>objexpr</i> reference is Nothing return True .

See Also

TypeName, **VarType**.

Example

```
'#Language "WWB.NET"
Sub Main
    Dim X As Object
    X = 1
    Debug.Print IsNothing(X) 'False
    X = "1"
    Debug.Print IsNothing(X) 'False
    'X = Nothing
    Debug.Print IsNothing(X) 'True
End Sub
```

IsNumeric Function

Syntax

`IsNumeric(expr)`

Group

Variable Info

Description

Return the **True** if *expr* is a numeric value.

Parameter	Description
<i>expr</i>	A variant expression is a numeric value if it is <i>numeric</i> or string value that represents a number.

See Also

TypeName, **VarType**.

Example

```
'#Language "WWB.NET"
Sub Main
    Dim X As Object
    X = 1
    Debug.Print IsNumeric(X) 'True
    X = "1"
    Debug.Print IsNumeric(X) 'True
```

```
X = "A"
Debug.Print IsNumeric(X) 'False
End Sub
```

IsReference Function

Syntax `IsReference(expr)`

Group Variable Info

Description Return the **True** if *expr* contains an object reference.

Parameter	Description
<i>var</i>	If <i>expr</i> is an object reference return True .

See Also **TypeName**, **VarType**.

Example '#Language "WWB.NET"

```
Sub Main
    Dim X As Object
    X = 1
    Debug.Print IsReference(X) 'False
    X = "1"
    Debug.Print IsReference(X) 'False
    'X = Nothing
    Debug.Print IsReference(X) 'True
End Sub
```

Join Function

Syntax `Join(StrArray, [Sep])`

Group Miscellaneous

Description Return a string by concatenating all the values in the array with *Sep* in between each one.

Parameter	Description
<i>StrArray</i>	Concatenate values from this array.
<i>Sep</i>	Use this string value to separate the values. (Default: "")

See Also **Split()**.

Example '#Language "WWB.NET"

```
Sub Main
    Debug.Print Join(Array(1,2,3)) "1 2 3"
End Sub
```

KeyName Function

Syntax `KeyName(Key)`

Group	Miscellaneous
Description	Return the key name for a key number. This is the name used by SendKeys .
<hr/>	
Parameter	Description
<hr/>	
Key	Key number.
<hr/>	
See Also	SendKeys .
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print KeyName(&H270) "'^{F1}" End Sub</pre>

Kill Instruction

Syntax	Kill <i>Nam'</i>
Group	File
Description	Delete the file named by <i>Nam'</i> .
<hr/>	
Parameter	Description
<hr/>	
<i>Nam'</i>	This string value is the path and name of the file. A path relative to the current directory can be used.
<hr/>	
Example	<pre>'#Language "WWB.NET" Sub Main Kill "XXX" End Sub</pre>

LBound Function

Syntax	<code>LBound(<i>arrayvar</i>[, <i>dimension</i>])</code>
Group	Variable Info
Description	Return the lowest index.
<hr/>	
Parameter	Description
<hr/>	
<i>arrayvar</i>	Return the lowest index for this array variable.
<i>dimension</i>	Return the lowest index for this dimension of <i>arrayvar</i> . If this is omitted then return the lowest index for the first dimension.
<hr/>	
See Also	UBound().
Example	<pre>'#Language "WWB.NET" Sub Main Dim A(-1 To 3,2 To 6) Debug.Print LBound(A) '-1 Debug.Print LBound(A,1) '-1 Debug.Print LBound(A,2) '2 End Sub</pre>

LCas' Function

Syntax	LCas'()
Group	String
Description	Return a string from ' where all the uppercase letters have been lowercased.
<hr/>	
Parameter	Description
'	Return the string value of this after all chars have been converted to lowercase.'
<hr/>	
See Also	StrComp(), StrCon'(), UCas'().
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print LCas'("Hello") "hello" End Sub</pre>

Lef' Function

Syntax	Lef'('Len)
Group	String
Description	Return a string from ' with only the Len chars.
<hr/>	
Parameter	Description
'	Return the left portion of this string value.'
Len	Return this many chars. If ' is shorter than that then just return '.
<hr/>	
See Also	InStr(), InStrRev(), Len(), Mi'(), Replac'(), Righ'().
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print Lef'("Hello",2) "He" End Sub</pre>

Len Function

Syntax	Len()
	-or-
	Len(<i>usertypevar</i>)
<hr/>	
Group	String
Description	Return the number of characters in '.
<hr/>	
Parameter	Description
'	Return the number of chars in this string value.'
<i>usertypevar</i>	Return the number of bytes required to store this user variable. If the user type has any dynamic String and Variant elements the length returned may not be as big as the actual number of bytes required.
<hr/>	

See Also [InStr\(\)](#), [InStrRev\(\)](#), [Lef'\(\)](#), [Mi'\(\)](#), [Replac'\(\)](#), [Righ'\(\)](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Len("Hello") ' 5
End Sub
```

Like Operator

Syntax *str1* Like *str2*

Group Operator

Description Return the **True** if *str1* matches pattern *str2*. The pattern in *str2* is one or more of the special character sequences shown in the following table.

Char(s)	Description
?	Match any single character.
*	Match zero or more characters.
#	Match a single digit (0-9).
[charlist]	Match any char in the list.
[!charlist]	Match any char not in the list.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print "abcdfgcdefg" Like ""' False
    Debug.Print "abcdfgcdefg" Like "a*g" ' True
    Debug.Print "abcdfgcdefg" Like "a*cde*g" ' True
    Debug.Print "abcdfgcdefg" Like "a*cd*cd*g" ' True
    Debug.Print "abcdfgcdefg" Like "a*cd*cd*g" ' True
    Debug.Print "00aa" Like "####" ' False
    Debug.Print "00aa" Like "????" ' True
    Debug.Print "00aa" Like "###?" ' True
    Debug.Print "00aa" Like "*##*" ' True
    Debug.Print "hk" Like "hk**" ' True
End Sub
```

LineInput Function

Syntax LineInput(*StreamNum*)

Group File

Description Get a line of input from *StreamNum*.

See Also [Input](#), [Print](#), [Write](#).

Example

```
'#Language "WWB.NET"
Sub Main
    FileOpen 1, "XXX", OpenMode.Input
    Dim S As String
    S = LineInput(1)
```

```

Debug.Print S
FileClose 1
End Sub

```

ListBox Dialog Item Definition

Syntax `ListBox X, Y, DX, DY, StrArra'(), .Field[, Options]`

Group User Dialog

Description Define a listbox item.

Parameter	Description
X	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
Y	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
DX	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
DY	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
StrArra'()	This one-dimensional array of strings establishes the list of choices. All the non-null elements of the array are used.
Field	The value of the list box is accessed via this field. It is the index of the <i>StrArra'()</i> var.
Options	This numeric value controls the type of list box. Choose one value from following table. (If this numeric value omitted then zero is used.)

Option	Description
0	List is not sorted.
1	List is not sorted and horizontally scrollable.
2	List is sorted.
3	List is sorted and horizontally scrollable.

See Also [Begin Dialog](#), [MultiListBox](#).

Example `'#Language "WWB.NET"`

```

Sub Main
    Dim list'(3)
    list'(0) = "List 0"
    list'(1) = "List 1"
    list'(2) = "List 2"
    list'(3) = "List 3"
    Begin Dialog UserDialog 200,120
        Text 10,10,180,15,"Please push the OK button"
        ListBox 10,25,180,60,list'(),list
        OKButton 80,90,40,20
    End Dialog
    Dim dlg As UserDialog
    dlg.list = 2
    Dialog dlg ' show dialog (wait for ok)
    Debug.Print dlg.list
End Sub

```

Loc Function

Syntax	<code>Loc(StreamNum)</code>
Group	File
Description	Return <i>StreamNum</i> file position. For Random mode files this is the current record number minus one. For Binary mode files it is the current byte position minus one. Otherwise, it is the current byte position minus one divided by 128. The first position in the file is 0.

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.

Example

```
'#Language "WWB.NET"
Sub Main
    FileOpen 1, "XXX" OpenMode.Input
    L = Loc(1)
    FileClose'1
    Debug.Print L ' 0
End Sub
```

Lock Instruction

Syntax	<code>Lock StreamNum</code> -or- <code>Lock StreamNum, RecordNum</code> -or- <code>Lock StreamNum, [start] To end</code>
Group	File
Description	<p>Form 1: Lock all of <i>StreamNum</i>.</p> <p>Form 2: Lock a record (or byte) of <i>StreamNum</i>.</p> <p>Form 3: Lock a range of records (or bytes) of <i>StreamNum</i>. If <i>start</i> is omitted then lock starting at the first record (or byte).</p>

Note: Be sure to **Unlock** for each Lock instruction.

Note: For sequential files (Input, Output and Append) lock always affects the entire file.

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
<i>RecordNum</i>	For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1.
<i>start</i>	First record (or byte) in the range.
<i>end</i>	Last record (or byte) in the range.

See Also [Open, Unlock.](#)

Example

```
'#Language "WWB.NET"
Sub Main
    Dim V As Variant
    FileOpen 1, "SAVE_V.DAT" OpenMode.Binary
    Lock'1
    Get'1, 1, V
    V = "Hello"
    Put'1, 1, V
    Unlock'1
    FileClose'1
End Sub
```

LOF Function

Syntax `LOF(StreamNum)`

Group File

Description Return *StreamNum* file length (in bytes).

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.

Example

```
'#Language "WWB.NET"
Sub Main
    FileOpen 1, "XXX" OpenMode.Input
    L = LOF(1)
    FileClose'1
    Debug.Print L
End Sub
```

Log Function

Syntax `Log(Num)`

Group Math

Description Return the natural logarithm.

Parameter	Description
<i>Num</i>	Return the natural logarithm of this numeric value. The value e is approximately 2.718282.

See Also [Exp.](#)

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Log(1) ' 0
End Sub
```

Long Data Type

Syntax	<code>Dim v As Long</code>
Group	Data Type
Description	A 64 bit signed integer value. The number of bits is controlled by the #Language setting.

LSe' Function

Syntax	<code>LSe'(', Len)</code>
Group	String
Description	Return a string from ' with only the <i>Len</i> chars.
<hr/>	
Parameter	Description
'	Return the left portion of this string value.
<i>Len</i>	Return this many chars. If ' is shorter than that then fill the remainder with spaces.
<hr/>	
See Also	RSe'() .
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print LSe'("Hello",6) "Hello " End Sub</pre>

LTri' Function

Syntax	<code>LTri'()</code>
Group	String
Description	Return the string with 's leading spaces removed.
<hr/>	
Parameter	Description
'	Copy this string without the leading spaces.'
<hr/>	
See Also	RTri'(), Tri'() .
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print ".;" LTri'(" x ");"." ".x . End Sub</pre>

MacroCheck Function

Syntax	<code>MacroCheck(<i>MacroNam</i>)</code>
Group	Flow Control

Description Check the syntax of a *macro/module*. Does not execute the macro/module. Returns an Err object if there is a syntax error, otherwise Nothing is returned.

Parameter	Description
<i>MacroNam'</i>	Check the macro named by this string value.

See Also **MacroCheckThis, MacroRun, MacroRunThis, ModuleLoad, ModuleLoadThis.**

Example

```
'#Language "WWB.NET"
Sub Main
    Dim E As ErrObject
    'E = MacroCheck("Demo")
    If Not E Is Nothing Then Debug.Print E.Description
End Sub
```

MacroCheckThis Function

Syntax MacroCheckThis(*MacroCod'*)

Group Flow Control

Description Check the syntax the *macro/module* code in *MacroCode*. The macro/module code is not executed. Returns an Err object if there is a syntax error, otherwise Nothing is returned.

Parameter	Description
<i>MacroNam'</i>	Check the macro code in this string value.

See Also **MacroCheck, MacroRun, MacroRunThis, ModuleLoad, ModuleLoadThis.**

Example

```
'#Language "WWB.NET"
Sub Main
    Dim E As ErrObject
    'E = MacroCheckThis("bad macro")
    If Not E Is Nothing Then Debug.Print E.Description
End Sub
```

MacroDi' Function

Syntax MacroDi'

Group Flow Control

Description Return the directory of the current macro. A run-time error occurs if the current macro has never been saved.

See Also **MacroRun.**

Example

```
'#Language "WWB.NET"
Sub Main
```

```

' open the file called Data that is in the
' same directory as the macro
FileOpen 1, MacroDir & "\Data", OpenMode.Input
Dim S As String
S = LineInput(1)
Debug.Print '
FileClose'1
End Sub

```

MacroRun Instruction

Syntax	MacroRun <i>MacroNam</i> [, <i>Comman</i>]
Group	Flow Control
Description	Play a <i>macro</i> . Execution will continue at the following statement after the macro has completed.
<hr/>	
Parameter	Description
<i>MacroNam</i> '	Run the macro named by this string value.
<i>Comman</i> '	Pass this string value as the macro's Comman ' value.
<hr/>	
See Also	Comman ', MacroCheck , MacroCheckThis , MacroDi ', MacroRunThis , ModuleLoad , ModuleLoadThis .
Example	<pre> '#Language "WWB.NET" Sub Main Debug.Print "Before Demo" MacroRun "Demo" Debug.Print "After Demo" End Sub </pre>

MacroRunThis Instruction

Syntax	MacroRunThis <i>MacroCod</i> '
Group	Flow Control
Description	Play the <i>macro</i> code in <i>MacroCode</i> . Execution will continue at the following statement after the macro code has completed. The macro code can be either a single line or a complete macro.
<hr/>	
Parameter	Description
<i>MacroNam</i> '	Run the macro code in this string value.
<hr/>	
See Also	Comman ', MacroCheck , MacroCheckThis , MacroDi ', MacroRun , ModuleLoad , ModuleLoadThis .
Example	<pre> '#Language "WWB.NET" Sub Main Debug.Print "Before Demo" </pre>

