RIS d.o.o.
Address: Pilepcic 10, 51215 Kastav, Croatia
Phone: +385 51 687 500, Fax:+385 51 687 501
e-mail: info@ris.hr, web: http://www.ris.hr

Last Update: August 2015

Thin@ Programmer's Guide V3.2

Introduction

Welcome to Thin@ Programmers Guide!

On the following pages you'll first learn about the Thin@ template options, followed by a chapter on Thin@ functions, methods and properties that the Thin@ template automatically adds to your application.

After that, there is a long chapter listing all Thin@ methods and properties that you can manually add to your application to perform specific tasks.

You'll also find out how to add Thin@ to a 100% hand-coded application, how to tweak the Thin@ NetClient application and how to run a local Thin@ test environment.

Contents

I. List of unsupported standard Clarion features in this version (3.1)

1. Browse Grid (Browse is supported).
2. STD procedure calls (for example STD:Close). Use POST(Event:CloseWindow) instead.
3. Business rules.
4. Multiple reports opened at the same time.

Features that are not recommended:

- Date and time display on menu frames not recommended due to unnecessary Client-Server communication

II. Thin@ Template

ThinNET is the name of the template shipped with Thin@ which adds Thin@-specific lines of code to a Clarion application thus making it Thin@-ready.
Some Thin@ features can be customized by the Thin@ template options.

This chapter gives an overview of Thin@ template options and code that gets generated by the template and also provides a sample Clarion application with added Thin@ support (that is automatically generated by the template).

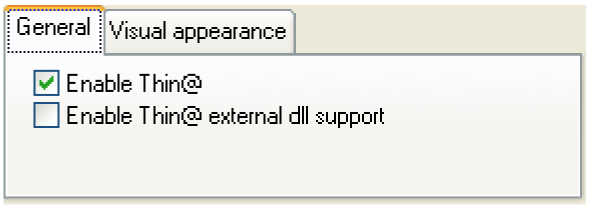
The chapter is divided in 5 parts:

- Thin@ Global Extension options
- Thin@ Procedure Extension options
- Thin@ options for Controls
- Thin@ Code generated in special cases
- Sample Clarion application with Thin@ support

II.1. Thin@ Global Extension options

Thin@ Global Extension options define Thin@ settings that affect the whole application. These options are spread on two tabs: General Tab and Visual appearance Tab.

II.1.1. General Tab



✓ Enable Thin@
☐ Enable Thin@ external dll support

a. Enable Thin@ - Enable/disable Thin@ application support

If checked, the template will generate Thin@-specific lines of code necessary to make an application work as a Thin@ internet application. Although you won't be using much of these function calls in your application (except if you're hand-coding your applications), it's good to know what the template does for you.

a1. <u>Including the Thin@ library in your application</u>

Generated at embed point 'After global includes':

```
INCLUDE('ThinN@.INC')
```

a2. <u>Declaring the Thin@ ThinNetMgr class</u>

Generated at embed point ‘Global data’:

```
ThinNetMgr NetManager
```

a3. Starting Thin@ communication (*)

Generated at embed point ‘Program setup’:

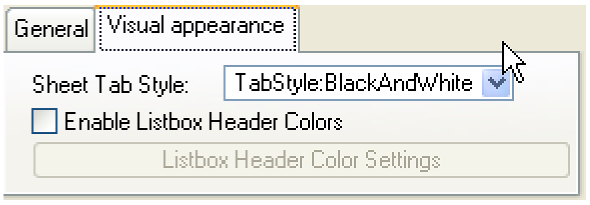
```
! This will start the Thin@ library and wait for the client to respond.
 ThinNetMgr.Start()
```

See also ThinNetMgr.Start.

b. Enable Thin@ external dll support

Enable/disable support for Thin@ External Class.
The Thin@ External Class is used by certain Clarion 3rd Party Vendors to help them add Thin@ support Thin@ to their products.
Leave unchecked unless you know that the 3rd party product that you have in your application is using this feature.

II.1.2. Visual appearance tab



Sheet Tab Style: TabStyle:BlackAndWhite

☐ Enable Listbox Header Colors

On this tab you can modify certain options that affect the visual appearance of your application.

a. SheetTab Style

This is the default style of a SHEET control (visible only in Thin@ Clients compiled in Clarion7 & Clarion8).

Generated code:

```
RisNet:SetDefaultTabStyle(<TabStyleNumber>)
```

See also RisNet:SetDefaultTabStyle.

b. Enable LIST box Header Colors

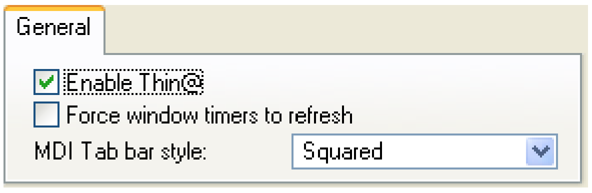
LIST box header color settings (visible only in Thin@ Clients compiled in Clarion7 & Clarion8)

Sample generated code:

```
RisNet:SetDefaultListboxHeaderColors(1, -2,-2,-2,-2,-2,-2,-2,-2)
```

See also RisNet:SetDefaultListboxHeaderColors.

II.2. Thin@ Procedure Extension options



[✓] Enable Thin@
[ ] Force window timers to refresh
MDI Tab bar style: Squared

a. Enable Thin@

Enable/disable Thin@ for the selected procedure.
If checked, the template will generate Thin@-specific lines of code necessary to enable Thin@ on a procedure level. Although you won't be using much of these function calls in your application (except if you're hand-coding your applications), it's good to know what the template does for you.

a1. After opening or closing a window (*)

```
IF ThinNetMgr.Active THEN ThinNetMgr.OpenWindow(<Window>).
IF ThinNetMgr.Active THEN ThinNetMgr.CloseWindow(<Window>).
```

***Note:** Although not required in most cases, ThinNetMgr.OpenWindow and ThinNetMgr.CloseWindow are needed in complex multithreading situations in which Thin@ won't work properly without calling these methods.

a2. Taking application events

Generated at the end of every ACCEPT LOOP:

```
IF ThinNetMgr.Active THEN ThinNetMgr.TakeEvent(<Window>).
```

There is other code that is optionally generated if this option is checked, and it is covered in the fourth section: Thin@ code generated in special cases.

b. Force window timers to refresh

If checked, timer events will cause the window to refresh.

Sample generated code:

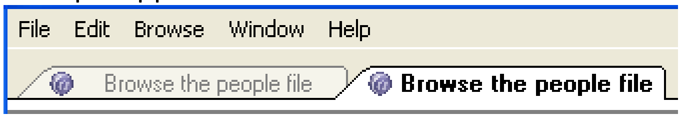
```
ThinNETMgr.AddOption(QuickWindow,0,'forcedtimerrefresh=1')
```

See also ThinNetMgr.AddOption.

c. MDI Tab bar style

Defines whether the MDI tab bar will be turned on or off. If turned on, possible choices are Black and White, Colored, Squared and Boxed.

Example application with MDI Tab Bar turned on:



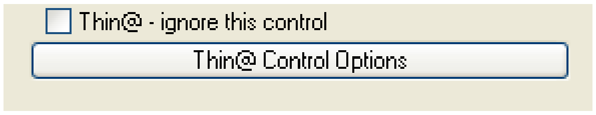
Sample generated code:

```
ThinNETMgr.AddOption(QuickWindow,0,'TabBarStyle=3')
```

See also ThinNetMgr.AddOption.

II.3. Thin@ options for Controls

The Thin@ template also generates lines of code specific to a control type. Some of the control-specific features are customizable through the Actions tab on the control properties.



a. Thin@ - ignore this Control

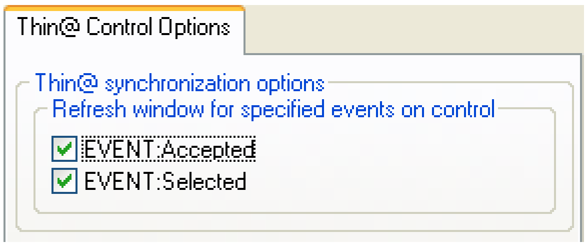
If checked, the control will not be visible in Thin@-mode.

Sample generated code:

```
ThinNETMgr.AddOption(QuickWindow, ?Listbox,'4')
```

See also ThinNetMgr.AddOption.

b. Thin@ synchronization options



Thin@ Control Options

Thin@ synchronization options
Refresh window for specified events on control
[✓] EVENT:Accepted
[✓] EVENT:Selected

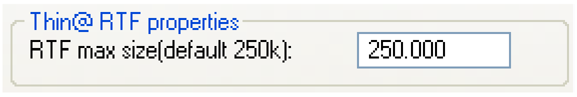
Thin@ synchronization options exist on every Clarion control that can trigger EVENT:Accepted and EVENT:Selected. By default, the Thin@ Client will communicate with the Thin@ Server on every such event. However, it's possible to disable this Thin@ Client-Server communication on EVENT:Accepted and/or EVENT:Selected for a control. This optimization is not necessary, but it can improve performance in certain cases. For example, it can result in quicker transitions from one control to the other when holding down the tab key on a window.