```

MacroRunThis "MsgBox ""Hello"""
Debug.Print "After Demo"
End Sub

```

Main Sub

Syntax	<pre> Sub Main() ... End Sub -or- Private Sub Main() ... End Sub </pre>
Group	Declaration
Description	<p>Form 1: Each <i>macro</i> must define Sub Main. A macro is a "program". Running a macro starts the Sub Main and continues to execute until the subroutine finishes.</p> <p>Form 2: A code <i>module</i> may define a Private Sub Main. This Sub Main is the code module initialization subroutine. If Main is not defined then no special initialization occurs.</p>
See Also	Code Module.

Me Object

Syntax	Me
Group	Object
Description	Me references the current macro/module. It can be used like any other <i>object variable</i> , except that it's reference can't be changed.
Example	<pre> '#Language "WWB.NET" Sub Main Dolt Me.Dolt ' calls the same sub End Sub Sub Dolt MsgBox "Hello" End Sub </pre>

Mi' Function/Assignment

Syntax	<pre> Mi'(', Index[, Len]) -or- Mi'(strvar, Index[, Len]) = ' </pre>
Group	String

Description Function: Return the substring of ' starting at *Index* for *Len* chars.

Instruction: Assign ' to the substring in *strvar* starting at *Index* for *Len* chars.

Parameter	Description (Mid Function)
'	Copy chars from this string value.'
<i>Index</i>	Start copying chars starting at this index value. If the string is not that long then return a null string.
<i>Len</i>	Copy this many chars. If the ' does not have that many chars starting at <i>Index</i> then copy the remainder of '.
Parameter	Description (Mid Assignment)
<i>strvar</i>	Change part of this string.
<i>Index</i>	Change <i>strvar</i> starting at this index value. If the string is not that long then it is not changed.
<i>Len</i>	The number of chars copied is smallest of: the value of <i>Len</i> , the length of ' and the remaining length of <i>strvar</i> . (If this value is omitted then the number of chars copied is the smallest of: the length of ' and the remaining length of <i>strvar</i> .)
'	Copy chars from this string value.

See Also [InStr\(\)](#), [Lef'\(\)](#), [Len\(\)](#), [Replac'\(\)](#), [Righ'\(\)](#).

Example

```
'#Language "WWB.NET"
Sub Main
    ' = "Hello There"
    Mi'(,7) = "?????????"
    Debug.Print '"Hello ????'
    Debug.Print Mi'("Hello",2,1) "e"
End Sub
```

Minute Function

Syntax Minute(*dateexpr*)

Group Time/Date

Description Return the minute of the hour (0 to 59).

Parameter	Description
<i>dateexpr</i>	Return the minute of the hour for this date value.'

See Also [Hour\(\)](#), [Second\(\)](#), [Time\(\)](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Minute(#12:00:01 AM#) '0
End Sub
```

MkDir Instruction

Syntax MkDir *Nam'*

Group File

Description	Make directory <i>Nam'</i> .
Parameter	Description
<i>Nam'</i>	This string value is the path and name of the directory. A path relative to the current directory can be used.
See Also	RmDir.
Example	<pre>'#Language "WWB.NET" Sub Main MkDir "C:\WWTEMP" End Sub</pre>

ModuleLoad Function

Syntax	ModuleLoad(<i>ModuleNam'</i> , <i>CreateNew</i>)
Group	Flow Control
Description	Load a <i>module</i> . Does not execute the module. Macro's can not be loaded. Returns an object if successful, otherwise Nothing is returned.
Parameter	Description
<i>ModuleNam'</i>	Load the module named by this string value.
<i>CreateNew</i>	Return a new instance if True. Otherwise return the default instance. A class module does not have a default instance. A code module can not have a new instance.
See Also	MacroCheck, MacroCheckThis, MacroRun, MacroRunThis, ModuleLoadThis.
Example	<pre>'#Language "WWB.NET" Sub Main Dim Obj As Object 'Obj = ModuleLoad("Demo") Obj.Dolt ' call Demo's Dolt method End Sub</pre>

ModuleLoadThis Function

Syntax	ModuleLoadThis(<i>ModuleCod'</i> , <i>CreateNew</i>)
Group	Flow Control
Description	Load <i>ModuleCode</i> as a <i>module</i> . Does not execute the module. Macro's can not be loaded. Returns an object if successful, otherwise Nothing is returned.
Parameter	Description
<i>ModuleCod'</i>	Load the module code in this string value.
<i>CreateNew</i>	Return a new instance if True. Otherwise return the default instance. A class module does not have a default instance. A code module can not have a new instance.

See Also [MacroCheck](#), [MacroCheckThis](#), [MacroRun](#), [MacroRunThis](#), [ModuleLoad](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Dim Obj As Object
    'Obj = ModuleLoadThis("Sub Dolt" & vbCrLf & "End Sub", False)
    Obj.Dolt ' call Demo's Dolt method
End Sub
```

Month Function

Syntax Month(*dateexpr*)

Group Time/Date

Description Return the month of the year (1 to 12).

Parameter	Description
<i>dateexpr</i>	Return the month of the year for this date value.'

See Also [Date\(\)](#), [Day\(\)](#), [MonthName\(\)](#), [Weekday\(\)](#), [Year\(\)](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Month(#1/1/1900#) ' 1
    Debug.Print Month(#2/1/1900#) ' 2
End Sub
```

MonthName Function

Syntax MonthName(*NumZ{month}*[, *CondZ{abbrev}*])

Group Time/Date

Description Return the localized name of the month.

Parameter	Description
<i>month</i>	Return the localized name of this month. (1-12)
<i>abbrev</i>	If this conditional value is True then return the abbreviated form of the month name.

See Also [Month\(\)](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print MonthName(1) 'January
    Debug.Print MonthName(Month(Now))
End Sub
```

MsgBox Instruction/Function

Syntax	<code>MsgBox <i>Messag</i>[, <i>Type</i>][, <i>Titl</i>]</code> -or- <code>MsgBox(<i>Messag</i>[, <i>Type</i>][, <i>Titl</i>])</code>	
Group	User Input	
Description	Show a message box titled <i>Titl</i> . <i>Type</i> controls what the message box looks like (choose one value from each category). Use <code>MsgBox()</code> if you need to know what button was pressed. The result indicates which button was pressed.	
<hr/>		
Result	Value	Button Pressed
vbOK	1	OK button
vbCancel	2	Cancel button
vbAbort	3	Abort button
vbRetry	4	Retry button
vbIgnore	5	Ignore button
vbYes	6	Yes button
vbNo	7	No button
<hr/>		
Parameter	Description	
<i>Messag</i> '	This string value is the text that is shown in the message box.	
<i>Type</i>	This numeric value controls the type of message box. Choose one value from each of the following tables.	
<i>Titl</i> '	This string value is the title of the message box.	
<hr/>		
Button	Value	Effect
vbOkOnly	0	OK button
vbOkCancel	1	OK and Cancel buttons
vbAbortRetryIgnore	2	Abort, Retry, Ignore buttons
vbYesNoCancel	3	Yes, No, Cancel buttons
vbYesNo	4	Yes and No buttons
vbRetryCancel	5	Retry and Cancel buttons
<hr/>		
Icon	Value	Effect
	0	No icon
vbCritical	16	Stop icon
vbQuestion	32	Question icon
vbExclamation	48	Attention icon
vbInformation	64	Information icon
<hr/>		
Default	Value	Effect
vbDefaultButton1	0	First button
vbDefaultButton2	256	Second button
vbDefaultButton3	512	Third button
<hr/>		
Mode	Value	Effect
vbApplicationModal	0	Application modal
vbSystemModal	4096	System modal
vbMsgBoxSetForeground	&h10000	Show message box in front of all other windows

Example

```
'#Language "WWB.NET"
Sub Main
    MsgBox "Please press OK button"
    If MsgBox("Please press OK button",vbOkCancel) = vbOK Then
        Debug.Print "OK was pressed"
    Else
        Debug.Print "Cancel was pressed"
    End If
End Sub
```

MultiListBox Dialog Item Definition

Syntax `MultiListBox X, Y, DX, DY, StrArra'(), .Field[, Options]`

Group User Dialog

Description Define a multiple selection listbox item.

Parameter	Description
X	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
Y	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
DX	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
DY	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
StrArra'()	This one-dimensional array of strings establishes the list of choices. All the non-null elements of the array are used.
Field	The values of the list box are accessed via this field. It is the index of the <code>StrArra'()</code> var.
Options	This numeric value controls the type of list box. Choose one value from following table. (If this numeric value omitted then zero is used.)

Option	Description
0	List is not sorted.
1	List is not sorted and horizontally scrollable.
2	List is sorted.
3	List is sorted and horizontally scrollable.

See Also [Begin Dialog, ListBox.](#)

Example

```
'#Language "WWB.NET"
Sub Main
    Dim list'(3)
    list'(0) = "List 0"
    list'(1) = "List 1"
    list'(2) = "List 2"
    list'(3) = "List 3"
    Begin Dialog UserDialog 200,120
        Text 10,10,180,15,"Please push the OK button"
        MultiListBox 10,25,180,60,list'().list
        OKButton 80,90,40,20
```

```

End Dialog
Dim dlg As UserDialog
Dim selects() As Integer = {0,2}
dlg.list = selects
Dialog dlg ' show dialog (wait for ok)
Dim i As Integer
For i = LBound(dlg.list) To UBound(dlg.list)
    Debug.Print dlg.list(i);
Next i
Debug.Print
End Sub

```

New Operator

Syntax	New <i>objtype</i> [(<i>param</i> [, ...])] -or- New <i>objtype</i> () { <i>expr</i> [, ...] \verb.} ,						
Group	Operator						
Description	Returns a new instance of <i>objtype</i> . Or, use { <i>expr</i> , ... } to create an array value.						
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">Parameter</th><th style="text-align: left; padding: 2px;">Description</th></tr> </thead> <tbody> <tr> <td style="padding: 2px;"><i>objtype</i></td><td style="padding: 2px;">This is the new object's type.</td></tr> <tr> <td style="padding: 2px;"><i>params</i></td><td style="padding: 2px;">A list of zero or more <i>params</i> used during the object creation.</td></tr> </tbody> </table>	Parameter	Description	<i>objtype</i>	This is the new object's type.	<i>params</i>	A list of zero or more <i>params</i> used during the object creation.
Parameter	Description						
<i>objtype</i>	This is the new object's type.						
<i>params</i>	A list of zero or more <i>params</i> used during the object creation.						
See Also	Objects.						
Example	<pre>'#Language "WWB.NET" Sub Main Dim obj As Object 'obj = New System.Collections.Hashtable End Sub</pre>						

Nothing Keyword

Group	Constant
Description	An <i>objexpr</i> that does not refer to any object.

Now Function

Syntax	Now
Group	Time/Date
Description	Return the current date and time as a Date value.
See Also	Date, Time.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Now ' example: 1/1/1995 10:05:32 AM
End Sub
```

Object Data Type

Syntax

```
Dim v As Object
```

Group

Data Type

Description

An object reference value. (see **Objects**) An object reference may also appear to have a data value, see table below:

Data	Description
get-reference	Use in any <i>object expression</i> .
set-reference	Use Assign to change the reference.
get-value	Use the reference as a data value. If it is not a data value, a "type mismatch error" will occur.
assign-value	Use Assign to change the value.

Object Module

Group

Declaration

Description

An object *module* implements an object.

- Has a set of **Public procedures** accessible from other *macros* and *modules*.
- These public symbols are accessed via the name of the object module or an object variable.
- Public **Consts**, **Structures**, arrays, fixed length strings are not allowed.
- Has an optional Private Sub **Object_Initialize** which is called when an instance is created.
- Has an optional Private Sub **Object_Terminate** which is called when an instance is destroyed.
- An object module is similar to a **Class module** except that one instance is automatically created. That instance has the same name as the object module's name.
- To create additional instances use:

```
Dim Obj As objectname
Obj = New objectname
```

See Also

Class Module, **Code Module**, **Uses**, **Object_Initialize**, **Object_Terminate**.

Example

```
'A.WWB
'#Language "WWB.NET"
'#Uses "System.OMB"
```

```

Sub Main
    Debug.Print Hex(System.Version)
End Sub

'System.ObM
'File|New Module|Object Module
>Edit|Properties|Name=System
'#Language "WWB.NET"
Option Explicit
Declare Function GetVersion16 Lib "Kernel" _
    Alias "GetVersion" () As Long
Declare Function GetVersion Lib "Kernel32" () As PortInt

Public Function Version() As PortInt
    If Win16 Then
        Version = GetVersion16
    Else
        Version = GetVersion
    End If
End Function

```

Object_Initialize Sub

Syntax	Private Sub Object_Initialize() ...
Group	Declaration
Description	Object module initialization subroutine. Each time a new instance is created for a Object module the Object_Initialize sub is called. If Object_Initialize is not defined then no special initialization occurs.
	Note: Object_Initialize is also called for the instance that is automatically created.
See Also	Object Module, Object_Terminate.

Object_Terminate Sub

Syntax	Private Sub Object_Terminate() ...
Group	Declaration
Description	Object module termination subroutine. Each time an instance is destroyed for a Object module the Object_Terminate sub is called. If Object_Terminate is not defined then no special termination occurs.
See Also	Object Module, Object_Initialize.

Oc' Function

Syntax	Oc'(Num)				
Group	String				
Description	Return a octal string.				
<hr/>					
<table border="1"> <thead> <tr> <th>Parameter</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Num</td><td>Return an octal encoded string for this numeric value.</td></tr> </tbody> </table>		Parameter	Description	Num	Return an octal encoded string for this numeric value.
Parameter	Description				
Num	Return an octal encoded string for this numeric value.				
See Also	He'() , St'() , Val() .				
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print Oc'(15) '17 End Sub</pre>				

OKButton Dialog Item Definition

Syntax	OKButton X, Y, DX, DY[, .Field]												
Group	User Dialog												
Description	Define an OK button item. Pressing the OK button updates the <i>dlgvar</i> field values and closes the dialog. (Dialog() function call returns -1.)												
<hr/>													
<table border="1"> <thead> <tr> <th>Parameter</th><th>Description</th></tr> </thead> <tbody> <tr> <td>X</td><td>This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.</td></tr> <tr> <td>Y</td><td>This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.</td></tr> <tr> <td>DX</td><td>This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.</td></tr> <tr> <td>DY</td><td>This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.</td></tr> <tr> <td>Field</td><td>This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i>. If this is omitted then the field name is "OK".</td></tr> </tbody> </table>		Parameter	Description	X	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.	Y	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.	DX	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.	DY	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.	Field	This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i> . If this is omitted then the field name is "OK".
Parameter	Description												
X	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.												
Y	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.												
DX	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.												
DY	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.												
Field	This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i> . If this is omitted then the field name is "OK".												
See Also	Begin Dialog.												
Example	<pre>'#Language "WWB.NET" Sub Main Begin Dialog UserDialog 200,120 Text 10,10,180,30,"Please push the OK button" OKButton 80,90,40,20 End Dialog Dim dlg As UserDialog Dialog dlg ' show dialog (wait for ok) End Sub</pre>												

On Error Instruction

Syntax	On Error GoTo 0 -or- On Error GoTo <i>label</i> -or- On Error Resume Next
Group	Error Handling
Description	<p>Form 1: Disable the error handler (default).</p> <p>Form 2: Send error conditions to an error handler.</p> <p>Form 3: Error conditions continue execution at the next statement.</p> <p>On Error sets or disables the error handler. Each user defined <i>procedure</i> has its own error handler. The default is to terminate the <i>macro</i> on any error. The Err object's properties are set whenever an error occurs. Once an error has occurred and the error handler is executing any further errors will terminate the macro, unless the Err object has been cleared.</p>
	<p>Note: This instruction clears the Err and sets Error`\$ to null.</p>
Example	<pre>#Language "WWB.NET" Sub Main On Error Resume Next Err.Raise 1 Debug.Print "RESUMING, Err="; Err On Error GoTo X Err.Raise 1 Exit Sub X: Debug.Print "Err="; Err Err.Clear Debug.Print "Err="; Err Resume Next End Sub</pre>

Operators

Syntax	$^$ Not * / \ Mod + - << >> & < <= > >= = <> Is IsNot And AndAlso Or OrElse Xor
Description	<p>These operators are available for numbers n1 and n2 or strings s1 and s2. If any value in an expression is System.DBNull.Value then the expression's value is System.DBNull.Value. The order of operator evaluation is controlled by operator precedence.</p>
Operator	Description
- n1	Negate n1.