Sample generated code:

```
ThinNETMgr.AddOption(QuickWindow, ?Entry,'3')
```

See also ThinNetMgr.AddOption.

c. Thin@ RTF control properties



RTF max size(default 250k): 250.000

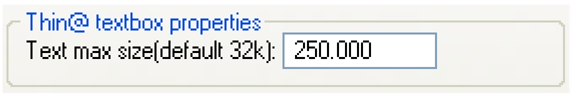
RTF max size (default 250kB) - The maximum size of a RTF control (in bytes) in Thin@-mode.

Sample generated code:

```
ThinNETMgr.AddOption(QuickWindow, ?RTFControl,'Size=250000')
```

See also ThinNetMgr.AddOption.

d. Thin@ textbox properties



Text max size(default 32k): 250.000

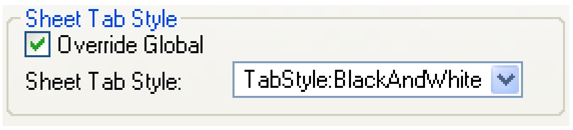
RTF max size (default 32kB) - The maximum size of a RTF control (in bytes) in Thin@-mode.

Sample generated code:

```
ThinNETMgr.AddOption(QuickWindow, ?TextBox,'Size=250000')
```

See also ThinNetMgr.AddOption.

e. Sheet Tab Style



✓ Override Global

Sheet Tab Style: TabStyle:BlackAndWhite

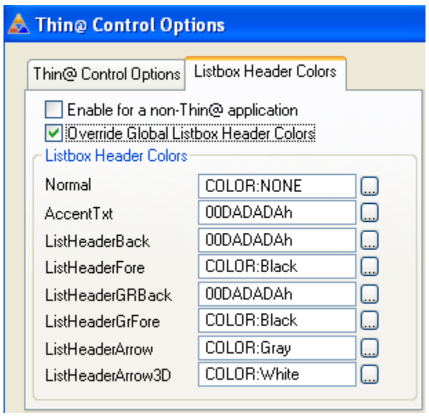
You can override the global tab style settings per control.

Sample generated code:

```
RisNet:SetTabStyle(?QuickWindow, ?Sheet1, 1)
```

See also RisNet:SetTabStyle.

f. ListBox Header Colors


Thin@ Control Options

Thin@ Control Options | Listbox Header Colors

☐ Enable for a non-Thin@ application
☑ Override Global Listbox Header Colors

Listbox Header Colors

Normal	COLOR:NONE
AccentTxt	00DADADAh
ListHeaderBack	00DADADAh
ListHeaderFore	COLOR:Black
ListHeaderGRBack	00DADADAh
ListHeaderGrFore	COLOR:Black
ListHeaderArrow	COLOR:Gray
ListHeaderArrow3D	COLOR:White

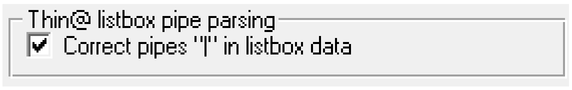
You can override the global LIST box header colors per control.

Sample generated code:

```
RisNet:SetListboxHeaderColors(?QuickWindow,?Listbox,1,-2,-2,-2,-2,-2,-2,-2,-2)
```

See also RisNet:SetListboxHeaderColors.

g. Correct incorrect parsing of pipe (|) character in LIST boxes


Thin@ listbox pipe parsing

☑ Correct pipes "|" in listbox data

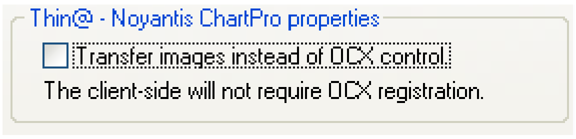
If the data in your LIST boxes contains pipe (|) characters, it’s possible that the data displayed in the LIST box will be messed up. Check this option for every LIST box that manifests that problem.

Sample generated code:

```
ThinNETMgr.AddOption(QuickWindow, ?ListBox,'ParsePipe=1')
```

See also ThinNetMgr.AddOption.

h. Noyantis ChartPro properties


Thin@ - Noyantis ChartPro properties

☐ Transfer images instead of OCX control

The client-side will not require OCX registration.

If you’re using the Noyantis ChartPro ActiveX wrapper template with Thin@, it’s possible to transfer images from the Thin@ Server to the Thin@ Client. In this way the OCX control does not have to be registered on the Thin@ Client computer, but it has to be registered on the Thin@ Server computer.

II.4. Thin@ code generated in special cases

In certain circumstances Thin@ automatically generates other code used to make the application Thin@-ready. Instances of such code are covered in this section.

a. Making Thin@ aware of a LIST box

Automatically generated at WINDOW opening for each WINDOW that contains a LIST box:

```
IF ThinNetMgr.Active THEN
      ThinNetMgr.AddListControl(<Window>,<LISTFeq>,<ListQueueSource>)
END !IF
```

See also ThinNetMgr.AddListControl.

b. Making Thin@ aware of Edit-In-Place

Automatically generated for every LIST box with a Edit-In-Place:

```
    IF ThinNETMgr.Active THEN
        ThinNETMgr.AddListEIPControl(<?ListBox>,<?ListBox{PROP:column}>,0)
    END !IF
```

See also ThinNetMgr.AddListEIPControl.

c. Overriding the default timeout for reports

Automatically generated for every report:

```
    IF ThinNetMgr.Active THEN
        ThinNetMgr.StartLongRunningProcess()
    END
```

See also ThinNetMgr.StartLongRunningProcess.

d. Downloading a report from the Server to the Client side

Sample code automatically generated for every report:

```
IF ThinNetMgr.Active AND SELF.Response = RequestCompleted THEN
   ENDPAGE(SELF.Report)
   ThinNetMgr.EndLongRunningProcess()
   ThinNetMgr.DownloadFiles(SELF.PreviewQueue, SELF.PreviewQueue.FileName,
       'REPORT<-4->'& Report{PROP:Landscape} & '<-5->' & SELF.SkipPreview)
   FREE(SELF.PreviewQueue)
   Previewer.PrintOk = ThinNetMgr.Printed
END
```

See also RisNet:DownloadFiles.

II.5. Sample Clarion application with Thin@ support

This is a typical Clarion application with added Thin@ support. The lines of code which are highlighted are required by Thin@ and are automatically generated by the Thin@ template.

```
PROGRAM

 MAP
Main PROCEDURE
 END

INCLUDE('ThinN@.inc')

ThinNetMgr NetManager

 CODE
 Main

Main PROCEDURE

ListQueue QUEUE
Field1 STRING(100)
Field2 STRING(100)
 END

MyWindow  WINDOW('MyWindow'),SYSTEM,AT(,,225,123),FONT('MS Sans Serif', 8,,
FONT:regular),GRAY,Maximize
PROMPT('From Queue:'), AT(7,9), USE(?Prompt1)
LIST, AT(9,20,97,76), USE(?List1), FORMAT('20L(2)|M'), FROM(ListQueue)
BUTTON('Refresh'), AT(183,102,35,14), USE(?Button)
PROMPT('From String:'), AT(121,9), USE(?Prompt2)
 END

  CODE

 ThinNetMgr.Start()

 OPEN(MyWindow)
IF ThinNETMgr.Active THEN ThinNETMgr.OpenWindow(MyWindow).

 ListQueue.Field1='FirstQueueRow'; ADD(ListQueue)
 ListQueue.Field2='SecondQueueRow'; ADD(ListQueue)

IF ThinNetMgr.Active THEN
    ThinNetMgr.AddListControl(MyWindow, ?List1,ListQueue)
 END

 ACCEPT
    IF EVENT()=EVENT:Accepted AND FIELD()=?Button THEN
        MESSAGE('Hello Word!')
    END

    IF ThinNetMgr.Active THEN
        ThinNetMgr.TakeEvent(MyWindow)
    END
 END

CLOSE(MyWindow)
IF ThinNETMgr.Active THEN ThinNETMgr.CloseWindow(MyWindow).
```

III. Programming functions and methods

This chapter explores the Thin@ functions, methods and properties that you can use in your applications.

The first section of this chapter covers Clarion functions that are replaced by the Thin@ version of the BUILTINS.CLW file.

The second section covers specific situations in which you need to manually add Thin@ calls in order to perform specific actions or make the application work as intended.

III.1. Clarion functions that are overridden by Thin@ version of BUILTINS.CLW

Thin@ ships with a modified version of the BUILTINS.CLW file in which some of the standard Clarion functions are overridden by Thin@ equivalents.

For example, when you call a PRINTERDIALOG() function, the application will in fact execute RisNet:PrinterDialog, which will call the PRINTERDIALOG on the client side.