$n1 \wedge n2$	Raise $n1$ to the power of $n2$.
$n1 * n2$	Multiply $n1$ by $n2$.
$n1 / n2$	Divide $n1$ by $n2$.
$n1 \backslash n2$	Divide the integer value of $n1$ by the integer value of $n2$.
$n1 \text{ Mod } n2$	Remainder of the integer value of $n1$ after dividing by the integer value of $n2$.
$n1 + n2$	Add $n1$ to $n2$.
$s1 + s2$	Concatenate $s1$ with $s2$.
$n1 - n2$	Difference of $n1$ and $n2$.
$n1 << n2$	Shift $n1$ by $n2$ bits to the left. The number of bits to shift is indicated by the lowest bits of $n2$.
$n1 >> n2$	Shift $n1$ by $n2$ bits to the right. The number of bits to shift is indicated by the lowest bits of $n2$. (For signed numbers the sign bit is propagated to the right.)
$s1 \& s2$	Concatenate $s1$ with $s2$.
$n1 < n2$	Return True if $n1$ is less than $n2$.
$n1 \leq n2$	Return True if $n1$ is less than or equal to $n2$.
$n1 > n2$	Return True if $n1$ is greater than $n2$.
$n1 \geq n2$	Return True if $n1$ is greater than or equal to $n2$.
$n1 = n2$	Return True if $n1$ is equal to $n2$.
$n1 \neq n2$	Return True if $n1$ is not equal to $n2$.
$s1 < s2$	Return True if $s1$ is less than $s2$.
$s1 \leq s2$	Return True if $s1$ is less than or equal to $s2$.
$s1 > s2$	Return True if $s1$ is greater than $s2$.
$s1 \geq s2$	Return True if $s1$ is greater than or equal to $s2$.
$s1 = s2$	Return True if $s1$ is equal to $s2$.
$s1 \neq s2$	Return True if $s1$ is not equal to $s2$.
$\text{Not } n1$	Bitwise invert the integer value of $n1$. Only Not True is False .
$n1 \text{ And } n2$	Bitwise and the integer value of $n1$ with the integer value $n2$.
$n1 \text{ AndAlso } n2$	Logical and of $n1$ with the $n2$. If $n1$ is False , $n2$ is not evaluated.
$n1 \text{ Or } n2$	Bitwise or the integer value of $n1$ with the integer value $n2$.
$n1 \text{OrElse } n2$	Logical or of $n1$ with the $n2$. If $n1$ is True , $n2$ is not evaluated.
$n1 \text{ Xor } n2$	Bitwise exclusive-or the integer value of $n1$ with the integer value $n2$.

Example

```
#Language "WWB.NET"
Sub Main
    N1 = 10
    N2 = 3
    S' = "asdfg"
    S' = "hjkl"
    Debug.Print -N1      '-10
    Debug.Print N1 ^ N2  '1000
    Debug.Print Not N1   '-11
    Debug.Print N1 * N2  '30
    Debug.Print N1 / N2  '3.33333333333333
    Debug.Print N1 \ N2  '3
    Debug.Print N1 Mod N2 '1
    Debug.Print N1 + N2  '13
    Debug.Print S' + S'  "asdfghjkl"
    Debug.Print N1 - N2  '7
    Debug.Print N1 << N2  '80
    Debug.Print N1 >> N2  '1
    Debug.Print N1 & N2  '"103"
```

```

Debug.Print N1 < N2  'False
Debug.Print N1 <= N2 'False
Debug.Print N1 > N2  'True
Debug.Print N1 >= N2 'True
Debug.Print N1 = N2  'False
Debug.Print N1 <> N2 'True
Debug.Print S' < S' 'True
Debug.Print S' <= S' 'True
Debug.Print S' > S' 'False
Debug.Print S' >= S' 'False
Debug.Print S' = S' 'False
Debug.Print S' <> S' 'True
Debug.Print N1 And N2  '2
Debug.Print N1 AndAlso N2 'True
Debug.Print N1 Or N2   '11
Debug.Print N1OrElse N2 'True
Debug.Print N1 Xor N2  '9
End Sub

```

Option Definition

Syntax	Option Compare [Binary Text] -or- Option Explicit [On Off] -or- Option Strict [On Off] -or- Option Private Module
Group	Declaration
Description	<p>Option Compare: Set the default comparison mode for <, <=, =, >, >=, <>, Like and StrComp.</p> <ul style="list-style-type: none"> • Option Compare Binary - compare string text using binary data (default) • Option Compare Text - compare string text using the collation rules <p>Option Explicit On: Require all variables to be declared prior to use. Variables are declared using Dim, Private, Public, Static or as a parameter of Sub, Function or Property blocks. (This is the default.)</p> <p>Option Strict: Ignored.</p> <p>Option Private: Public symbols defined by the module are only accessible from the same project.</p>
Example	<pre>'#Language "WWB.NET" Option Explicit Off Sub Main A = 1 ' don't need to declare End Sub</pre>

OptionButton Dialog Item Definition

Syntax OptionButton *X, Y, DX, DY, Titl[, .Field]*

Group User Dialog

Description Define an option button item.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Titl</i>	The value of this string is the title of the option button.

See Also [Begin Dialog, OptionGroup.](#)

Example '#Language "WWB.NET"

```
Sub Main
    Begin Dialog UserDialog 200,120
        Text 10,10,180,15,"Please push the OK button"
        OptionGroup .options
            OptionButton 10,30,180,15,"Option &0"
            OptionButton 10,45,180,15,"Option &1"
            OptionButton 10,60,180,15,"Option &2"
        OKButton 80,90,40,20
    End Dialog
    Dim dlg As UserDialog
    dlg.options = 2
    Dialog dlg ' show dialog (wait for ok)
    Debug.Print dlg.options
End Sub
```

OptionGroup Dialog Item Definition

Syntax OptionGroup *.Field*

OptionButton *X, Y, DX, DY, Titl[, .Field]*

OptionButton *X, Y, DX, DY, Titl[, .Field]*

...

Group User Dialog

Description Define a optiongroup and option button items.

Parameter	Description
<i>Field</i>	The value of the option group is accessed via this field. This first option button is 0, the second is 1, etc.
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.

<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Title</i>	The value of this string is the title of the option button.

See Also**Begin Dialog, OptionButton.****Example**

```
#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120
        Text 10,10,180,15,"Please push the OK button"
        OptionGroup .options
            OptionButton 10,30,180,15,"Option &0"
            OptionButton 10,45,180,15,"Option &1"
            OptionButton 10,60,180,15,"Option &2"
        OKButton 80,90,40,20
    End Dialog
    Dim dlg As UserDialog
    dlg.options = 2
    Dialog dlg ' show dialog (wait for ok)
    Debug.Print dlg.options
End Sub
```

Picture Dialog Item Definition

SyntaxPicture *X, Y, DX, DY, FileName', Type[, .Field]***Group**

User Dialog

Description

Define a picture item. The bitmap is automatically sized to fit the item's entire area.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>FileName'</i>	The value of this string is the .BMP file shown in the picture control.
<i>Type</i>	This numeric value indicates the type of bitmap used. See below.
<i>Field</i>	This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i> . If this identifier is omitted then the first two words of the title are used.

Type	Effect
0	<i>FileName</i> is the name of the bitmap file. If the file does not exist then "(missing picture)" is displayed.
3	The clipboard's bitmap is displayed. Not supported.
+16	Instead of displaying "(missing picture)" a run-time error occurs.

See Also [Begin Dialog.](#)**Example**

```
'#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120
        Picture 10,10,180,75,"SAMPLE.BMP",0
        OKButton 80,90,40,20
    End Dialog
    Dim dlg As UserDialog
    Dialog dlg ' show dialog (wait for ok)
End Sub
```

Print Instruction

Syntax `Print StreamNum, [expr[, ...]]`**Group** File

Description Print the *expr(s)* to *StreamNum*. A *num* is it automatically converted to a string before printing (just like **St'()**).

See Also [Input](#), [LineInput](#), [PrintLine](#), [Write](#), [WriteLine](#).**Example**

```
'#Language "WWB.NET"
Sub Main
    Dim A, B, '
    A = 1
    B = 2
    '= "Hello"
    FileOpen 1, "XXX", OpenMode.Output
    Print'1, A, "", B, """", '
    FileClose'1
End Sub
```

PrintLine Instruction

Syntax `PrintLine StreamNum, [expr[, ...]]`**Group** File

Description Print the *expr(s)* to *StreamNum*. A *num* is it automatically converted to a string before printing (just like **St'()**). A newline is printed at the end.

See Also [Input](#), [LineInput](#), [Print](#), [Write](#), [WriteLine](#).**Example**

```
'#Language "WWB.NET"
Sub Main
    Dim A, B, C
    A = 1
    B = 2
    C = "Hello"
    FileOpen 1, "XXX", OpenMode.Output
    PrintLine 1, A, "", B, """", C, """"
```

```
FileClose 1
End Sub
```

Private Definition

Syntax	Private [WithEvents] <i>vardeclaration</i> [= <i>initialvalue</i>] [, ...]
Group	Declaration
Description	Create arrays (or simple variables) which are available to the entire <i>macro/module</i> , but not other macros/modules. Dimension var array(s) using the <i>dimensions</i> to establish the minimum and maximum index value for each dimension. If the <i>dimensions</i> are omitted then a scalar (single value) variable is defined. A dynamic array is declared using () without any <i>dimensions</i> . It must be ReDimensioned before it can be used. The Private statement must be placed outside of Sub , Function or Property blocks.
See Also	Dim, Option Base, Public, ReDim, Static, WithEvents.
Example	<pre>'#Language "WWB.NET" Private A0, A1(1), A2(1,1) Sub Init A0 = 1 A1(0) = 2 A2(0,0) = 3 End Sub Sub Main Init Debug.Print A0; A1(0); A2(0,0) ' 1 2 3 End Sub</pre>

Private Keyword

Group	Declaration
Description	Private Consts, Declares, Functions, Propertys, Subs and Structures are only available in the current <i>macro/module</i> .

Property Definition

Syntax	<pre>[Private Public Friend]_ [Default] [ReadOnly WriteOnly]_ Property <i>name</i>[<i>type</i>]([<i>param</i>]) [As <i>type</i>[()]] [Get <i>statements</i> End Get] [Set(<i>param</i>) <i>statements</i> End Set] End Property</pre>
Group	Declaration
Description	<p>User defined property. The property defines a set of <i>statements</i> to be executed when its value is used or changed. A property acts like a variable, except that getting its value calls Property's Get block and changing its value calls Property's Set block. The values of the calling <i>arglist</i> are assigned to the <i>params</i>.</p> <ul style="list-style-type: none"> • At a minimum, either a Get or Set block must be specified. • If only the Get block is specified the property must be marked as ReadOnly. • If only the Set block is specified the property must be marked as WriteOnly. • If both the Get and Set blocks are specified the property must not be marked as ReadOnly or WriteOnly. • The Set block's parameter type must match the Property's return type.
Access	If no access is specified then Public is assumed.
See Also	Function, Sub.
Example	<pre>'#Language "WWB.NET" Dim X_Value As String Property X() As String Get X = X_Value End Get Set(ByVal Value As String) X_Value = Value End Set End Property Sub Main X = "Hello" Debug.Print X "Hello" End Sub</pre>

Public Definition

Syntax	Public [WithEvents] <i>vardeclaration</i> [= <i>initialvalue</i>], ...]
Group	Declaration

Description Create arrays (or simple variables) which are available to the entire *macro/module* and other macros/modules. Dimension var array(s) using the *dimensions* to establish the minimum and maximum index value for each dimension. If the *dimensions* are omitted then a scalar (single value) variable is defined. A dynamic array is declared using () without any *dimensions*. It must be **ReDimensioned** before it can be used. The Public statement must be placed outside of **Sub**, **Function** or **Property** blocks.

See Also **Dim, Option Base, Private, ReDim, Static, WithEvents.**

Example

```
'#Language "WWB.NET"
Public A0, A1(1), A2(1,1)
```

```
Sub Init
    A0 = 1
    A1(0) = 2
    A2(0,0) = 3
End Sub

Sub Main
    Init
    Debug.Print A0; A1(0); A2(0,0) ' 1 2 3
End Sub
```

Public Keyword

Group Declaration

Description **Public Consts, Declares, Functions, Propertys, Subs and Structures** in a *module* are available in all other *macros/modules* that access it.

PushButton Dialog Item Definition

Syntax PushButton *X, Y, DX, DY, Titl[, Field]*

Group User Dialog

Description Define a push button item. Pressing the push button updates the *dlgvar* field values and closes the dialog. (**Dialog()** function call returns the push button's ordinal number in the dialog. The first push button returns 1.)

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12ths of the character height for the dialog's font.
<i>Titl</i>	The value of this string is the title of the push button control.