You can still call the standard Clarion function (which will be executed on the server side), by calling the function with the *_OLD* suffix. For example, PRINTERDIALOG_OLD.

This is a list of Clarion functions that are overridden by the Thin@ version of the BUILTINS.CLW file:

Clarion Function	Thin@ equivalent	Description
START	RisNet:START	Begins a new execution thread.
MESSAGE	RisNet:Message	Displays a message dialog box and returns the button the user pressed
MOUSEX	RisNet:MouseX	Return mouse horizontal position
MOUSEY	RisNet:MouseY	Return mouse vertical position
FILEDIALOG	RisNet:FileDialog	Displays Windows standard file choice dialogs to allow the user to choose a file. *NOTE: when an application is run in the Thin@ mode FILEDIALOG will show client-side files and folders.*
FILEDIALOGA	RisNet:FileDialogA	Displays Windows standard file choice dialogs to allow the user to choose a file. *NOTE: when an application is run in the Thin@ mode FILEDIALOGA will show client-side files and folders.*
COLORDIALOG	RisNet:ColorDialog	Displays the Windows standard color choice dialog box to allow the user to choose a color.
FONTDIALOG	RisNet:FontDialog	Displays the standard Windows font choice dialog box to allow the user to choose a font.
FONTDIALOGA	RisNet:FontDialogA	Displays the standard Windows font choice dialog box to allow the user to choose a font and character set.
PRINTERDIALOG	RisNet:PrinterDialog	Displays the Windows standard printer choice dialog box to allow the user to select or configure a printer.

		NOTE: The standard STD:PrintSetup call cannot be used. Call the PrinterDialog procedure instead. *To do it, write* PrinterDialog *in the ACCEPTED embed point of your Print Setup item.*
SELECT	RisNet:Select	Sets the next control to receive input focus.
POPUP	RisNet:Popup	Returns an integer indicating the user's choice from the menu.
DESTROY	RisNet:Destroy	Removes window controls.
PRESSKEY	RisNet:PressKey	Places one keystroke in the keyboard input buffer.
PRESS	RisNet:Press	Places characters in the keyboard input buffer.
HALT	RisNet:Halt	Immediately terminates the program.
DRAGID	RisNet:DragID	Returns matching host and target signatures on a successful drag-and-drop operation.

III.2. List of Thin@ Functions, Methods and Properties

There are parts in your application which require a Thin@ function call, but the Thin@ template does not generate it automatically (for example a file download/upload).

In those situations you need to manually add Thin@ specific calls in order to make the application function as intended. Those will be covered in this section.

Note that this list does not include Thin@ functions that are automatically overridden by the Thin@ version of BUILTINS.CLW (covered in section III.1), e.g. RisNet:START.

This chapter is divided in 4 parts:

- List of Thin@ Functions
- List of Thin@ Class Methods
- List of Thin@ Functions and Methods related to ActiveX/OLE/OCX
- List of Thin@ Class Properties

III.2.1. List of Thin@ Functions

RisNet:DownloadFile	Downloads a single file from server side to client side.
RisNet:DownloadFiles	Downloads multiple files from server side to client side *(Automatically generated by the Thin@ template for downloading report files.)*
RisNet:UploadFile	Uploads a single file from client side to server side.
RisNet:CancelCloseWindow	Cancels WINDOW termination, even after EVENT:CloseWindow is processed.
RisNet:RunOnClient	Runs an application on the client side.
RisNet:ShowProgressDialog	Displays a progress dialog that is automatically refreshed every 3 seconds.
RisNet:GetClientPath	Returns the PATH on the client.
RisNet:SetClientPath	Sets the PATH on the client.
RisNet:Sleep	Delays program execution for a specified time without using processor time.
RisNet:GetWindowMaxState	Returns 1 if the window is maximized, 0 if it's not.
RisNet:SetWindowMaxState	Setting this to 1 maximizes the window, 0 minimizes the window.
RisNet:SetDefaultListboxHeaderColors	Sets a default header color for all LIST box headers in an application. *(Automatically generated by the Thin@ template based on selected global template options).*
RisNet:SetListboxHeaderColors	Sets header colors for a specific LIST box and overrides default settings *(Automatically generated by the Thin@ template based on selected control template options).*

RisNet:SetDefaultTabStyle	Sets a default tab style for application sheets *(Automatically generated by the Thin@ template based on selected global template options).*
RisNet:SetTabStyle	Sets the tab style for a specific window *(Automatically generated by the Thin@ template based on selected control template options).*
RisNet:RunThinetApplication	Runs a Thin@ application.
RisNet:GroupToXML	Converts a Clarion GROUP to an XML structure which can be easily transported between server and client.
RisNet:XMLToGroup	Fills a Clarion GROUP structure from an XML previously created by GroupToXML.
RisNet:QueueToXML	Converts a Clarion QUEUE to an XML structure which can be easily transported between server and client.
RisNet:XMLToQueue	Fills a Clarion QUEUE structure from an XML previously created by QueueToXML.
RisNet:Exec	Runs a program (similar to Clarion RUN, but with more options).
RisNet:Exists	Checks if a file exists on the Client side.
RisNet:SetTopWindow	Brings a window on top of all other windows.
RisNet:ShellExecute	Wrapper function for the Windows ShellExecute API function.
RisNet:SendMail	Send an email using the default client-side email client *(Embedded into the default print preview procedure).*
RisNet:ShowWindow	Sets the specified window's SHOW state. Wrapper for ShowWindow WinAPI function.
RisNet:CreatePDFfromWMF	Creates an optimized PDF file from a WMF file. *(Embedded into the default print preview procedure).*
RisNet:ScaleWindow	Automatically resizes (to maximum visible area) and scales (enlarges windows font) a window. *(Automatically generated by the Thin@ template based on selected template options).*
RisNet:DisableWindowScaling	Globally disables window resizing and/or scaling. *(Automatically generated by the Thin@ template based on selected template options).*
Added in thin@ 3.1	
RisNet:SetActiveThread	Set's current active thread.
RisNet:TabInsteadEnter	Sets the status of tabInsteadEnter (numeric enter becomes tab) client feature.
RisNet:ExpandComboBox	Force expansion of a combo box.
RisNet:GetResolution	Gets the client resolution in width,height form.
RisNet:PauseNewThreads	Pauses new starting threads.
RisNet:SetClipboard	Sets client side clipboard content.
RisNet:GetClipboard	Gets the client side clipboard contents.

III.2.2. List of Thin@ Class Methods

ThinNet Class Method Name	Description
OpenWindow	Required after Clarion OPEN(<Window>) statement. *(Automatically generated by the Thin@ template).*
CloseWindow	Required before Clarion CLOSE(<Window>) statement. *(Automatically generated by the Thin@ template).*
StartLongRunningProcess	Notifies Thin@ that a long running process (e.g. report) is about to start and disables session timeout. *(Automatically generated by the Thin@ template for reports).*
EndLongRunningProcess	Notifies Thin@ that a long running process (e.g. report) is completed. *(Automatically generated by the Thin@ template for reports).*
TakeEvent	Required by Thin@ at the bottom of every ACCEPT loop to take window events. *(Automatically generated by the Thin@ template).*
AddListControl (QUEUE as the data source)	Required by Thin@ after every LIST box, COMBO box or DROP down list declaration. *(Automatically generated by the Thin@ template).*
AddListControl (string expression as the data source)	Required by Thin@ after every LIST box, COMBO box or DROP down list declaration. *(Automatically generated by the Thin@ template).*
SetClientPrinter	Sets the default printer on the Client
ExecuteClientSource	Executes a piece of code on the Client side. *(NOTE: To execute the appropriate code on the client, you need to modify the Thin@ Client program (NetClient.app)).*
Start	Starts Thin@ communication *(Automatically generated by the Thin@ template).*
DisplayThread	Forces a WINDOW on another thread to refresh (e.g. after POSTing an EVENT to it).
AddImageFolder	Registers a custom subfolder (in the application directory on the server) that contains application resources (images, icons, executables etc.).
AddImageLibrary	Registers a custom archive file (in the application directory on the server) that contains application resources (images, icons, executables etc.). The file can be password protected to protect intellectual property.
Display	Forces immediate window refresh.
AddListEIPControl	This function is used to inform Thin@ that the PROP:EDIT property of a LIST box changed value. Note that the Thin@ template automatically generates this function call. *(Automatically generated by the Thin@ template).*

AddOption	This function can be used to assign various Thin@ options to WINDOW and CONTROL objects.
DisplayControlImage	This function is used to force image refresh of an image file that was modified during runtime.
AddRtfControl	This function is used to inform Thin@ that a WINDOW contains an RTF Control. *(Automatically generated by the Thin@ template).*
AddFileToDownloadList	This function can be used to add a file to the Thin@ download list.
AfterCreatingWindow	VIRTUAL function; derivable. Called on the Client each time a WINDOW is opened.
BeforePaintingControl	VIRTUAL function; derivable. Called before a CONTROL is drawn on the Client side.
AfterPaintingControl	VIRTUAL function; derivable. Called after a CONTROL is drawn on the Client side.
GetControlOption	This function can be used to return Client-side CONTROL properties from the Thin@ library.
TakeClientEvent	VIRTUAL function; derivable. Called for each event generated inside an ACCEPT loop.