<i>Field</i>	This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i> . If this identifier is omitted then the first two words of the title are used.
--------------	---

See Also**Begin Dialog.****Example**

```
#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120
        Text 10,10,180,30,"Please push the Dolt button"
        OKButton 40,90,40,20
        PushButton 110,90,60,20,"&Do It"
    End Dialog
    Dim dlg As UserDialog
    Debug.Print Dialog(dlg)
End Sub
```

Put Instruction

SyntaxPut' *StreamNum*, [*RecordNum*], *var***Group**

File

DescriptionWrite a variable's value to *StreamNum*.

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
<i>RecordNum</i>	For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1. If this is omitted then the current position (or record number) is used.
<i>var</i>	This variable value is written to the file. For a fixed length variable (like Long) the number of bytes required to store the variable are written. For a Variant variable two bytes which describe its type are written and then the variable value is written accordingly. For a <i>user structure</i> variable each field is written in sequence. For an array variable each element is written in sequence. For a dynamic array variable the number of dimensions and range of each dimension is written prior to writing the array values. All binary data values are written to the file in <i>little-endian</i> format.
Note: When writing a string (or a dynamic array) to a Binary mode file the string length (or array dimension) information is not written. Only the string data or array elements are written.	

See Also**Get, Open.****Example**

```
#Language "WWB.NET"
Sub Main
    Dim V As Variant
    FileOpen 1, "SAVE_V.DAT", OpenMode.Binary, OpenAccess.Access
    Put'1, , V
    FileClose'1
End Sub
```

QBColor Function

Syntax QBColor(*num*)

Group Miscellaneous

Description Return the appropriate color defined by Quick Basic.

num	color
0	black
1	blue
2	green
3	cyan
4	red
5	magenta
6	yellow
7	white
8	gray
9	light blue
10	light green
11	light cyan
12	light red
13	light magenta
14	light yellow
15	bright white

See Also [RGB\(\)](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Hex(QBColor(1)) "800000"
    Debug.Print Hex(QBColor(7)) "C0C0C0"
    Debug.Print Hex(QBColor(8)) "808080"
    Debug.Print Hex(QBColor(9)) "FF0000"
    Debug.Print Hex(QBColor(10)) "FF00"
    Debug.Print Hex(QBColor(12)) "FF"
    Debug.Print Hex(QBColor(15)) "FFFFFF"
End Sub
```

RaiseEvent Instruction

Syntax RaiseEvent *name*[(*arglist*)]

Group Flow Control

Description Raise an **Event**. Evaluate the *arglist* and call *name* with those values.

See Also [Event, WithEvents](#).

Example

```
'Class1.cls
'#Language "WWB.NET"
```

```

Event Changing(ByVal OldValue As String, ByVal newValue As String)
Private Value_ As String

Property Get Value As String
    Value = Value_
End Property

Property Let Value(ByVal newValue As String)
    RaiseEvent Changing(Value_, newValue)
    Value_ = newValue
End Property

'Macro.bas
'#Uses "Class1.cls"
'#Language "WWB.NET"

Dim WithEvents c1 As Class1

Sub Main
    'c1 = New Class1
    c1.Value = "Hello"
    c1.Value = "Goodbye"
End Sub

Sub c1_Changing(ByVal OldValue As String, ByVal newValue As String) Handles c1.Changing
    Debug.Print "OldValue=""" & OldValue & """", newValue=""" & newValue & """
End Sub

```

Randomize Instruction

Syntax	Randomize [<i>Seed</i>]
Group	Math
Description	Randomize the random number generator.
<hr/>	
Parameter	Description
<i>Seed</i>	This numeric value sets the initial seed for the random number generator. If this value is omitted then the current time is used as the seed.
<hr/>	
See Also	Rnd().
Example	<pre> '#Language "WWB.NET" Sub Main Randomize Debug.Print Rnd ' 0.??????????????? End Sub </pre>

ReDim Instruction

Syntax	ReDim [Preserve] <i>vardeclaration</i> [<i>As type</i>][, ...]
---------------	--

Group	Declaration
Description	Redimension a dynamic <i>arrayvar</i> or <i>user defined structure</i> array element. Use <i>Preserve</i> to keep the array values. Otherwise, the array values will all be reset. When using <i>preserve</i> only the last index of the array may change, but the number of indexes may not. (A one-dimensional array can't be redimensioned as a two-dimensional array.)
See Also	Dim, Option Base, Private, Public, Static.
Example	'#Language "WWB.NET" Sub Main Dim X() ReDim X(3) Debug.Print UBound(X) ' 3 ReDim X(200) Debug.Print UBound(X) ' 200 End Sub

Rem Instruction

Syntax	Rem ... -or- '...
Group	Miscellaneous
Description	Both forms are comments. The Rem form is an instruction. The ' form can be used at the end of any line. All text from either ' or Rem to the end of the line is part of the comment. That text is not executed.
Example	'#Language "WWB.NET" Sub Main Debug.Print "Hello" ' prints to the output window Rem the macro terminates at Main's End Sub End Sub

RemoveHandler Instruction

Syntax	RemoveHandler <i>expr.name, delegate</i>
Group	Flow Control
Description	Remove a <i>delegate</i> from the <i>expr.name</i> 's list of handlers.

Parameter	Description
<i>expr</i>	This is an expression which returns an object reference.
<i>name</i>	This is an event name.
<i>delegate</i>	This is an expression which returns a delegate.

See Also	AddHandler.
-----------------	--------------------

Example

```

'#Reference #System.Windows.Forms, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089, processorArchitecture=MSIL
'#Language "WWB.NET"
Imports System.Windows.Forms

Sub Main
    Using t As New Timer
        AddHandler t.Tick, AddressOf OnTick
        t.Interval = 1000
        t.Enabled = True
        Wait 5
        RemoveHandler t.Tick, AddressOf OnTick
    End Using
End Sub

Private Sub OnTick(ByVal sender As Object, ByVal e As System.EventArgs)
    Debug.Print Now
End Sub

```

Rename Instruction

SyntaxRename *OldNam'*, *NewNam'***Group**

File

DescriptionRename file *OldNam'* as *NewNam'*.

Parameter	Description
<i>OldNam'</i>	This string value is the path and name of the file. A path relative to the current directory can be used.
<i>NewNam'</i>	This is the new file name (and path). A path relative to the current directory can be used.

Example

```

'#Language "WWB.NET"
Sub Main
    Rename "AUTOEXEC.BAK", "AUTOEXEC.SAV"
End Sub

```

Replac' Function

SyntaxReplac'(' *Pa'*, *Re'*, [*Index*], [*Count*])**Group**

String

DescriptionReplace *Pa'* with *Re'* in '.

Parameter	Description
'	This string value is searched. Replacements are made in the string returned by Replace.
<i>Pa'</i>	This string value is the pattern to look for.
<i>Re'</i>	This string value is the replacement.
<i>Index</i>	This numeric value is the starting index in '. Replace(S,Pat,Rep,N) is equivalent to Replace(Mid(S,N),Pat,Rep). If this is omitted use 1.

Count This numeric value is the maximum number of replacements that will be done. If this is omitted use -1 (which means replace all occurrences).

See Also**InStr(), InStrRev(), Lef'(), Len(), Mi'(), Righ'().****Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Replac("abcabc","b","B")  "aBcaBc"
    Debug.Print Replac("abcabc","b","B",,1) "aBcabc"
    Debug.Print Replac("abcabc","b","B",3) "caBc"
    Debug.Print Replac("abcabc","b","B",9) ""
End Sub
```

Reset Instruction

Syntax Reset**Group** File**Description** Close all open streams for the current *macro/module*.**See Also** **Close, Open.****Example** '#Language "WWB.NET"

```
Sub Main
    ' read the first line of XXX and print it
    FileOpen 1, "XXX", OpenMode.Input
    Dim L As String
    L = LineInput(1)
    Debug.Print '
    Reset
End Sub
```

Resume Instruction

Syntax Resume *label*

-or-

Resume Next

Group Error Handling**Description** Form 1: Resume execution at *label*.

Form 2: Resume execution at the next statement.

Once an error has occurred, the error handler can use Resume to continue execution. The error handler must use Resume or **Exit** at the end.

Note: This instruction clears the **Err** and sets **Error`\$** to null.

Example

#Language "WWB.NET"

Sub Main

```

On Error GoTo X
Err.Raise 1
Debug.Print "RESUMING"
Exit Sub

X: Debug.Print "Err="; Err
Resume Next
End Sub

```

Return Instruction

Syntax	Return <i>expr</i>
Group	Flow Control
Description	The return instruction causes the Function block to exit with the value of the <i>expr</i> .
Example	<pre> '#Language "WWB.NET" Sub Main Debug.Print Func(2) ' 6 End Sub Function Func(N) Return N*3 End Function </pre>

RGB Function

Syntax	RGB(<i>red</i> , <i>green</i> , <i>blue</i>)
Group	Miscellaneous
Description	Return a color. Some useful color constants are predefined:
	<ul style="list-style-type: none"> • vbBlack - same as RGB(0,0,0) • vbRed - same as RGB(255,0,0) • vbGreen - same as RGB(0,255,0) • vbYellow - same as RGB(255,255,0) • vbBlue - same as RGB(0,0,255) • vbMagenta - same as RGB(255,0,255) • vbCyan - same as RGB(0,255,255) • vbWhite - same as RGB(255,255,255)
See Also	QBColor().
Example	<pre> '#Language "WWB.NET" Sub Main Debug.Print Hex(RGB(255,0,0)) "FF0000" End Sub </pre>

Righ' Function

Syntax Righ'('; *Len*)

Group String

Description Return the last *Len* chars of '.

Parameter	Description
'	Return the right portion of this string value.'
<i>Len</i>	Return this many chars. If ' is shorter than that then just return '.

See Also **InStr(), InStrRev(), Lef'(), Len(), Mi'(), Replac'().**

Example

```
#Language "WWB.NET"
Sub Main
    Debug.Print Righ'("Hello",3) "llo"
End Sub
```

RmDir Instruction

Syntax RmDir *Nam'*

Group File

Description Remove directory *Nam'*.

Parameter	Description
<i>Nam'</i>	This string value is the path and name of the directory. A path relative to the current directory can be used.

See Also **MkDir.**

Example

```
#Language "WWB.NET"
Sub Main
    RmDir "C:\WWTEMP"
End Sub
```

Rnd Function

Syntax Rnd([*Num*])

Group Math

Description Return a random number greater than or equal to zero and less than one.

Parameter	Description
<i>Num</i>	See table below.
Num	Description
<0	Return the same number every time, using <i>Num</i> as the seed.
>0	Return the next random number in the sequence.
0	Return the most recently generated number.

omitted	Return the next random number in the sequence.
---------	--

See Also**Randomize.****Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Rnd() ' 0.???????????????
End Sub
```

Round Function

Syntax Round([Num][, Places])**Group** Math**Description** Return the number rounded to the specified number of decimal places.

Parameter	Description
Num	Round this numeric value.'
Places	Round to this number of decimal places. If this is omitted then round to the nearest integer value.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Round(.5)    '0
    Debug.Print Round(.500001) '1
    Debug.Print Round(1.499999)'1
    Debug.Print Round(1.5)    '2
    Debug.Print Round(11.11)   '11
    Debug.Print Round(11.11,1)'11.1
End Sub
```

RSe' Function

Syntax RSe(', Len)**Group** String**Description** Return a string from ' with only the Len chars.

Parameter	Description
'	Return the this string value right justified.
Len	Return this many chars. If ' is shorter than that then fill the remainder with spaces.

See Also**LSe'().****Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print RSe("Hello",6) "Hello "
End Sub
```

RTri' Function

Syntax	RTri'()
Group	String
Description	Return the string with 's trailing spaces removed.
<hr/>	
Parameter	Description
'	Copy this string without the trailing spaces.'
<hr/>	
See Also	LTri'(), Tri'().
Example	<pre>#Language "WWB.NET" Sub Main Debug.Print ";" & RTri'(" x "); ";" & x. End Sub</pre>

SaveSetting Instruction

Syntax	SaveSetting <i>AppNam'</i> , <i>Sectio'</i> , <i>Ke'</i> , <i>Setting</i>
Group	Settings
Description	Save the <i>Setting</i> for Key in <i>Section</i> in project <i>AppName</i> . Win16 and Win32s store settings in a .ini file named <i>AppName</i> . Win32/Win64 store settings in the registration database under "HKEY_CURRENT_USER\ Software\ VB and VBA Program Settings\ <i>AppName</i> \ <i>Section</i> \ Key". If <i>AppName</i> starts with "..\" then "VB and VBA Program Settings\" is omitted.
<hr/>	
Parameter	Description
<i>AppNam'</i>	This string value is the name of the project which has this <i>Section</i> and <i>Key</i> .
<i>Sectio'</i>	This string value is the name of the section of the project settings.
<i>Ke'</i>	This string value is the name of the key in the section of the project settings.
<i>Setting</i>	Set the key to this value. (The value is stored as a string.)
<hr/>	
Example	<pre>#Language "WWB.NET" Sub Main SaveSetting "MyApp","Font","Size",10 End Sub</pre>

Second Function

Syntax	Second(<i>dateexpr</i>)
Group	Time/Date
Description	Return the second of the minute (0 to 59).
<hr/>	
Parameter	Description
<i>dateexpr</i>	Return the second of the minute for this date value.'
<hr/>	

See Also [Hour\(\), Minute\(\), Time\(\).](#)

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Second(#12:00:01 AM#) ' 1
End Sub
```

Seek Instruction

Syntax Seek'*StreamNum*, *Count*

Group File

Description Position *StreamNum* for input *Count*.

Note: Unicode text files opened with Input mode use character positions, not byte positions.

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
<i>Count</i>	For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1.

See Also [Seek\(\).](#)

Example

```
'#Language "WWB.NET"
Sub Main
    FileOpen 1, "XXX", OpenMode.Input
    L = LineInput(1)
    Seek'1, 1 ' rewind to start of file
    Input'1, A
    FileClose'1
    Debug.Print A
End Sub
```

Seek Function

Syntax Seek(*StreamNum*)

Group File

Description Return *StreamNum* current position. For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1.

Note: Unicode text files opened with Input mode use character positions, not byte positions.

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.

See Also [Seek.](#)

Example

```
'#Language "WWB.NET"
Sub Main
    FileOpen 1, "XXX", OpenMode.Input
    Debug.Print Seek(1) '1
    L = LineInput(1)
    Debug.Print Seek(1)
    FileClose'1
End Sub
```

Select Case Statement

Syntax

```
Select Case expr
[Case caseexpr[, ...] [ : instruction ]
   statements]...
[Case Else [ : instruction ]
   statements]
End Select
```

Group Flow Control

Description Select the appropriate case by comparing the *expr* with each of the *caseexprs*. Select the Case Else part if no *caseexpr* matches. (If the Case Else is omitted then skip the entire Select...End Select block.) Only the *statements* from the matched Case up to the next Case are executed.

caseexpr	Description
<i>expr</i>	Execute if equal.
Is < <i>expr</i>	Execute if less than.
Is <= <i>expr</i>	Execute if less than or equal to.
Is > <i>expr</i>	Execute if greater than.
Is >= <i>expr</i>	Execute if greater than or equal to.
Is <> <i>expr</i>	Execute if not equal to.
<i>expr1</i> To <i>expr2</i>	Execute if greater than or equal to <i>expr1</i> and less than or equal to <i>expr2</i> .

See Also **If, Choose(), IIf().**

Example

```
'#Language "WWB.NET"
Sub Main
    S = InputBox("Enter hello, goodbye, dinner or sleep:")
    Select Case UCase(S)
        Case "HELLO", "HI"
            Debug.Print "come in"
        Case "GOODBYE", "BYE"
            Debug.Print "see you later"
        Case "DINNER"
            Debug.Print "Please come in."
            Debug.Print "Dinner will be ready soon."
        Case "SLEEP"
            Debug.Print "Sorry."
            Debug.Print "We are full for the night"
        Case Else
            Debug.Print "What?"
```

```
End Select
End Sub
```

SendKeys Instruction

Syntax SendKeys *Key*[, *Wait*]

Group Miscellaneous

Description Send *Key*' to Windows.

Parameter	Description
<i>Key'</i>	Send the keys in this string value to Windows. (Refer to table below.)
<i>Wait</i>	If this is not zero then the keys are sent before executing the next instruction. If this is omitted or zero then the keys are sent during the following instructions.
<hr/>	
Key	Description
+	Shift modifier key: the following key is a shifted key
^	Ctrl modifier key: the following key is a control key
%	Alt modifier key: the following key is an alt key
(keys)	Modifiers apply to all keys
~	Send Enter key
k	Send k Key (k is any single char)
K	Send Shift k Key (K is any capital letter)
{special n}	special key (n is an optional repeat count)
{mouse x,y}	mouse key (x,y is an optional screen position)
{k}	Send k Key (any single char)
{K}	Send Shift k Key (any single char)
{Cancel}	Send Break Key
{Esc}	Send Escape Key
{Escape}	Send Escape Key
{Enter}	Send Enter Key
{Menu}	Send Menu Key (Alt)
{Help}	Send Help Key (?)
{Prtsc}	Send Print Screen Key
{Print}	Send
{Execute}	Send ?
{Tab}	Send
{Pause}	Send Pause Key
{Tab}	Send Tab Key
{BS}	Send Back Space Key
{BkSp}	Send Back Space Key
{BackSpace}	Send Back Space Key
{Del}	Send Delete Key
{Delete}	Send Delete Key
{Ins}	Send Insert Key
{Insert}	Send Insert Key
{Left}	Send Left Arrow Key
{Right}	Send Right Arrow Key
{Up}	Send Up Arrow Key

{Down}	Send Down Arrow Key
{PgUp}	Send Page Up Key
{PgDn}	Send Page Down Key
{Home}	Send Home Key
{End}	Send End Key
{Select}	Send ?
{Clear}	Send Num Pad 5 Key
{Pad0..9}	Send Num Pad 0-9 Keys
{Pad*}	Send Num Pad * Key
{Pad+}	Send Pad + Key
{PadEnter}	Send Num Pad Enter
{Pad.}	Send Num Pad . Key
{Pad-}	Send Num Pad - Key
{Pad/}	Send Num Pad / Key
{F1..24}	Send F1 to F24 Keys

Mouse

Mouse movement and button clicks:

- {Move x,y} - move the mouse to (x,y)
- {ClickLeft x,y} - move the mouse to (x,y) and click the left button. (This is the same as {DownLeft x,y}{UpLeft}.)
- {DoubleClickLeft x,y} - move the mouse to (x,y) and click the left button. (This is NOT the same as {ClickLeft x,y}{ClickLeft}.)
- {DownLeft x,y} - move the mouse to (x,y) and push the left button down.
- {UpLeft x,y} - move the mouse to (x,y) and release the left button.
- {...Middle x,y} - similarly named keys for the middle mouse button.
- {...Right x,y} - similarly named keys for the right mouse button.

The x,y values are screen pixel locations, where (0,0) is in the upper-left corner. In all cases the x,y is optional. If omitted, the previous mouse position is used.

See Also

[AppActivate](#), [KeyName](#), [Shell\(\)](#).

Example

```
'#Language "WWB.NET"
Sub Main
    SendKeys "%S"    ' send Alt-S (Search)
    SendKeys "GoTo~~" ' send G o T o {Enter} {Enter}
End Sub
```

SetAttr Instruction

Syntax

SetAttr *Nam'*, *Attrib*

Group

File

Description

Set the *attributes* for file *Nam'*. If the file does not exist then a run-time error occurs.

Parameter	Description
<i>Nam'</i>	This string value is the path and name of the file. A path relative to the current directory can be used.

Attrib Set the file's *attributes* to this numeric value.

Example

```
'#Language "WWB.NET"
Sub Main
    Attrib = GetAttr("XXX")
    SetAttr "XXX",1 ' readonly
    Debug.Print GetAttr("XXX") ' 1
    SetAttr "XXX",Attrib
End Sub
```

SetLocale Instruction

Syntax SetLocale *LocaleID*

Group Miscellaneous

Description Set the *LocaleID* for the current thread.

Pocket PC Not supported.

Parameter	Description
<i>LocaleID</i>	Set the current thread's locale to this value.

See Also **GetLocale.**

Example

```
'#Language "WWB.NET"
Sub Main
    SetLocale &H409 ' English, US
End Sub
```

Sgn Function

Syntax Sgn(*Num*)

Group Math

Description Return the sign.

Parameter	Description
<i>Num</i>	Return the sign of this numeric value. Return -1 for negative. Return 0 for zero. Return 1 for positive.

See Also **Abs.**

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Sgn(9) ' 1
    Debug.Print Sgn(0) ' 0
    Debug.Print Sgn(-9) '-1
End Sub
```

Shell Function

Syntax	<code>Shell(Nam[, WindowType])</code>	
Group	Miscellaneous	
Description	Execute program <i>Nam</i> '. This is the same as using File Run from the Program Manager. This instruction can run .COM, .EXE, .BAT and .PIF files. If successful, return the task ID.	
Pocket PC	The <i>WindowType</i> parameter is ignored.	
<hr/>		
Parameter	Description	
<i>Nam'</i>	This string value is the path and name of the program to run. Command line arguments follow the program name. (A long file name containing a space must be surrounded by literal double quotes.)	
<i>WindowType</i>	This controls how the application's main window is shown. See the table below.	
<hr/>		
WindowType	Value	Effect
vbHide	0	Hide Window
vbNormalFocus	1, 5, 9	Normal Window
vbMinimizedFocus		
	2	Minimized Window (default)
vbMaximizedFocus	3	Maximized Window
vbNormalNoFocus	4, 8	Normal Deactivated Window
vbMinimizedNoFocus	6, 7	Minimized Deactivated Window
<hr/>		

See Also [AppActivate](#), [SendKeys](#).

Example

```
'#Language "WWB.NET"
Sub Main
    X = Shell("Calc") ' run the calc program
    AppActivate X
    SendKeys "% R" ' restore calc's main window
    SendKeys "30*2{+}10=",1 '70
End Sub
```

SByte Data Type

Syntax	<code>Dim v As SByte</code>
Group	Data Type
Description	An 8 bit signed integer value.

Short Data Type

Syntax	<code>Dim v As Short</code>
---------------	-----------------------------

Group	Data Type
Description	A 16 bit signed integer value.

ShowPopupMenu Function

Syntax ShowPopupMenu(*StrArra'()*[, *PopupMenuStyle*][, *XPos*, *YPos*])

Group User Input

Description Show a popup menu and return the number of the item selected. The item number is the index of the StrArray selected minus LBound(StrArray). The value -1 is returned in no menu item is selected.

Parameter	Description
<i>StrArra'()</i>	This one-dimensional array of strings establishes the list of choices. All the non-null elements of the array are used.
<i>PopupMenuStyle</i>	This controls how the popup menu is aligned. Any combination of styles may be used together. See the table below.
<i>XPos</i>	When the menu is put up the alignment will be at this window position. If this is omitted then the current mouse position is used.
<i>YPos</i>	When the menu is put up the alignment will be at this window position. If this is omitted then the current mouse position is used.

PopupMenuStyle	Value	Effect
vbPopupLeftTopAlign	0	Align menu left edge at XPos and top at YPos. (default)
vbPopupUseLeftButton	1	User can select menu choices with the left mouse button only.
vbPopupUseRightButton	2	User can select menu choices with the left or right mouse button.
vbPopupRightAlign	4	Align menu with right edge at the XPos.
vbPopupCenterAlign	8	Align menu center at the XPos.
vbPopupVCenterAlign	16	Align menu center at the YPos.
vbPopupBottomAlign	32	Align menu bottom at the YPos.

Example

```
'#Language "WWB.NET"
Sub Main
    Dim Items(0 To 2) As String
    Items(0) = "Item &1"
    Items(1) = "Item &2"
    Items(2) = "Item &3"
    X = ShowPopupMenu(Items) ' show popup menu
    Debug.Print X ' item selected
End Sub
```

Sin Function

Syntax Sin(*Num*)

Group Math

Description Return the sine.

Parameter	Description
<i>Num</i>	Return the sine of this numeric value. This is the number of radians. There are 2*Pi radians in a full circle.

See Also**Atn, Cos, Tan.****Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Sin(1) ' 0.8414709848079
End Sub
```

Single Data Type

Syntax**Dim v As Single****Group**

Data Type

Description

A 32 bit real value.

Spac' Function

Syntax**Spac'(*Len*)****Group**

String

DescriptionReturn the string *Len* spaces long.

Parameter	Description
<i>Len</i>	Create a string this many spaces long.

See Also**StrDu'()**.**Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print ":"; Spac'(3); ":" "."
End Sub
```

Split Function

Syntax**Split(*Str*, [*Sep*], [*Max*])****Group**

Miscellaneous

Description

Return a string array containing substrings from the original string.

Parameter	Description
<i>Str</i>	Extract substrings from this string value.
<i>Sep</i>	Look for this string value to separate the substrings. (Default: "")
<i>Max</i>	Create at most this many substrings. (Default -1, which means create as many as are found.)

See Also**Join()**.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Split("1 2 3")(1) "2"
End Sub
```

Sqr Function

Syntax

`Sqr(Num)`

Group

Math

Description

Return the square root.

Parameter	Description
<code>Num</code>	Return the square root of this numeric value.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Sqr(9) '3
End Sub
```

Static Definition

Syntax

`Static vardeclaration [= initialvalue][, ...]`

Group

Declaration

Description

A static variable retains its value between *procedure* calls. Dimension var array (s) using the *dimensions* to establish the minimum and maximum index value for each dimension. If the *dimensions* are omitted then a scalar (single value) variable is defined. A dynamic array is declared using () without any *dimensions*. It must be **ReDim**ensioned before it can be used.

See Also

Dim, Option Base, Private, Public, ReDim.

Example

`'#Language "WWB.NET"`

```
Sub A
    Static X
    Debug.Print X
    X = "Hello"
End Sub
```

```
Sub Main
    A
    A ' prints "Hello"
End Sub
```

Stop Instruction

Syntax

`Stop`

Group	Flow Control
Description	Pause execution. If execution is resumed then it starts at the next instruction. Use End to terminate the <i>macro</i> completely.
Example	'#Language "WWB.NET" Sub Main For I = 1 To 10 Debug.Print I If I = 3 Then Stop Next I End Sub

St' Function

Syntax	St'(Num)
Group	String
Description	Return the string representation of <i>Num</i> .
<hr/>	
Parameter	Description
Num	Return the string representation of this numeric value. Positive values begin with a blank. Negative values begin with a dash '-'.
<hr/>	
See Also	CStr(), He'(), Oc'(), Val().
Example	'#Language "WWB.NET" Sub Main Debug.Print St'(9*9) ' 81 End Sub

StrCom' Function

Syntax	StrComp(<i>Str1,Str2,Comp</i>)	
Group	String	
Description	Compare two strings.	
<hr/>		
Parameter	Description	
<i>Str1</i>	Compare this string with <i>Str2</i> . '	
<i>Str2</i>	Compare this string with <i>Str1</i> . '	
<i>Comp</i>	This numeric value indicates the type of comparison. See <i>Comp</i> table below.	
<hr/>		
Result	Description	
-1	<i>Str1</i> is less than <i>Str2</i> .	
0	<i>Str1</i> is equal to <i>Str2</i> .	
1	<i>Str1</i> is greater than <i>Str2</i> .	
<hr/>		
Comp	Value	Effect
vbUseCompareOption	-1	Performs the comparison using the Option Compare statement value.

vbBinaryCompare	0	C.compares the string's binary data.
vbTextCompare	1	C.compares the string's text using the collation rules.
vbDatabaseCompare	2	Microsoft Access only. (Not supported.)

See Also**LCas'()**, **Option Compare**, **StrCon'()**, **UCas'()**.**Example**

```
#Language "WWB.NET"
Sub Main
    Debug.Print StrComp("F","e") '-1
    Debug.Print StrComp("F","e",1) '1
    Debug.Print StrComp("F","t",1) '0
End Sub
```

StrCon' Function

Syntax `StrCon'(Str,Conv)`**Group** String**Description** Convert the string.

Parameter	Description	
<i>Str</i>	Convert this string value.'	
<i>Conv</i>	This numeric value indicates the type of conversion. See conversion table below.	
<hr/>		
Conv	Value	Effect
vbUpperCase	1	Convert a String to upper case.
vbLowerCase	2	Convert a String to lower case.
vbProperCase	3	Convert a String to proper case. (Not supported.)
vbWide	4	Convert a String to wide. (Only supported for eastern locales.)
vbNarrow	8	Convert a String to narrow. (Only supported for eastern locales.)
vbKatakana	16	Convert a String to Katakana. (Only supported for Japanese locales.)
vbHiragana	32	Convert a String to Hiragana. (Only supported for Japanese locales.)
vbUnicode or vbFromANSIBytes	64	Convert an ANSI (locale dependent) byte array to a Unicode string.
vbANSI	4160	Convert an ANSI (locale dependent) string to a Unicode string.
vbFromUnicode or vbANSIBytes	128	Convert from Unicode to an ANSI (locale dependent) byte array.
vbANSI	4224	Convert from Unicode to an ANSI (locale dependent) string.
vbUTF8	4352	Convert a Unicode string to a UTF-8 string.
vbUTF8Bytes	256	Convert a Unicode string to a UTF-8 byte array.
vbFromUTF8	4608	Convert a UTF-8 string to a Unicode string.
vbFromUTF8Bytes	512	Convert a UTF-8 byte array to a Unicode string.
vbToBytes	1024	Convert a String to a byte array containing the low byte of each char.
vbFromBytes	2048	Convert a byte array to a String by setting the low byte of each char.

Conversion Rules If multiple conversions are specified, the conversions occur in this order:

- vbFromBytes, vbUnicode, vbFromANSI, vbFromANSIBytes, vbFromUTF8 or vbFromUTF8Bytes (choose one, optional)

- vbUpperCase, vbLowerCase, vbWide, vbNarrow, vbKatakana or vbHiragana (choose one or more, optional)
- vbToBytes, vbFromUnicode, vbANSI, vbANSIBBytes, vbUTF8 or vbUTF8Bytes (choose one, optional)

See Also [LCas'\(\)](#), [StrComp\(\)](#), [UCas'\(\)](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Dim B(1 To 3) As Byte
    B(1) = 65
    B(2) = 66
    B(3) = 67
    Debug.Print StrCon'(B,vbUnicode) "ABC"
End Sub
```

StrDu' Function

Syntax `StrDu'(Len, Char)`

Group String

Description Return the string *Len* long filled with *Char* or the first char of *Cha'*.