III.2.3. List of Thin@ Functions and Methods related to ActiveX/OLE/OCX

Thin@ Function/Method Name	**Description**
ThinNetMgr.AddOCXControl	Used in a Thin@ application instead of the PROP:Create statement.
RisNet:SetOCXProperty	Sets a property of an OCX/OLE control.
RisNet:GetOCXProperty	Gets a property of an OCX/OLE control. **See also** RisNet:OCXRegisterEventProc RisNet:OCXGetLastEventName RisNet:OCXGetParamCount RisNet:OCXGetParam
RisNet:OCXRegisterEventProc	Registers an OCX event callback procedure for the control.
RisNet:OCXGetLastEventName	Gets the name of the last event sent to an OCX control.
RisNet:OCXGetParamCount	Returns the number of parameters associated with the current OCX event.
RisNet:OCXGetParam	This function returns the value a parameter associated with the current OCX event.
RisNet:OCXRegister	Registers the specified OCX control with the client-side OS.
RisNet:OCXBind	This function binds an OCX variable so that it can be used in dynamic expressions with Thin@ OCX functions.
RisNet:OCXAddSkipEvent	This function is used to add an OCX event to the Thin@ OCX Event

	Ignore List in order to improve performance.

III.2.4. List of Thin@ Class Properties

Thin@ Class Property Name	**Description**
Active	Returns 1 if the application is running in Thin@ mode, 0 if not.
ClientPrinter	Default printer on the client side.
State	State of the Thin@ Server.
CurrentThread	Thin@ current thread. It can be different from THREAD().
ThreadBusy	Thread number that is currently communicating with the Client.
ClientPath	Default temporary folder on the Client side.
FilePath	Default temporary folder on the Server side.
FileDirPath	Default temporary folder created on the Server for storing files of each Thin@ user.
Stats	Thin@ user session properties.

III.2.5. Description of Thin@ Functions, Class Methods and Properties

Each Thin@ function, class method and property listed above is described in detail in this section.

1. Downloading a single file

This function downloads a single file from Server side to Client side.

Prototype

```
RisNet:DownloadFile  PROCEDURE(STRING FileName,<STRING Sign>,<STRING
FilePath>,<STRING TargetName>,<*? ChangeCheck>,<*?
ClientDirChoice>),STRING,PROC
```

Parameters

Name	Description
FileName	Full path name on the application server.
<Sign>	You can modify the behavior of the function by using one of these predefined values: *RisNet:FileDialog* - force the Client to open the file dialog and prompt for a download destination. *RisNet:NoDirCreate* - don't create a directory. *RisNet:DirCreate* – create a directory. *RisNet:AskDirCreate* – ask whether to create a directory.
<FilePath>	Default download path on the Client side. If omitted, the files are stored in the default temporary folder on the client side, accessible by the *ThinNetMgr.ClientPath* variable
<TargetName>	You can optionally set a new filename for the downloaded file.
<ChangeCheck>	If 1, Thin@ will check if there is already a file on the Client with the same name, and if there is, compare it with the file which is preparing for download. The file will be downloaded only if there is a difference between the two. Default is 0.
<ClientDirChoice>	The folder on the Client machine selected by the user.

Return value: STRING

If the function succeeds, it returns 0.
If the function fails because the *FilePath* does not exists and *RisNet:DirCreate* is not used (or *RisNet:AskDirCreate* is used and the user picked not to create the directory), the return value is 3.
If the function fails for some other reason, it is a 7z error code. For a list of 7z error codes see http://sevenzip.sourceforge.jp/chm/cmdline/exit_codes.htm.

Usage notes

ThinNetMgr.FileDirPath is the default temporary directory created for storing files of each user on the server side. See RisNet:UploadFile.

Example

The following command will download the file *Filename”* (which is located in the user’s temporary folder on the server) to C:\ on the client. The command will prompt for a download destination.

```
RisNet:DownloadFile(ThinNetMgr.FileDirPath & Filename”,'File<-4->1','C:\')
```

See also

RisNet:DownloadFiles

RisNet:UploadFile

ThinNetMgr.ClientPath

ThinNetMgr.FileDirPath

2. Downloading multiple files

This function downloads multiple files from Server side to Client side.

Prototype

```
RisNet:DownloadFiles PROCEDURE(*QUEUE SourceQueue,*? FileName,STRING
Sign,<STRING FilePath>,<*? ClientDirChoice>),STRING
```

Parameters

Name	**Description**
SourceQueue	The queue name containing multiple files.
Sign	See DownloadFile.
<FilePath>	Default download path on the Client side. If omitted, the files are stored in the default temporary folder on the client side, accessible by the *ThinNetMgr.ClientPath* variable.
<ClientDirChoice>	The folder on the Client machine selected by the user.

Return value: STRING

See RisNet:DownloadFile

Usage notes

Both RisNet:DownloadFile and RisNet:DownloadFiles will compress the files before downloading them to the Client machine.

See also

RisNet:DownloadFile

RisNet:UploadFile

ThinNetMgr.ClientPath

ThinNetMgr.FileDirPath

3. Uploading files

This function uploads a single file from Client to Server side.

Prototype

```
RisNet:UploadFile PROCEDURE(STRING FileName,<STRING FilePath>)
```

Parameters

Name	Description
SourceQueue	The queue name containing multiple files.
Sign	See DownloadFile.
FilePath	The upload path on the Server side. If not supplied the Server will store the uploaded file to a temporary created storage that can be accessed through the *ThinNetMgr.FileDirPath* variable.

Usage notes

The temporary user directory created by Thin@ and stored in the *ThinNetMgr.FileDirPath* variable is not automatically deleted so it's suggested to use the REMOVE command on all the uploaded files and REMOVE(ThinNetMgr.FileDirPath) for removing the temporary directory).

Example

```
RisNet:UploadFile(Filename”, ‘C:\’)
```

See also

RisNet:DownloadFile

RisNet:DownloadFiles

ThinNetMgr.ClientPath

ThinNetMgr.FileDirPath

4. Canceling WINDOW close

This function cancels WINDOW termination, even after EVENT:CloseWindow is processed.

Prototype

```
RisNet:CancelCloseWindow PROCEDURE()
```

Usage notes

Sometimes in programming there is a need to cancel WINDOW termination even after EVENT:CloseWindow is processed. In a standard Client / Server environment it is done by simply calling the CYCLE() function before accept termination. However, in the Thin@ environment you need to call this function before calling CYCLE().

Example

```
ThisWindow.InsertAction PROCEDURE
ReturnValue          BYTE,AUTO

  CODE
  ReturnValue = PARENT.InsertAction()
  ! Thin@ support
  IF ThinNetMgr.Active THEN
      IF ~ReturnValue = Level:Benign THEN RisNet:CancelCloseWindow().
  END
  RETURN ReturnValue
```

5. Running programs on the Client side

This function executes a program on the Client side.
The Thin@ Client ships with a program START.EXE which can be used to start various applications without providing its full path or to open documents by the default program associated with it. Alternatively, you can use RisNet:ShellExecute.

Prototype

```
RisNet:RunOnClient PROCEDURE(? ExecuteString,? WaitFlag,<?
ShowFlag>),LONG,PROC
```

Parameters

Name	Description
ExecuteString	Execute expression.
WaitFlag	If set to 1, waits for an instruction to end. Default is 0.
<ShowFlag>	If set to 1, shows the activated process. Default is 0.

Return value: LONG
If the function succeeds it will return a value which depends on the program from which it was called. For more info about read the following article: http://msdn.microsoft.com/en-us/library/ms683189(v=vs.85).aspx
If the function fails it will return one of the system error codes. For more info about system error codes read the following article: http://msdn.microsoft.com/en-us/library/ms681382(v=vs.85).aspx

Example

```
RisNet:RunOnClient('cmd /c START winword') ! Open a document in word
RisNet:RunOnClient('cmd /c START http://thinetsolution.com') ! Open a webpage
RisNet:RunOnClient('c:\Script.bat') ! Run a batch script
```

See also
RisNet:ShellExecute
ThinNetMgr.ExecuteClientSource
RisNet:RunThinetApplication

6. Showing progress dialog window

This function shows a progress dialog window containing a progress bar that is automatically populated and refreshed every 3 seconds.

Prototype

```
RisNet:ShowProgressDialog PROCEDURE(? pDone,? pTotal,BYTE pCancel=0,<?
pTitle>,<? pFormatExpression>),BYTE
```

Parameters

Name	Description
pDone	Percentage of job done (e.g. 10).
pTotal	Total job percentage (e.g. 100).
pCancel	A value of 1 will show a Cancel button on the window. Default is 0.
<pTitle>	Window title
<pFormatExpression>	Clarion picture @n15

Return value: BYTE
The procedure returns 1 if the process is completed and 0 if it's not completed.

Usage notes
Remember to add this line of code after the loop in case that the Window is closed before the progress bar reaches 100%:

```
IF RisNet:ShowProgressDialog(Processed#,Total#,1,'Processing movies..')=1
THEN BREAK.
```

Example

```
LOOP
      Processed# += 1
      RisNet:ShowProgressDialog(Processed#,Total#,1,'Processing movies..')
END
IF RisNet:ShowProgressDialog(Processed#,Total#,1,'Processing movies..')=1
THEN BREAK.
```

7. Getting the Client folder path

This function returns the current folder path on the Client computer.

Prototype

```
RisNet:GetClientPath PROCEDURE(),STRING
```

Return value: STRING
The function returns the current folder path on the client side computer.

See also
RisNet:SetClientPath

8. Setting the Client folder path

The function sets the current folder path on the Client side computer.

Prototype

```
RisNet:SetClientPath PROCEDURE(STRING PathExpression),STRING
```

Parameters

Name	Description
PathExpression	Folder path

Return value: STRING
If the function succeeds, it returns 0.
If the function fails, it will return the error code 03 – Path not found.

Example

```
RisNet:SetClientPath('C:\')
```

See also
RisNet:GetClientPath

9. Delaying the application without using processor time

This function delays a thread on the Server for a number of milliseconds. It does not use processor time, as would delaying the application using a LOOP.

Prototype

```
RisNet:Sleep PROCEDURE(ULONG Timeout)
```

Parameters

Name	Description
Timeout	Time in miliseconds by which the application execution is delayed.

Usage notes

This function wraps the SLEEP Win API function.

Example

```
RisNet:Sleep(1000)
```

10. Getting maximized state of a window

Use this function to find out if a window is maximized or not (on the Client side).

Prototype

```
RisNet:GetWindowMaxState PROCEDURE(*WINDOW WindowHandle),BYTE
```

Parameters

Name	Description
WindowHandle	The name of the Window object.

Return value
Returns 1 if the window is maximized and 0 if the window is not maximized.

Usage notes
Window maximization works different in Thin@-mode than in a standard Clarion application. In a standard Clarion application, when you press the maximize button on a window (or set PROP:MAXIMIZE attribute to 1), the window maximizes to its maximum size determined by OS resolution.

On the other hand, when an application runs in Thin@-mode and a user maximizes the window, that window is resized on the server to the maximum size determined by her local OS resolution. The application that is running on the server will never be maximized.

Therefore, you can't use the PROP:MAXIMIZE attribute of a window to find out if a window is maximized or to maximize a window.

In Thin@, to find out if a window is maximized you should use RisNet:GetWindowMaxState, and to set window size to maximized, you should use RisNet:SetWindowMaxState.

Example

```
RisNet:GetWindowMaxState(QuickWindow)
```

See also

RisNet:SetWindowMaxState

11. Maximizing a window

Use this function to maximize a window.

Prototype

```
RisNet:SetWindowMaxState PROCEDURE(*WINDOW WindowHandle, BYTE MaxState)
```

Parameters

Name	Description
WindowHandle	Name of the Window object.
MaxState	1 = maximize; 0 = normal

Example

```
!Check if a window is maximized and maximize it if it's not
IF ThinNetMgr.Active THEN
      IF ~RisNet:GetWindowMaxState(MainWin) THEN
      RisNet:SetWindowMaxState(MainWin, 1).
ELSE
    IF MainWin{prop:max}=0 THEN MainWin{prop:max} = 1.
END
```

See also

RisNet:GetWindowMaxState

12. Setting a default header color for application LIST box headers

This function is used to set a default header color for all LIST box headers in an application.

Prototype

```
RisNet:SetDefaultListboxHeaderColors PROCEDURE(BYTE Enabled=1, LONG cNormal =
-2, LONG cAccentTxt = -2, LONG cListHeaderBack = -2, LONG cListHeaderFore = -
2, LONG cListHeaderGRBack = -2, LONG cListHeaderGrFore = -2, LONG
cListHeaderArrow = -2, LONG cListHeaderArrow3D = -2)
```

Parameters

Name	Description
Enabled	Enable(1) or disable(0) ListBox Header Colors. Default value is 1.
cListHeaderBack	Header background color. Default value is -2.
cListHeaderFore	Header foreground (text) color. Default value is -2.
cListHeaderGRBack	Header group background color. Default value is -2.
cListHeaderGrFore	Header group foreground (text) color. Default value is -2.
cListHeaderArrow	Header (sort) arrow color. Default value is -2.
cListHeaderArrow3D	Header (sort) 3D arrow color. Default value is -2.

Usage notes
This property is configurable as a Thin@ template option.

See also

RisNet:SetListboxHeaderColors

13. Setting header color for a LIST box header

This function enables you to set LIST box header colors for a specific LIST box and overrides default settings.

Prototype

```
RisNet:SetListboxHeaderColors PROCEDURE(*Window WindowHandle, LONG Feq, SHORT
Enabled=1, LONG cNormal = -2, LONG cAccentTxt = -2, LONG cListHeaderBack = -
2, LONG cListHeaderFore = -2, LONG cListHeaderGRBack = -2, LONG
cListHeaderGrFore = -2,LONG cListHeaderArrow = -2,LONG cListHeaderArrow3D=-2)
```

Parameters

Name	Description
WindowHandle	Name of the window object.
Feq	FEQ of the LIST box control.
Enabled	Enable(1) or disable(0) LIST box Header Colors. Default value is 1.
cListHeaderBack	Header background color. Default value is -2.
cListHeaderFore	Header foreground (text) color. Default value is -2.
cListHeaderGRBack	Header group background color. Default value is -2.
cListHeaderGrFore	Header group foreground (text) color. Default value is -2.
cListHeaderArrow	Header (sort) arrow color. Default value is -2.
cListHeaderArrow3D	Header (sort) 3D arrow color. Default value is -2.

Usage notes

This property is configurable as a Thin@ template option.

See also

RisNet:SetDefaultListboxHeaderColors

14. Setting a default tab style for SHEET controls

This function sets a default tab style for application sheets.

Property

```
RisNet:SetDefaultTabStyle PROCEDURE(BYTE pTabStyle = 1)
```

Parameters

Name	Description
pTabStyle	A numeric value indicating the style of the tab. Tab style EQUATES: TabStyle:Default EQUATE(0) TabStyle:BlackAndWhite EQUATE(1) TabStyle:Colored EQUATE(2) TabStyle:Boxed EQUATE(3)

Usage notes

In Thin@ it is possible to use the Clarion 7/8 tab style feature even in a Clarion 6 application. The Thin@ client needs to be compiled at least in Clarion 7.
This property is configurable as a Thin@ template option.

See also

RisNet:SetTabStyle

15. Setting a tab style for a specific SHEET control

This function enables you to set the tab style a specific sheet control and override the default settings.

Prototype

```
RisNet:SetTabStyle PROCEDURE(*Window WindowHandle,LONG Feq,BYTE pTabStyle=1)
```

Parameters

Name	Description
Window	Name of the window object.
FEQ	FEQ of the sheet control.
pTabStyle	Numeric value indicating the style of the tab.

Usage notes
This property is configurable as a Thin@ template option.

See also

RisNet:SetDefaultTabStyle

16. Starting a Thin@ application programmatically

This function starts another Thin@ applications from an already running Thin@ application instance.
The application you wish to run must be registered in the Thin@ Main Application Server database. The application will run with the default user credentials.

Prototype

```
RisNet:RunThinetApplication PROCEDURE(STRING ApplicationName)
```

Parameters

Name	Description
ApplicationName	Name of the Thin@ application registered in a Thin@ server that you wish to run

Moving Clarion GROUPs and QUEUEs between client and server

17. Transforming a Clarion GROUP to an XML structure

Prototype

```
RisNet:GroupToXML PROCEDURE(*GROUP pGroup),STRING
```

Parameters

Name	Description
pGroup	Name of a GROUP structure.

Return value: STRING
The function returns a delimited STRING.

See also

RisNet:XMLToGroup
RisNet:QueueToXML
RisNet:XMLToQueue

18. Transforming a properly formatted XML structure to a Clarion GROUP

Prototype

```
RisNet:XMLToGroup PROCEDURE(STRING pString, *GROUP pGroup, <STRING
pIgnoreString>),BYTE
```

Parameters

Name	Description
pString	A properly formatted XML structure (return value from the RisNet:GroupToXML function)
pGroup	Name of a GROUP structure which you wish to populate.
<pIgnoreString>	Name of a variable in a GROUP structure that will be ignored. If you want to ignore multiple variables, separate them with a comma or white space.