Parameter	Description
<i>Len</i>	Create a string this many chars long.
<i>Char</i>	Fill the string with this char value. If this is a numeric value then use the ASCII char equivalent. If this is a string value use the first char of that string. '

See Also [Spac'\(\)](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print StrDu'(4,65)  "AAAA"
    Debug.Print StrDu'(4,"ABC") "AAAA"
End Sub
```

String Data Type

Syntax `Dim v As String`

Group Data Type

Description An arbitrary length string value. Some useful string constants are predefined:

- vbNullChar - same as Chr(0)
- vbNullString - not implemented
- vbCrLf - same as Chr(13) & Chr(10)
- vbCr - same as Chr(13)
- vbLf - same as Chr(10)
- vbBack - same as Chr(8)

- vbFormFeed - same as Chr(12)
- vbTab - same as Chr(9)
- vbVerticalTab - same as Chr(11)

StrRevers' Function

Syntax StrRevers'(S)

Group String

Description Return the string with the characters in reverse order.

Parameter	Description
S	Return this string with the characters in reverse order.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print StrRevers("ABC") 'CBA
End Sub
```

Structure Definition

Syntax [| Private | Public]_

```
Structure name
    [ Dim | Private | Public ] elem [(dimension[, ...])] As type
    [...]
End Structure
```

Group Declaration

Description Define a new *user structure*. Each *elem* defines an element of the type for storing data. *As type* defines the type of data that can be stored. A *user defined structure variable* has a value for each *elem*. Use *.elem* to access individual element values.

Access If no access is specified then **Public** is assumed.

Example

```
'#Language "WWB.NET"
Structure Employee
    Dim FirstName As String
    Dim LastName As String
    Dim Title As String
    Dim Salary As Double
End Structure

Sub Main
    Dim e As Employee
    e.FirstName = "John"
    e.LastName = "Doe"
    e.Title = "President"
    e.Salary = 100000
```

```

    Debug.Print e.FirstName "John"
    Debug.Print e.LastName "Doe"
    Debug.Print e.Title "President"
    Debug.Print e.Salary ' 100000
End Sub

```

Sub Definition

Syntax	[Private Public Friend]_ Sub <i>name</i> [(<i>param</i> [, ...])] _ [Handles <i>var.eventname</i>] <i>statements</i> End Sub
Group	Declaration
Description	User defined subroutine. The subroutine defines a set of <i>statements</i> to be executed when it is called. The values of the calling <i>arglist</i> are assigned to the <i>params</i> . A subroutine does not return a result.
Access	If no access is specified then Public is assumed.
Handles	The Sub provides an event handler for the WithEvents variable's (<i>var</i>) event (<i>eventname</i>).
See Also	Declare, Function, Property, WithEvents.
Example	<pre>'#Language "WWB.NET" Sub IdentityArray(A()) ' A() is an array of numbers For I = LBound(A) To UBound(A) A(I) = I Next I End Sub Sub CalcArray(A(), B, C) ' A() is an array of numbers For I = LBound(A) To UBound(A) A(I) = A(I)*B+C Next I End Sub Sub ShowArray(A()) ' A() is an array of numbers For I = LBound(A) To UBound(A) Debug.Print "("; I; ")="; A(I) Next I End Sub Sub Main Dim X(1 To 4) IdentityArray X() ' X(1)=1, X(2)=2, X(3)=3, X(4)=4 CalcArray X(), 2, 3 ' X(1)=5, X(2)=7, X(3)=9, X(4)=11 ShowArray X() ' print X(1), X(2), X(3), X(4) End Sub</pre>

SystemTypeName Function

Syntax	SystemTypeName'(Name)
Group	Variable Info
Description	Return a fully qualified Common Language Runtime type name corresponding to the VB type name. Return Nothing if the specified <i>Name</i> is not a valid VB type name.
<hr/>	
Parameter	Description
Name	This is the VB type name.
<hr/>	
See Also	TypeName, VbTypeName, VarType.
Example	<pre>'#Language "WWB.NET" Sub Main Dim X As Object Debug.Print SystemTypeName(X) "Empty" X = 1 Debug.Print SystemTypeName(X) "Integer" End Sub</pre>

Tan Function

Syntax	Tan(Num)
Group	Math
Description	Return the tangent.
<hr/>	
Parameter	Description
Num	Return the tangent of this numeric value.
<hr/>	
See Also	Atn, Cos, Sin.
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print Tan(1) ' 1.5574077246549 End Sub</pre>

Text Dialog Item Definition

Syntax	Text <i>X, Y, DX, DY, Titl[, .Field][, Options]</i>
Group	User Dialog
Description	Define a text item.
<hr/>	
Parameter	Description
X	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.

<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Title</i>	The value of this string is the title of the text control.
<i>Field</i>	This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i> . If this identifier is omitted then the first two words of the title are used.
<i>Options</i>	This numeric value controls the alignment of the text. Choose one value from following table. (If this numeric value omitted then zero is used.)

Option	Description
0	Text is left aligned.
1	Text is right aligned.
2	Text is centered.

See Also**Begin Dialog.****Example**

```
'#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120
        Text 10,10,180,15,"Please push the OK button"
        OKButton 80,90,40,20
    End Dialog
    Dim dlg As UserDialog
    Dialog dlg ' show dialog (wait for ok)
End Sub
```

TextBox Dialog Item Definition

Syntax `TextBox X, Y, DX, DY, .Field[, Options]`**Group** User Dialog**Description** Define a textbox item.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Field</i>	The value of the text box is accessed via this field.
<i>Options</i>	This numeric value controls the type of text box. Choose one value from following table. (If this numeric value omitted then zero is used.)

Option	Description
0	Text box allows a single line of text to be entered.
1	Text box allows multiple lines of text can be entered.
2	Locked text box displays multiple lines of text.

-1 Text box allows a hidden password can be entered.

See Also**Begin Dialog.****Example**

```
'#Language "WWB.NET"
Sub Main
    Begin Dialog UserDialog 200,120
        Text 10,10,180,15,"Please push the OK button"
        TextBox 10,25,180,20,.Tex'
        OKButton 80,90,40,20
    End Dialog
    Dim dlg As UserDialog
    dlg.Tex' = "none"
    Dialog dlg ' show dialog (wait for ok)
    Debug.Print dlg.Tex'
End Sub
```

Timer Function

Syntax

Timer

Group

Time/Date

Description

Return the number of seconds past midnight. (This is a real number, accurate to about 1/18th of a second.)

See Also**Date, Now, Time.****Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Timer ' example: 45188.13
End Sub
```

TimeSerial Function

SyntaxTimeSerial(*Hour, Minute, Second*)**Group**

Time/Date

DescriptionReturn a **Date** value.

Parameter	Description
<i>Hour</i>	This numeric value is the hour (0 to 23).
<i>Minute</i>	This numeric value is the minute (0 to 59).
<i>Second</i>	This numeric value is the second (0 to 59).

See Also**DateSerial, DateValue, TimeValue.****Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print TimeSerial(13,30,0) '1:30:00 PM
End Sub
```

TimeValue Function

Syntax TimeValue(*Dat'*)

Group Time/Date

Description Return the time part of date encoded as a string value.

Parameter	Description
<i>Dat'</i>	Convert this string value to the time part of date it represents.

See Also **DateSerial**, **DateValue**, **TimeSerial**.

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print TimeValue("1/1/2000 12:00:01 AM")
    '12:00:01 AM
End Sub
```

Throw Instruction/Function

Syntax Throw [*exception*]

Group Throw Handling

Description Throw an exception the caller. Execution of the current procedure is terminated immediately.

Parameter	Description
<i>exception</i>	Throw this exception. The exception expression may be omitted in a Catch block.

See Also **Try**.

Example

```
'#Language "WWB.NET"
Sub Main
    Try
        Dolt
        Catch Ex As Exception
            Debug.Print Ex.ToString() "error"
        End Try
    End Sub

    Sub Dolt
        Throw New Exception("error")
    End Sub
```

Tri' Function

Syntax Tri'()

Group String

Description Return the string with 's leading and trailing spaces removed.

Parameter	Description
'	Copy this string without the leading or trailing spaces.'

See Also [LTri'\(\)](#), [RTri'\(\)](#).**Example**

```
#Language "WWB.NET"
Sub Main
    Debug.Print "."; Tri'(" x "); ".x."
End Sub
```

True Keyword

Group Constant**Description** A *conditional expression* is True when its value is non-zero. A function that returns True returns the value -1.

Try Statement

Syntax

```
Try
    statements
[ Catch [exception [As type]] [When condition]
    statements]...
[ Finally
    statements]
End Try
```

Group Error Handling**Description** A Try block allows exceptions to be handled programmatically. When an exception occurs in the Try block, each Catch block is checked for a matching *condition*. (If the condition expression is omitted, the Catch block is matched.) If a matching Catch block is found, the block's statements are executed. The optional Finally block is always executed regardless of whether an exception occurs or not.**See Also** [Throw](#), [Using](#).**Example**

```
#Language "WWB.NET"
Sub Main
    Try
        Dolt
    Catch Ex As System.Exception
        Debug.Print Ex.ToString() "error"
    End Try
End Sub

Sub Dolt
    Throw New System.Exception("error")
End Sub
```

TryCast Function

Syntax	<code>TryCast(expr, objtype)</code>						
Group	Conversion						
Description	Return <i>expr</i> 's type is related to <i>objtype</i> type. If it is not, Nothing will be returned.						
	<hr/>						
	<table border="1"> <thead> <tr> <th>Parameter</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>expr</i></td><td>Cast the value of this expression.</td></tr> <tr> <td><i>objtype</i></td><td>Cast to this type.</td></tr> </tbody> </table> <hr/>	Parameter	Description	<i>expr</i>	Cast the value of this expression.	<i>objtype</i>	Cast to this type.
Parameter	Description						
<i>expr</i>	Cast the value of this expression.						
<i>objtype</i>	Cast to this type.						
See Also	 CType, DirectCast.						
Example	<pre>'#Language "WWB.NET" Sub Main Dim V As Object V = Err Debug.Print TypeName(TryCast(V, ErrObject)) ' ErrObject End Sub</pre>						

TypeName Function

Syntax	<code>TypeNam'(var)</code>																																						
Group	Variable Info																																						
Description	Return a string indicating the type of value stored in <i>var</i> .																																						
	<hr/>																																						
	<table border="1"> <thead> <tr> <th>Parameter</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>var</i></td><td>Return a string indicating the type of value stored in this variable.</td></tr> </tbody> </table> <hr/>	Parameter	Description	<i>var</i>	Return a string indicating the type of value stored in this variable.																																		
Parameter	Description																																						
<i>var</i>	Return a string indicating the type of value stored in this variable.																																						
	<table border="1"> <thead> <tr> <th>Result</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Empty</td><td><i>Object</i> variable is empty. It has never been assigned a value.</td></tr> <tr> <td>Null</td><td><i>Object</i> variable is null.</td></tr> <tr> <td>Boolean</td><td>Variable contains a Boolean value.</td></tr> <tr> <td>Byte</td><td>Variable contains a Byte value.</td></tr> <tr> <td>SByte</td><td>Variable contains a SByte value.</td></tr> <tr> <td>Short</td><td>Variable contains an Short value.</td></tr> <tr> <td>UShort</td><td>Variable contains an UShort value.</td></tr> <tr> <td>Integer</td><td>Variable contains an Integer value.</td></tr> <tr> <td>UInteger</td><td>Variable contains an UInteger value.</td></tr> <tr> <td>Long</td><td>Variable contains a Long value.</td></tr> <tr> <td>ULong</td><td>Variable contains a ULong value.</td></tr> <tr> <td>Decimal</td><td>Variable contains a Decimal value.</td></tr> <tr> <td>Single</td><td>Variable contains a Single value.</td></tr> <tr> <td>Double</td><td>Variable contains a Double value.</td></tr> <tr> <td>Date</td><td>Variable contains a Date value.</td></tr> <tr> <td>String</td><td>Variable contains a String value.</td></tr> <tr> <td>Object</td><td>Variable contains an Object reference that is not Nothing. (An object may return a type name specific to that type of object.)</td></tr> <tr> <td>Nothing</td><td>Variable contains an Object reference that is Nothing.</td></tr> </tbody> </table>	Result	Description	Empty	<i>Object</i> variable is empty. It has never been assigned a value.	Null	<i>Object</i> variable is null.	Boolean	Variable contains a Boolean value.	Byte	Variable contains a Byte value.	SByte	Variable contains a SByte value.	Short	Variable contains an Short value.	UShort	Variable contains an UShort value.	Integer	Variable contains an Integer value.	UInteger	Variable contains an UInteger value.	Long	Variable contains a Long value.	ULong	Variable contains a ULong value.	Decimal	Variable contains a Decimal value.	Single	Variable contains a Single value.	Double	Variable contains a Double value.	Date	Variable contains a Date value.	String	Variable contains a String value.	Object	Variable contains an Object reference that is not Nothing. (An object may return a type name specific to that type of object.)	Nothing	Variable contains an Object reference that is Nothing.
Result	Description																																						
Empty	<i>Object</i> variable is empty. It has never been assigned a value.																																						
Null	<i>Object</i> variable is null.																																						
Boolean	Variable contains a Boolean value.																																						
Byte	Variable contains a Byte value.																																						
SByte	Variable contains a SByte value.																																						
Short	Variable contains an Short value.																																						
UShort	Variable contains an UShort value.																																						
Integer	Variable contains an Integer value.																																						
UInteger	Variable contains an UInteger value.																																						
Long	Variable contains a Long value.																																						
ULong	Variable contains a ULong value.																																						
Decimal	Variable contains a Decimal value.																																						
Single	Variable contains a Single value.																																						
Double	Variable contains a Double value.																																						
Date	Variable contains a Date value.																																						
String	Variable contains a String value.																																						
Object	Variable contains an Object reference that is not Nothing. (An object may return a type name specific to that type of object.)																																						
Nothing	Variable contains an Object reference that is Nothing.																																						

Error	Variable contains a error code value.
Unknown	Variable contains a non-ActiveX Automation object reference.
()	Variable contains an array value. The TypeName of the element followed by () .

See Also**SystemTypeName, VarType, VbTypeName.****Example**