Return value: BYTE
Returns 0 if the input string is not properly formatted.
Returns 2 if there is a record in the XML which does not match any GROUP variable by name.
Returns 1 if there is no error.

See also

RisNet:GroupToXML
RisNet:QueueToXML
RisNet:XMLToQueue

19. Transforming a Clarion QUEUE to an XML structure

Prototype

```
RisNet:QueueToXML PROCEDURE(*QUEUE pQueue),STRING
```

Parameters

Name	**Description**
pQueue	Name of a QUEUE structure.

Return value: STRING
The function returns a delimited STRING.

See also

RisNet:GroupToXML
RisNet:XMLToGroup
RisNet:XMLToQueue

20. Transforming a properly formatted XML structure to a Clarion QUEUE

Prototype

```
RisNet:XMLToQueue PROCEDURE(STRING pString, *QUEUE pQueue),BYTE
```

Parameters

Name	Description
pString	A properly formatted XML structure (return value from the RisNet:QueueToXML function)
pQueue	Name of a GROUP structure which you wish to populate.

Return value: BYTE

Returns 0 if the input string is not properly formatted.
Returns 2 if there is a record in the XML which does not match any QUEUE variable by name.
Returns 1 if there is no error.

Example

In the following example we use the RisNet:GroupToXML and RisNet:XMLToGroup functions to move data between Server and Client.
Suppose that you want to populate a Browse Queue with a list of client-side printers.

To do so, you would need to modify the NetClient.app (see ThinNetMgr.ExecuteClientSource). In the appropriate section of the NetClient.app you could use the following code:

```
IF Var1='GetPrinterQ' THEN
! Write your own code here to get a list of client printers, put them in a
Queue, e.g. PrintersQ
RETURN RisNet:QueueToXML(PrintersQ) !This will pass the printer queue back to
the server-side app
END
```

In your application you could use the following code:

```
ClientPrintersQ QUEUE
PrinterQXML CSTRING(255)

CODE
IF ThinNetMgr.Active THEN
      PrinterQXML = ThinNetMgr.ExecuteClientSource('GetPrinterQ')
END !IF

RisNet:XMLToQueue(PrinterQXML, ClientPrintersQ)
```

The ClientPrintersQ should now contain a list of Client-side printers.

See also

RisNet:GroupToXML
RisNet:XMLToGroup
RisNet:QueueToXML

21. Executing a program from your application

This function can be used to execute programs from your application. Programs are executed on the server side. This function is similar to the standard Clarion RUN function, but it offers more options. It uses the CreateProcess API function.

Prototype

```
RisNet:Exec PROCEDURE(STRING pCommand,LONG Wait=0, LONG Show=1, LONG
Timeout=0, <STRING pWindowStation>, <STRING pDesktop>, <STRING
pCurrentDirectory>),LONG
```

Parameters

Name	**Description**
pCommand	Command to be executed.
Wait	If set to 1, the function will wait for the process to finish before continuing. Default is 0.
Show	If set to 1, the function will display the content of the command line window. Default is 1.
Timeout	If the Wait flag is set to 1, this parameter will determine how much time (in milliseconds) is the command going to wait for the process to finish before timing out. The default value of 0 means that it will wait indefinitely.
<pWindowStation>	Win OS WindowStation in which the program is started. Default is the current WindowStation.
<pDesktop>	Win OS Desktop in which the program is started. Default is the current Desktop.
<pCurrentDirectory>	The “Start In” folder in which the program is going to be executed.
<Priority>	The OS priority which will be given to the executing process. Default is 2 (Normal). Possible values are: 0 default (normal) 1 Idle (lowest) 2 Normal 3 High 4 Real time (highest)

Return value: LONG
If the function succeeds, the return value is zero.
If the function fails, the return Error Code is non-zero and is return by the CreateProcess API function.

22. Checking if a file exists on the Client side

This function can be used to check if a file exists on the Client side. Note that you can't use the Clarion EXISTS() function because it checks the existence of the file on the Server side.

Prototype

```
RisNet:Exists PROCEDURE(STRING PathExpression),LONG
```

Parameters

Name	Description
PathExpression	Full path to the file.

Return value: LONG
If the file exists, the return value is 1.
If the file does not exist, the return value is 0.

23. Bringing a window on top of other windows on the Client side

This function can be used to bring a window on top of other windows on the Client side. It uses the BringWindowToTop API function.

Prototype

```
RisNet:SetTopWindow PROCEDURE(STRING WindowCaption),ULONG
```

Parameters

Name	Description
WindowCaption	Caption of the window.

Return value: ULONG
If the function succeeds, the return value is 0.
If the function fails, the return Error Code is non-zero and is returned by the BringWindowToTop API function.

See also
RisNet:ShowWindow

24. Executing/opening files on the Client (with the default associated application)

This is a wrapper function for the ShellExecute Windows API function and can be used to run/open any file on the Client using the default application associated to it.

Prototype

```
RisNet:ShellExecute PROCEDURE(ULONG HWND, STRING pOperation, STRING pFile,
STRING pParameters, STRING pDirectory, ULONG pShowCmd),ULONG
```

Parameters

Please refer to http://msdn.microsoft.com/en-us/library/windows/desktop/bb762153(v=vs.85).aspx for details on input parameters and return values.

See also
RisNet:RunOnClient

25. Sending emails from your application

This function can be used to send emails from the application using the default mailing client. It can send the email both from the Server and from the Client computer.

Prototype

```
RisNet:SendMail PROCEDURE(STRING SendTo, STRING Subject, STRING MessageText,
STRING Attachment, BYTE pSend = 0, BYTE SendFromServer = 0),PROC,LONG
```

Parameters

Name	Description
SendTo	Email Recipient.
Subject	Email subject.
MessageText	Email message text.
Attachment	Email file attachment.
pSend	If set to 1, sends the email immediately. If set to 0, opens the default email client. Default value is 0.
SendFromServer	If set to 1, sends the email from the Server side. If set to 0, sends the email from the Client side. Default value is 0.

Return value: LONG
If the function succeeds, it returns 0.
If the function fails, it return a Win API GetLastError (for more info see http://msdn.microsoft.com/en-us/library/windows/desktop/ms679360(v=vs.85).aspx)

Example

```
RisNet:SendMail('user@domain.com','MessageSubject','Hello','Report.PDF')
```

26.Setting the SHOW state of a specified window

This is a wrapper function for the ShowWindow WinAPI function and can be used to set the show state of a specified window.

Prototype

```
RisNet:ShowWindow PROCEDURE(ULONG HWND, ULONG ShowCmd),BYTE,PROC
```

Parameters

Please refer to http://msdn.microsoft.com/en-us/library/windows/desktop/ms633548(v=vs.85).aspx for details on input parameters and return values.

See also

RisNet:SetTopWindow

27. Creating an optimized PDF file from a WMF file

This function creates an optimized PDF file from a WMF file.

Prototype

```
RisNet:CreatePDFfromWMF PROCEDURE(*QUEUE vSourceQueue, *? vFileName, STRING
PdfFileName)
```

Parameters

Name	Description
vSourceQueue	
vFileName	
PdfFileName	

Example

In this example we call the CreatePDFFromWMF function in the *Before Print Preview* legacy embed instead of calling the default print preview procedure. Therefore, the report will not go to the print preview but it will be converted to a PDF file called OutputPDF.PDF under C:\.

```
IF ThinNetMgr.Active AND SELF.Response = RequestCompleted THEN
        ENDPAGE(SELF.Report)
        ThinNetMgr.EndLongRunningProcess()
        ThinNetMgr.CreatePDFFromWMF(SELF.PreviewQueue,
SELF.PreviewQueue.FileName,'C:\OutputPDF.PDF')
        FREE(SELF.PreviewQueue)
END
```

28. Resizing and/or scaling a window

This function is used to resize and/or scale a window.
Resizing a window in this case means increasing the WIDTH and HEIGHT properties of a window in order to fill the visible area inside the application frame.
Scaling a window in this case means increasing window font in order to better use higher resolutions.

Prototype

```
RisNet:ScaleWindow PROCEDURE(*WINDOW WindowName, BYTE ScaleWindow=1, BYTE
MaxWidth=0, BYTE MaxHeight=0, LONG MarginX=0, LONG MarginY=0)
```

Parameters

Name	Description
WindowName	Name of the WINDOW object.
ScaleWindow	If set to 1, the function will scale the window. Default is 1.
MaxWidth	If set to 1, the function will resize the window to maximum width. Default is 0.
MaxHeight	If set to 1, the function will resize the window to maximum height. Default is 0.
MarginX	If nonzero, the function will resize the window using a fixed left margin. Default is 0.
MarginY	If nonzero, the function will resize the window using a fixed upper margin. Default is 0.

Example

```
RisNet:ScaleWindow(QuickWindow, 1, 1, 1, 0, 0)
```

See also

RisNet:DisableWindowScaling

29. Disabling window scaling

This function can be used to globally disable window resizing and/or scaling.

Prototype

```
RisNet:DisableWindowScaling PROCEDURE(BYTE DisableWindowScaling=0, BYTE
DisableWindowResizing=0)
```

Parameters

Name	**Description**
DisableWindowScaling	If set to 1, window scaling will be globally disabled. Default is 0.
DisableWindowResizing	If set to 1, window resizing will be globally disabled. Default is 0.

See also
RisNet:ScaleWindow

30. Setting currently active thread

This function is used to set current active thread. The call to this function will activate the top window in used thread on the client side.

Prototype

```
RisNet:SetActiveThread PROCEDURE(LONG ThreadNo, BYTE WaitCallingThread =
True, BYTE PostEvent = True)
```

Parameters

Name	**Description**
ThreadNo	Thread id
WaitCallingThread	If set to true, the switch to new thread will happen only after calling thread has finished all the processing and there is no pending events. Default is true.
PostEvent	If set to true, the switch to new thread will post additional event to the switching thread forcing the thread's ACCEPT loop to accepts the transition. Set this to false in case that switching thread does not contain active ACCEPT loop. Default is true.

See also
ThinNetMgr.DisplayThread

31. Setting TabInsteadEnter client side option

This function is used to set current behavior of client TabInsteadEnter option.

Prototype

```
RisNet:TabInsteadEnter PROCEDURE(BYTE Param = 1)
```

Parameters

Name	Description
Param	0 - sets parameter inactive 1 - sets parameter inactive

32. Expanding a combo box

This function is used to force expansion of a ComboBox control.

Prototype

```
RisNet:ExpandComboBox PROCEDURE(LONG Feq, BYTE Clear = False)
```

Parameters

Name	Description
Feq	Feq of the ComboBox control
Clear	If set to true, it will clear previous expansion requests for this control in this unfinished refresh cycle. Default is false.

33. Fetching client resolution

This function is used to fetch client side desktop resolution.

Prototype

```
RisNet:GetResolution PROCEDURE(*? xRes, *? yRes)
```

Parameters

Name	Description
xRes	Variable that will receive horizontal resolution of the client desktop.
yRes	Variable that will receive vertical resolution of the client desktop.

34. Pause new thread

This function is used to pause new starting threads while in thin@ mode. This is useful in scenario where you are doing complex screen drawing and have a need to refresh that temporary data to the client side but you don't want to be distracted with possible new threads and their opening windows.

Prototype

```
RisNet:PauseNewThreads PROCEDURE(LONG MaxThread)
```

Parameters

Name	Description
MaxThread	Contains current maximum thread that is not paused. Any thread that has higher id the MaxThread value will be paused.

See also
ThinNetMgr.DisplayThread

35. Setting a client side clipboard

This function is used to set a clipboard content on the client side.

Prototype

```
RisNet:SetClipboard PROCEDURE(STRING Expression)
```

Parameters

Name	Description
Expression	Contains clipboard expression.

See also
RisNet:GetClipboard

36. Getting a client side clipboard

This function is used to get a clipboard content on the client side.

Prototype

```
RisNet:GetClipboard PROCEDURE,STRING
```

Return value: STRING
The return value contains client side clipboard content.

See also
RisNet:SetClipboard

37. Informing Thin@ that a WINDOW opened

This function should be used after each OPEN(<Window>) statement to inform Thin@ that a WINDOW opened. It is automatically generated by the Thin@ template.

Prototype

```
OpenWindow PROCEDURE(*WINDOW Wind)
```

Parameters

Name	Description
Wind	Name of the WINDOW object.

Usage notes
Using this function after a WINDOW opened is usually not required, but its use is strongly recommended. The application will not work properly in certain multithreading scenarios if this function is not called.

Example

```
OPEN(<Window>)
IF ThinNetMgr.Active THEN ThinNetMgr.OpenWindow(<Window>).
```

See also
ThinNetMgr.CloseWindow

38. Informing Thin@ that a WINDOW closed

This function should be used before each CLOSE(<Window>) statement to inform Thin@ that a WINDOW opened. It is automatically generated by the Thin@ template.

Prototype

```
CloseWindow PROCEDURE(*WINDOW Wind)
```

Parameters

Name	Description
Wind	Name of the WINDOW object.

Usage notes

Using this function before a WINDOW opened is usually not required, but its use is strongly recommended. The application will not work properly in certain multithreading scenarios if this function is not called.

Example

```
IF ThinNetMgr.Active THEN ThinNetMgr.CloseWindow(<Window>).
CLOSE(<Window>)
```

See also

ThinNetMgr.OpenWindow

39. Informing Thin@ that a long running Server job is about to start

Use this function when you want to inform Thin@ that a long-running Server job (such as a long-running report) is about to start, and you expect that its execution is going to last more than 40 seconds.
Note that the Thin@ template automatically generates this function for reports.

Prototype

```
ThinNetMgr.StartLongRunningProcess PROCEDURE()
```

Usage notes

After each action by the application user, the Client sends the request to the Server, which starts to process it. If the Server is unresponsive for longer than 40 seconds (due to a long running process), the Client will break the connection with the Server and the reconnect window will appear.
The Client will reconnect to the Server-side application eventually but only after the long running process has ended.
The solution is using this function to warn the client of an active batch process. It is advisable to use this function with statements which are expected to be executed longer then 40 seconds.
In order to limit the maximum wait time for long running processes, you can use a server side parameter, <long timeout period>. Its default value is set to 1 hour and 30 minutes.
The Thin@ template automatically generates this function for reports.

Example

The following code is used to inform Thin@ that a long running process is going to start before opening a Progress Window.

```
    ! End of "Legacy: After Opening the Window"
    IF ThinNetMgr.Active THEN
        ThinNetMgr.StartLongRunningProcess()
        ThinNetMgr.OpenWindow(ProgressWindow).
    END
```

See also

ThinNetMgr.EndLongRunningProcess

40. Informing Thin@ that a long running Server job is about to end

Use this function when you want to inform Thin@ that a long-running Server job (such as a long-running report) is about to end.
Note that the Thin@ template automatically generates this function call for reports.

Prototype

```
ThinNetMgr.EndLongRunningProcess    PROCEDURE()
```

Usage notes

See ThinNetMgr.StartLongRunningProcess.

Example

In this example the EndLongRunningProcess is called in the ThisWindow.AskPreview method and is used to inform Thin@ that a long running process has ended.
It is called after the report is generated and before it is transferred from the Server to the Client and previewed on the Client.

```
IF ThinNetMgr.Active AND SELF.Response = RequestCompleted THEN
        ENDPAGE(SELF.Report)
        RisNetMgr.EndLongRunningProcess()
        RisNetMgr.DownloadFiles(SELF.PreviewQueue,
SELF.PreviewQueue.FileName,'REPORT<-4->'& Report{PROP:Landscape} & '<-5->' &
SELF.SkipPreview)
END !IF
```

See also

ThinNetMgr.StartLongRunningProcess

41. Informing Thin@ about application events

This function is used to inform Thin@ about application events.
Note that the Thin@ template automatically generates this function call at the end of every ACCEPT loop.

Prototype

```
RisNet:TakeEvent PROCEDURE(*WINDOW Wind)
```

Parameters

Name	Description
Wind	Name of the WINDOW object.

Example

```
OPEN(MyWindow)
ACCEPT
    IF ThinNetMgr.Active THEN !Has to be added in the end of the ACCEPT loop
        ThinNetMgr.TakeEvent(MyWindow)
    END !IF
 END !ACCEPT
CLOSE(MyWindow)
```

42. Informing Thin@ that a WINDOW contains a LIST box (QUEUE as the data source)

This function is used to inform Thin@ that a window contains a LIST box, COMBO box or a DROP down list control.
Note that the Thin@ template automatically generates this function call.

Prototype

```
ThinNetMgr.AddListControl PROCEDURE(*Window WindowHandle,LONG Feq,*QUEUE
ListQueue)
```

Parameters

Name	Description
WindowHandle	Name of the WINDOW object.
Feq	FEQ of the LIST box, COMBO box or a DROP down list control.
ListQueue	Name of the QUEUE.

Usage notes

The Thin@ template automatically generates this function call after the OPEN statement on any WINDOW that contains a LIST box, COMBO box or DROP down list control.
However, if you create the control manually (or using the CREATE statement) or if change the source QUEUE by modifying the PROP:FROM property, you need to inform Thin@ about that.