```
'#Language "WWB.NET"
Sub Main
    Dim X As Object
    Debug.Print TypeName(X) "Empty"
    X = 1
    Debug.Print TypeName(X) "Integer"
    X = 100000
    Debug.Print TypeName(X) "Long"
    X = 1.1
    Debug.Print TypeName(X) "Double"
    X = "A"
    Debug.Print TypeName(X) "String"
    'X = CreateObject("Word.Basic")
    Debug.Print TypeName(X) "Object"
    X = Array(0,1,2)
    Debug.Print TypeName(X) "Object()"
End Sub
```

TypeOf Operator

Syntax `TypeOf expr Is objtype`**Group** Operator**Description** Return the **True** if *expr* refers to an object of *objtype*.**See Also** **Objects.****Example** '#Language "WWB.NET"

```
Sub Main
    Debug.Print TypeOf Err Is ErrObject ' True
End Sub
```

UBound Function

Syntax `UBound(arrayvar[, dimension])`**Group** Variable Info**Description** Return the highest index.

Parameter	Description
<i>arrayvar</i>	Return the highest index for this array variable.
<i>dimension</i>	Return the highest index for this dimension of <i>arrayvar</i> . If this is omitted then return the highest index for the first dimension.

See Also**LBound().**

Example

```
'#Language "WWB.NET"
Sub Main
    Dim A(3,6)
    Debug.Print UBound(A) ' 3
    Debug.Print UBound(A,1) ' 3
    Debug.Print UBound(A,2) ' 6
End Sub
```

UCas' Function

Syntax	UCas'()
Group	String
Description	Return a string from ' where all the lowercase letters have been uppercased.
<hr/>	
Parameter	Description
'	Return the string value of this after all chars have been converted to lowercase.'
<hr/>	
See Also	LCas'(), StrComp(), StrCon'().
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print UCas'("Hello") ""HELLO" End Sub</pre>

UInteger Data Type

Syntax	Dim v As UInteger
Group	Data Type
Description	A 32 bit unsigned integer value. The number of bits is controlled by the #Language setting.

ULong Data Type

Syntax	Dim v As ULong
Group	Data Type
Description	A 64 bit unsigned integer value. The number of bits is controlled by the #Language setting.

Unlock Instruction

Syntax	Unlock <i>StreamNum</i> -or- Unlock <i>StreamNum</i> , <i>RecordNum</i> -or- Unlock <i>StreamNum</i> , [<i>start</i>] To <i>end</i>
Group	File
Description	<p>Form 1: Unlock all of <i>StreamNum</i>.</p> <p>Form 2: Unlock a record (or byte) of <i>StreamNum</i>.</p> <p>Form 3: Unlock a range of records (or bytes) of <i>StreamNum</i>. If <i>start</i> is omitted then unlock starting at the first record (or byte).</p> <p>Note: For sequential files (Input, Output and Append) unlock always affects the entire file.</p>
<hr/>	
Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
<i>RecordNum</i>	For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1.
<i>start</i>	First record (or byte) in the range.
<i>end</i>	Last record (or byte) in the range.

See Also [Lock, Open.](#)

Example

```
'#Language "WWB.NET"
Sub Main
    Dim V As Variant
    FileOpen 1, "SAVE_V.DAT", OpenMode.Binary
    Lock'1
    Get'1, 1, V
    V = "Hello"
    Put'1, 1, V
    Unlock'1
    FileClose'1
End Sub
```

UShort Data Type

Syntax	Dim <i>v</i> As UShort
Group	Data Type
Description	A 16 bit unsigned integer value.

Val Function

Syntax	Val()
Group	String
Description	Return the value of the '.
<hr/>	
Parameter	Description
'	Return the numeric value for this string value. A string value begins with &O is an octal number. A string value begins with &H is a hex number. Otherwise it is decimal number.
<hr/>	
Example	<pre>'#Language "WWB.NET" Sub Main Debug.Print Val("-1000") '-1000 End Sub</pre>

Using Statement

Syntax	<pre>Using expr statements End Using</pre> <p>-or-</p> <pre>Using vardeclaration[, ...] statements End Using</pre>
Group	Error Handling
Description	A Using block allows resources to be freed automatically. A resource implements the IDisposable interface. When the End Using statement is executed all of the resources are freed.
See Also	Try.
Example	<pre>'#Language "WWB.NET" Sub Main Using res As New Resource ' ... use res End Using ' res.Dispose is called End Sub</pre>

VarType Function

Syntax	VarType(<i>var</i>)
Group	Variable Info
Description	Return a number indicating the type of value stored in <i>var</i> .
<hr/>	
Parameter	Description

<code>var</code>		Return a number indicating the type of value stored in this variable.
Result	Value	Description
<code>vbEmpty</code>	0	<i>Object</i> variable is Nothing. It has never been assigned a value.
<code>vbShort</code>	2	Variable contains an Short value.
<code>vbInteger</code>	3	Variable contains an Integer value. The value depends on the #Language setting.
<code>vbLong</code>	20	Variable contains a Long value. The value depends on the #Language setting.
<code>vbSingle</code>	4	Variable contains a Single value.
<code>vbDouble</code>	5	Variable contains a Double value.
<code>vbCurrency</code>	6	Variable contains a Currency value.
<code>vbDate</code>	7	Variable contains a Date value.
<code>vbString</code>	8	Variable contains a String value.
<code>vbObject</code>	9	Variable contains an Object reference.
<code>vbError</code>	10	Variable contains a error code value.
<code>vbBoolean</code>	11	Variable contains a Boolean value.
<code>vbDataObject</code>	13	Variable contains a non-ActiveX Automation object reference.
<code>vbDecimal</code>	14	Variable contains a Decimal value.
<code>vbSByte</code>	16	Variable contains a SByte value.
<code>vbByte</code>	17	Variable contains a Byte value.
<code>vbUShort</code>	18	Variable contains a UShort value.
<code>vbUInteger</code>	19	Variable contains a UInteger value. The value depends on the #Language setting.
<code>vbULong</code>	21	Variable contains a ULong value. The value depends on the #Language setting.
<code>vbUserDefinedType</code>	36	Variable contains a User Defined Structure value.
<code>+vbArray</code>	8192	Variable contains an array value. Use VarType() And 255 to get the type of element stored in the array.

See Also**SystemTypeName**, **TypeName**, **VbTypeName**.**Example**

```
'#Language "WWB.NET"
Sub Main
    Dim X As Object
    Debug.Print VarType(X) ' 0
    X = 1
    Debug.Print VarType(X) ' 2
    X = 100000
    Debug.Print VarType(X) ' 3
    X = 1.1
    Debug.Print VarType(X) ' 5
    X = "A"
    Debug.Print VarType(X) ' 8
    'X = CreateObject("Word.Basic")
    Debug.Print VarType(X) ' 9
    X = Array(0,1,2)
    Debug.Print VarType(X) ' 8204 (8192+12)
End Sub
```

VbTypeName Function

Syntax	VbTypeNam'(Name)
Group	Variable Info
Description	Return a string indicating VB type name corresponding to the fully qualified Common Language Runtime type name. Return Nothing if the specified <i>Name</i> is not a valid VB type name.

Parameter	Description
<i>Name</i>	This is Common Language Runtime type name.

See Also [SystemTypeName](#), [TypeName](#), [VarType](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Dim X As Object
    Debug.Print VbTypeName(X) "Empty"
    X = 1
    Debug.Print VbTypeName(X) "Integer"
End Sub
```

Wait Instruction

Syntax	Wait [<i>Delay</i>]
Group	Miscellaneous
Description	Wait for <i>Delay</i> seconds.

Parameter	Description
<i>Delay</i>	Wait for this number of seconds. If omitted then wait for 5 seconds.'

Example

```
'#Language "WWB.NET"
Sub Main
    Wait .5 'wait for one half second
End Sub
```

Weekday Function

Syntax	Weekday(<i>dateexpr</i>)
Group	Time/Date
Description	Return the weekday.

- vbSunday (1) - Sunday
- vbMonday (2) - Monday
- vbTuesday (3) - Tuesday
- vbWednesday (4) - Wednesday

- vbThursday (5) - Thursday
- vbFriday (6) - Friday
- vbSaturday (7) - Saturday

Parameter	Description
dateexpr	Return the weekday for this date value.'

See Also**Date(), Day(), Month(), WeekdayName(), Year().****Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Weekday(#1/1/1900#) ' 2
    Debug.Print Weekday(#1/1/2000#) ' 7
End Sub
```

WeekdayName Function

Syntax

WeekdayName(NumZ{day}[, CondZ{abbrev}])

Group

Time/Date

Description

Return the localized name of the weekday.

Parameter	Description
day	Return the localized name of this weekday. (1-7)
abbrev	If this conditional value is True then return the abbreviated form of the weekday name.

See Also**Weekday().****Example**

```
'#Language "WWB.NET"
Sub Main
    Debug.Print WeekdayName(1) 'Sunday
    Debug.Print WeekdayName(Weekday(Now))
End Sub
```

While Statement

Syntax

```
While condexpr
    statements
End While
```

Group

Flow Control

DescriptionExecute *statements* while *condexpr* is **True**.**See Also****Do, For, For Each, Exit** While.**Example**

```
'#Language "WWB.NET"
Sub Main
    I = 2
    While I < 10
```

```
I = I*2
End While
Debug.Print I ' 16
End Sub
```

Win16 Keyword

Group	Constant
Description	True if running in 16 bits. False if running in 32 or 64 bits.

Win32 Keyword

Group	Constant
Description	True if running in 32 bits. False if running in 16 or 64 bits.

Win64 Keyword

Group	Constant
Description	True if running in 64 bits. False if running in 16 or 32 bits.

With Statement

Syntax	<pre>With <i>objexpr</i> <i>statements</i> End With</pre>
Group	Object
Description	<i>Method</i> and <i>property</i> references may be abbreviated inside a With block. Use <i>.method</i> or <i>.property</i> to access the object specified by the With <i>objexpr</i> .
Example	<pre>'#Language "WWB.NET" Sub Main Dim App As Object 'App = CreateObject("WinWrap.CppDemoApplication") With App .Move 20,30 ' move icon to 20,30 End With End Sub</pre>

WithEvents Definition

Syntax	<pre>[Dim Private Public]_ WithEvents <i>name</i> As <i>objtype</i>[, ...]</pre>
---------------	--

Group	Declaration
Description	Dimensioning a module level variable WithEvents allows the macro to implement event handling Subs . The variable's As type must be a type from a referenced type library (or language extension) which implements events.
See Also	Dim, Private, Public, RaiseEvent.
Example	<pre>'#Language "WWB.NET" Dim WithEvents X As Thing Sub Main 'X = New Thing X.Dolt ' Dolt method raises DoingIt event End Sub Private Sub X_DoingIt Handles X.DoingIt Debug.Print "X.DoingIt event" End Sub</pre>

Write Instruction

Syntax	Write <i>StreamNum</i> , <i>expr</i> [, ...]
Group	File
Description	Write's <i>expr</i> (s) to <i>StreamNum</i> . String values are quoted. Null values are written as #NULL#. Boolean values are written as #FALSE# or #TRUE#. Date values are written as #date#. Error codes are written as #ERROR number#.
See Also	Input, LineInput, Print, PrintLine, WriteLine.
Example	<pre>'#Language "WWB.NET" Sub Main Dim A, B, C A = 1 B = 2 ' = "Hello" FileOpen 1, "XXX", OpenMode.Output Write'1, A, B, ' FileClose'1 End Sub</pre>

WriteLine Instruction

Syntax	WriteLine <i>StreamNum</i> , <i>expr</i> [, ...]
Group	File
Description	Write's <i>expr</i> (s) to <i>StreamNum</i> . String values are quoted. A newline is printed at the end. Null values are written as #NULL#. Boolean values are written as #FALSE# or #TRUE#. Date values are written as #date#. Error codes are written as #ERROR number#.

See Also [Input](#), [LineInput](#), [Print](#), [PrintLine](#), [Write](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Dim A, B, C
    A = 1
    B = 2
    ' = "Hello"
    FileOpen 1, "XXX", OpenMode.Output
    Write 1, A, B, C
    FileClose 1
End Sub
```

Year Function

Syntax Year(*dateexpr*)

Group Time/Date

Description Return the year.

Parameter	Description
<i>dateexpr</i>	Return the year for this date value.'

See Also [Date\(\)](#), [Day\(\)](#), [Month\(\)](#), [Weekday\(\)](#).

Example

```
'#Language "WWB.NET"
Sub Main
    Debug.Print Year(#1/1/1900#) ' 1900
    Debug.Print Year(#1/1/2000#) ' 2000
End Sub
```

Objects Overview

Each object supports a particular set of *methods* and *properties*. Each method/property has zero or more parameters. Parameters may be optional, in which case the parameter can be specified by using name := value.

- *objexpr.method*[*expr*][, ...] [*param* := *expr*][, ...]
Call *method* for *objexpr*.
- *objexpr.method*[([*expr*][, ...] [*param* := *expr*][, ...])]
Return the value of *method* for *objexpr*.
- *objexpr.property*[([*expr*][, ...] [*param* := *expr*][, ...])]
Return the value of *property* for *objexpr*.
- *objexpr.property*[([*expr*][, ...])] = *expr*
Assign the value of *property* for *objexpr*.
- Set *objexpr.property*[([*expr*][, ...])] = *objexpr*
Set the object reference of *property* for *objexpr*.

Note: `objexpr!name` is short hand for `objexpr.defaultproperty("name")`. Use `objexpr![name]` if name contains any characters that are not allowed in an identifier.

Error List

The following table lists all error codes with the associated error text.

Error	Description
10000	Execution interrupted.
10001	Out of memory.
10008	Invalid '#Uses "module" comment.
10009	Invalid '#Uses module dependency.
10010	Macro is already running.
10011	Can't allocate memory to macro/module.
10012	Macro/module has syntax errors.
10013	Macro/module does not exist.
10014	Another macro is paused and can't continue at this time.
10017	No macro is currently active.
10018	Sub/Function does not exist.
10019	Wrong number of parameters.
10021	Can't allocate large array.
10022	Array is not dimensioned.
10023	Array index out of range.
10024	Array lower bound is larger than upper bound.
10025	Array has a different number of indexes.
10030	User dialog has not been defined.
10031	User pressed cancel.
10032	User dialog item id is out of range.
10033	No UserDialog is currently displayed.
10034	Current UserDialog is inaccessible.
10035	Wrong with, don't GoTo into or out of With blocks.
10040	Module could not be loaded.
10041	Function not found in module.
10048	File not opened with read access.
10049	File not opened with write access.
10050	Record length exceeded.
10051	Could not open file.
10052	File is not open.
10053	Attempt to read past end-of-file.
10054	Expecting a stream number in the range 1 to 511.
10055	Input does not match var type.
10056	Expecting a length in the range 1 to 32767.
10057	Stream number is already open.
10058	File opened in the wrong mode for this operation.
10059	Error occurred during file operation.
10060	Expression has an invalid floating point operation.
10061	Divide by zero.
10062	Overflow.

10063	Expression underflowed minimum representation.
10064	Expression loss of precision in representation.
10069	String value is not a valid number.
10071	Resume can only be used in an On Error handler.
10075	Null value can't be used here.
10080	Type mismatch.
10081	Type mismatch for parameter #1.
10082	Type mismatch for parameter #2.
10083	Type mismatch for parameter #3.
10084	Type mismatch for parameter #4.
10085	Type mismatch for parameter #5.
10086	Type mismatch for parameter #6.
10087	Type mismatch for parameter #7.
10088	Type mismatch for parameter #8.
10089	Type mismatch for parameter #9.
10090	OLE Automation error.
10091	OLE Automation: no such property or method.
10092	OLE Automation: server cannot create object.
10093	OLE Automation: server cannot load file.
10094	OLE Automation: Object var is 'Nothing'.
10095	OLE Automation: server could not be found.
10096	OLE Automation: no object currently active.
10097	OLE Automation: wrong number of parameters.
10098	OLE Automation: bad index.
10099	OLE Automation: no such named parameter.
10100	Directory could not be found.
10101	File could not be killed.
10102	Directory could not be created.
10103	File could not be renamed.
10104	Directory could not be removed.
10105	Drive not found.
10106	Source file could not be opened.
10107	Destination file could not be created.
10108	Source file could not be completely read.
10109	Destination file could not be completely written.
10110	Missing close brace '}'.
10111	Invalid key name.
10112	Missing close paren ')'.
10113	Missing close bracket ']'.
10114	Missing comma ','.
10115	Missing semi-colon ';'.
10116	SendKeys couldn't install the Windows journal playback hook.
10119	String too long (too many keys).
10120	Window could not be found.
10130	DDE is not available.
10131	Too many simultaneous DDE conversations.
10132	Invalid channel number.
10133	DDE operation did not complete in time.
10134	DDE server died.
10135	DDE operation failed.

10140	Can't access the clipboard.
10150	Window style must be in the range from 1 to 9.
10151	Shell failed.
10160	Declare is not implemented.
10200	Basic is halted due to an unrecoverable error condition.
10201	Basic is busy and can't provide the requested service.
10202	Basic call failed.
10203	Handler property: prototype specification is invalid.
10204	Handler is already in use.

Terms

arglist	<code>[expr param:=expr][, ...]</code>
	A list of zero or more <i>exprs</i> that are assigned to the parameters of the <i>procedure</i> .
	<ul style="list-style-type: none"> • A positional parameter may be skipped by omitting the expression. Only optional parameters may be skipped. • Positional parameter assignment is done with <i>expr</i>. Each parameter is assigned in turn. By name parameter assignment may follow. • By name parameter assignment is done with <i>param:=expr</i>. All following parameters must be assigned by name.
arrayvar	A variable that holds an array of values. A <i>Object</i> variable can hold an array. Dynamic arrays can be ReDim ensioned.
As [New] type	<p>As type -or-</p> <p>As New <i>objtype</i>[(Of <i>objtype</i>[, ...])][(Param[,...])]</p> <p>Dim, Private, Public and Static statements may declare variable types using As type or As New <i>objtype</i>. A variable declared using As New <i>objtype</i> is created using a list of zero or more <i>params</i>.</p>
As type	Variable and parameter types, as well as, function and property results may be specified using As type: Boolean, Byte, Char, Date, Decimal, Double, Integer, Long, Object, SByte, Short, Single, String, UInteger, ULong, UShort, objtype, user delegate, user dialog, user enum, user structure .
attribute	A file attribute is zero or more of the following values added together.

Attribute	Value	Description
vbNormal	0	Normal file.
vbReadOnly	1	Read-only file.
vbHidden	2	Hidden file.
vbSystem	4	System file.
vbVolume	8	Volume label.
vbDirectory	16	MS-DOS directory.
vbArchive	32	File has changes since last backup.
vbAlias	64	File is an alias.

big-endian	Multiple byte data values (not strings) are stored with the highest order byte first. For example, the long integer &H01020304 is stored as this sequence of four bytes: &H01, &H02, &H03 and &H04. A Binary or Random file written using Put uses <i>little-endian</i> format so that it can be read using Get on any machine. (Big-endian machines, like the Power-PC, reverse the bytes as they are read by Get or written by Put .)
bytearray	A variable that holds an array of byte values.
caseexpr	An expression which specifies a single value or a range. Refer to the Select Case statement.
charlist	A group of one or more characters enclosed by [] as part of Like operator's right string expression. <ul style="list-style-type: none"> • This list contains single characters and/or character ranges which describe the characters in the list. • A range of characters is indicated with a hyphen (-) between two characters. The first character must be ordinally less than or equal to the second character. • Special pattern characters like ?, *, # and [can be matched as literal characters. • The] character can not be part of charlist, but it can be part of the pattern outside the charlist.
condexpr	An expression that returns a numeric result. If the result is zero then the conditional is False . If the result is non-zero then the conditional is True . <pre>0 'false -1 'true X > 20 'true if X is greater than 20 ' = "hello" 'true if ' equals "hello"</pre>
dateexpr	An expression that returns a date result. Use #literal-date# to express a date value. <pre>#1/1/2000# ' Jan 1, 2000 Now+7 ' seven days from now DateSerial(Year(Now)+1,Month(Now),Day(Now)) ' one year from now</pre>
dialogfunc	A dialog function executes while a <i>user dialog</i> is visible.
dimension	[lower To] upper
	Array dimension.
dlgvar	A dialog variable holds values for fields in the dialog. Dialog variables are declared using Dim dlgvar As user dialog .
expr	An simple or complex expression that returns the appropriate result. <ul style="list-style-type: none"> • Simple: <i>var, cond, date, num, str, obj, field, method, function</i> (result) or property (result). • Complex: One or more simple expressions with parentheses and Operators.

field	Use .field to access individual fields in a dialog variable. dlg.LastName dlg.ZipCode
initialvalue	Initial value for a variable. Use { <i>expr</i> , ... } to create an array value.
instruction	A single command. Beep Debug.Print "Hello" Today = Date Multiple instructions may be used instead of a single instruction by separating the single instructions with colons. X = 1:Debug.Print X If X = 1 Then Debug.Print "X=";X:Stop Beep ' must resume from Stop to get to here
label	An identifier that <i>names</i> a statement. Identifiers start with a letter. Following chars may be a letter, an underscore or a digit.
little-endian	Multiple byte data values (not strings) are stored with the lowest order byte first. For example, the long integer &H01020304 is stored as this sequence of four bytes: &H04, &H03, &H02 and &H01. A Binary or Random file written using Put uses little-endian format so that it can be read using Get on any machine. (<i>Big-endian</i> machines, like the Power-PC, reverse the bytes as they are read by Get or written by Put .)
macro	A macro is like an application. Execution starts at the macro's Sub Main .
method	An object provides methods and <i>properties</i> . Methods can be called as subs (the return value is ignored), or used as functions (the return value is used). If the method name contains characters that are not legal in a <i>name</i> , surround the method name with []. App.[Title\$]
module	A file with public symbols that are accessible by other modules/ <i>macros</i> via the '# Uses special comment. <ul style="list-style-type: none"> • A module is loaded on demand. • A code module is a code library. • An object module or class module implements an object. • A module may also access other modules with its own '#Uses special comments.
name	An identifier that names a variable or a user defined <i>procedure</i> . Identifiers start with a letter. Following chars may be a letter, an underscore or a digit. Count DaysTill2000 Get_Data
num	An expression that returns a numeric result. Use &O to express an octal number. Use &H to express a hex number. The interpretation &H and &O is affected by the

'#Language setting. Specific types of numbers can be indicated by using one of the endings shown in the table below:

Ending	Description
S	The number is a Short value.
I or %	The number is a Integer value. The interpretation of the type is controlled by the '#Language setting.
L or &	The number is a Long value. The interpretation of the type is controlled by the '#Language setting.
US	The number is a UShort value.
UI	The number is a UInteger value. The interpretation of the type is controlled by the '#Language setting.
UL	The number is a ULong value. The interpretation of the type is controlled by the '#Language setting.
D	The number is a Decimal value.
F or !	The number is a Single value.
R or #	The number is a Double value.
@	The number is a Decimal value. The interpretation of the type is controlled by the '#Language setting.

numstr	An expression that returns a <i>numeric</i> or <i>string</i> result.
numvar	A variable that holds one numeric value.
objexpr	A expression that returns a reference to an object or <i>module</i> . CreateObject("WinWrap.CDemoApplication")
objtype	A specific type defined by your application, another application or by an object module or class module .
objvar	A variable that holds a <i>objexpr</i> which references an object. Object variables are declared using As <i>Object</i> in a Dim , Private or Public statement.
param	[[Optional] [ByVal ByRef] [ByVal] ParamArray] param[type][()] [As type] [= <i>defaultvalue</i>]

The *param* receives the value of the associated expression in the **Declare**, **Sub**, **Function** or **Property** call. (See *arglist*.)

- An Optional *param* may be omitted from the call. It must also have a *defaultvalue*. The parameter receives the defaultvalue if a value is not specified by the call.
- All parameters following an Optional parameter must also be Optional.
- ParamArray may be used on the final *param*. It must be an array of **Object** type. It must not follow any Optional parameters. The ParamArray receives all the expressions at the end of the call as an array. If **LBound**(*param*) > **UBound**(*param*) then the ParamArray didn't receive any expressions.
- If the *param* is not ByVal and the expression is merely a variable then the *param* is a reference to that variable (ByRef). (Changing *param* changes the variable.) Otherwise, the parameter variable is local to the *procedure*, so changing its value does not affect the caller.

- Use `param()` to specify an array parameter. An array parameter must be referenced and can not be passed by value. The bounds of the parameter array are available via **LBound()** and **UBound()**.

precedence When several operators are used in an expression, each operator is evaluated in a predetermined order. Operators are evaluated in this order:

- `^` (power)
- `-` (negate)
- `*` (multiply), `/` (divide)
- `\` (integer divide)
- Mod (integer remainder)
- `+` (add), `-` (difference)
- `<<` (shift left), `>>` (shift right)
- `&` (string concatenate)
- `=` (equal), `<>` (not equal), `<` (less than) `>` (greater than), `<=` (less than or equal to), `>=` (greater than or equal to), **Like**, (string similarity) **New**, (object creation) **TypeOf**, (object type) **Is**, (object equivalence) **IsNot** (object non-equivalence)
- Not (bitwise invert)
- And (bitwise and), **AndAlso** (short-circuit logical and)
- Or (bitwise or), **OrElse** (short-circuit logical or)
- Xor (bitwise exclusive-or)

Operators shown on the same line are evaluated from left to right.

procedure A **subroutine**, **function** or **property**.

property An object provides *methods* and properties. Properties may be used as values (like a function call) or changed (using assignment syntax).

If the property name contains characters that are not legal in a *name*, surround the property name with `[]`.

`App.[Title$]`

statement Zero or more *instructions*. A statement is at least one line long. **Begin Dialog**, **Do**, **For**, **If** (multiline), **Select Case**, **While** and **With** statements are always more than one line long. A single line statement continues on the next line if it ends a line with a space and an underscore `'_'`.

```
' = "This long string is easier to read, " + _
  "if it is broken across two lines."
```

`Debug.Print '`

str An expression that returns a string result.

```
"Hello"
'
'+ " Goodbye"
'& " Goodbye"
Mi',2)
```

A string constant may have a 'C' suffix

' = "x"C

The 'C' suffix indicates that the string is one character long.

strarray A variable that holds an array of string values.

streamnum An expression that returns a numeric result. Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.

strvar A variable that holds one string value.

FirstNam'

type Variable and parameter types, as well as, function and property results may be specified using a type character as the last character in their name.

Type char	As Type
%	Integer , The interpretation of the type is controlled by the '#Language setting.
&	Long , The interpretation of the type is controlled by the '#Language setting.
!	Single
#	Double
@	Decimal , The interpretation of the type is controlled by the '#Language setting.
\$	String

user delegate User defined delegates are defined with **Delegate**.

user dialog User defined dialogs are defined with **Begin Dialog**.

user enum User defined enums are defined with **Enum**.

user structure User defined structures are defined with **Structure**.

userstructurevar A user defined structure variable holds values for elements of the user defined structure. User defined structures are defined using **Structure**.

- Declare with **Dim**, **Private**, **Public** or **Static**.
- Declare as a parameter of **Sub**, **Function** or **Property** definition.

var A variable holds either a string, a numeric value or an array of values depending on its type.

vardeclaration *name[type][([dimension[, ...]])][As [New] type]*

The *name* declares a variable.

variantvar A variant variable can hold any type of value or it can hold an array.