Example

```
ListQueue QUEUE
Field1 STRING(100)
 END

MyWindow  WINDOW('MyWindow'),SYSTEM,AT(,,225,123)
LIST, AT(9,20,97,76), USE(?List1), FORMAT('20L(2)|M'), FROM(ListQueue)
END

  CODE
 OPEN(MyWindow)

 ListQueue.Field1='FirstQueueRow'; ADD(ListQueue)
 ListQueue.Field2='SecondQueueRow'; ADD(ListQueue)

 !Has to be added after window opening for each ListBox control
 IF ThinNetMgr.Active THEN
    ThinNetMgr.AddListControl(MyWindow, ?List1, ListQueue)
 END !IF
!....
CLOSE(MyWindow)
```

See also

ThinNetMgr.AddListControl (string expression as the data source)
ThinNetMgr.AddListEIPControl

43. Informing Thin@ that a WINDOW contains a LIST box (string expression as the data source)

This function is used to inform Thin@ that a window contains a LIST box, COMBO box or a DROP down list control. Note that the Thin@ template automatically generates this function call.

Prototype

```
ThinNETMgr.AddListControl PROCEDURE(*Window WindowHandle, LONG Feq, ?
DataString)
```

Parameters

Name	Description
WindowHandle	Name of the WINDOW object.
Feq	FEQ of the LIST box, COMBO box or a DROP down list control.
DataString	Source data in form of a string expression.

Usage notes

See ThinNetMgr.AddListControl (QUEUE as the data source).

Example

In this example the function is called twice: the first after the WINDOW is opened, and the second time after the source string of the LIST box is modified.

```
MyWindow  WINDOW('MyWindow'),SYSTEM,AT(,,225,123)
LIST, AT(121,20,97,76), USE(?List2), FROM('FirstStringRow|SecondStringRow')
BUTTON('Refresh'), AT(183,102,35,14), USE(?Button)
END

  CODE
 OPEN(MyWindow)

 !Has to be added after opening the window for every ListBox control
 IF ThinNetMgr.Active THEN
    ThinNetMgr.AddListControl(MyWindow, ?List2, ?List2{PROP:From})
 END !IF

 ACCEPT
    IF EVENT()=EVENT:Accepted AND FIELD()=?Button THEN
        ?List2{PROP:From} = ?List2{Prop:From} & '|ThirdStringRow'
!Has to be called every time you modify the source string
        IF ThinNetMgr.Active THEN
            ThinNetMgr.AddListControl(MyWindow, ?List2, ?List2{PROP:From})
        END !IF
    END !IF

END !ACCEPT
CLOSE(MyWindow)
```

See also

ThinNetMgr.AddListControl (QUEUE as the data source)
ThinNetMgr.AddListEIPControl

44. Setting the default printer on the Client side

This function sets the default printer on the Client side.
Note that the currently set Client printer is readable by the ThinNetMgr.ClientPrinter property.

Prototype

```
ThinNetMgr.SetClientPrinter      PROCEDURE(STRING PrinterName)
```

Parameters

Name	Description
PrinterName	The name of the printer on the Client side.

Example

In this example we call PRINTERDIALOG and use its return value to set the Client printer.

```
PRINTERDIALOG('Choose Printer')
PrinterName" = ThinNetMgr.ClientPrinter
ThinNetMgr.SetClientPrinter = PrinterName"
```

See also

ThinNetMgr.ClientPrinter

45. Executing a code on the Client side

This function executes a piece of code on the Client side.

Prototype

```
ThinNetMgr.ExecuteClientSource   PROCEDURE(STRING Var1,<STRING Var2>,<STRING
Var3>,<STRING Var4>,<STRING Var5>,<STRING Var6>,<STRING Var7>),STRING, PROC
```

Parameters

Name	Description
Var1	General-purpose input string used to pass values from the Server to the Client side. Common use is to specify the exact block of code that you wish to execute on the Client.
<Var2>…<Var7>	Same as above.

Return value: STRING
The function can return any string value from the Client back to the Server.

Usage notes
Before you can call this function in your application, you need to modify the Thin@ Client (NetClient.app) and add the source code that you wish to execute on the Client machine.

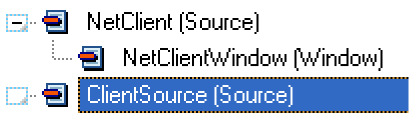
Example 1
In this example we show a STOP message on the Client side.
Note that the STOP message is not executed in the application instance that is running on the Server.

This line of code is added to the (Server-side) application:

```
IF ThinNetMgr.Active THEN
  ThinNetMgr.ExecuteClientSource(‘1’)
END
```

This line of code is added to the ClientSource procedure in the Thin@ client (NetClient.app) application:

```
IF Var1='1' THEN STOP('stop shown').
```



Example 2

In this example we return a value from the Client to the Server.
Suppose that we want to detect the Client's machine screen resolution and send that information to the (Server-side) application.

This line of code is added to the (Server-side) application:

```
IF ThinNetMgr.Active THEN
  ClientScreenResolution = ThinNetMgr.ExecuteClientSource('ScreenResolution')
END
```

This line of code is added to the ClientSource procedure in the Thin@ client (NetClient.app) application:

```
If Var1=' ScreenResolution' THEN
      ! Write your code to detect the client machine screen resolution
RETURN ScreenResolution
END !IF
```

See also

RisNet:RunOnClient

46. Starting Thin@ communication

This function is used to start Thin@ Server-Client communication in an application.
Note that the Thin@ template automatically generates this function call.

Prototype

```
Start                      PROCEDURE(<STRING TemplateVersion>)
```

Parameters

Name	Description
<TemplateVersion>	The version of the Thin@ template which is used to start Thin@. If entered, this value is compared with the version of the Thin@ runtime library (ThinN@.DLL) and if the two numbers are not equal the program will report an error.

Example

```
ThinNetMgr.Start('2.2')
```

47. Forcing a WINDOW running in another thread to refresh

This function is used to force a WINDOW running in another THREAD to refresh, e.g. after posting an EVENT to it.

Prototype

```
ThinNetMgr.DisplayThread PROCEDURE(LONG ThreadNo=0)
```

Parameters

Name	Description
ThreadNo	THREAD() number that contains the WINDOW object that you wish to refresh. Default is 0 (it refreshes all active windows).

Usage notes

Although Thin@ allows POSTing an EVENT to a WINDOW on another THREAD, the results of that EVENT won't be displayed on the other window until it gains focus.
If you want to force refresh of another threaded WINDOW while the focus is still on the current WINDOW, you can accomplish that using this function. It forces refresh of another threaded WINDOW but the focus returns to the current WINDOW.

Important note is that the ThinNetMgr.DisplayThread (or synonym RisNet:DisplayThread()) will force refresh on specific window thread and wait that specific thread is free of all events before sending information to the client side. This means that entire refresh process will be locked until all refreshing threads are done with their processing.

Example

In this example there are two WINDOW objects, each running on its own THREAD.
Window 2 saves its THREAD number after opening:

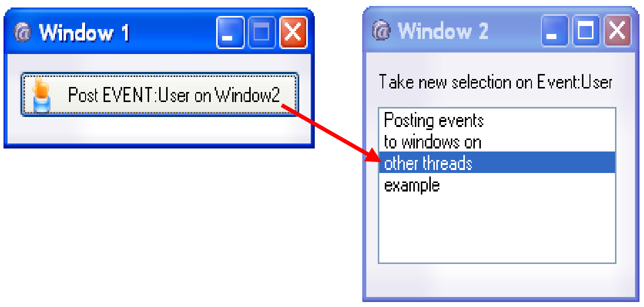
```
! Embed point: After Opening the Window
Winthread[2] = THREAD() ! Save thread 2
```

Window 2 has some code in the EVENT:User embed:

```
! EVENT:User posted; Selecting next item in list...
OF Event:User
 IF choice(?list)=4 THEN Select(?list,1) ELSE Select(?list,choice(?list)+1).
 DISPLAY()
```

Window 1 posts EVENT:User to Window 2 and forces it to DISPLAY, effectively selecting the next item in the list:

```
! Force display thread 2
ThinNetMgr.DisplayThread()
POST(EVENT:User,, WinThread[2])
```



See also

ThinNetMgr.Display

48. Registering a custom resources subfolder (for images, icons etc.)

This function registers a custom subfolder (in the application directory on the Server) that contains application resources (images, icons, executables etc.).

Prototype

```
AddImageFolder PROCEDURE(STRING ImagePath,BYTE OnTop=0)
```

Parameters

Name	Description
ImagePath	Full PATH to the resource folder on the Server.
OnTop	If set to 1, this resource folder will be set on top of the scanning list (possible speed optimization). Default is 0.

Usage notes
In a Thin@ application, all images and icons must be found on the Server and can't be embedded in the application EXE.
Thin@ by default scans the root application folder and the \Images subfolder (if it exists). If you're using an image in your application and it is not found in these folders the image won't display.
However, if you would like to use a custom subfolder for your resource files, you can use this function.

Example
In this example we register a resource folder on the Server in which Thin@ will search for resource files such as images.

```
ThinNETMgr.AddImageFolder('D:\Thin@DemoApp\ImageFolder\')
```

See also
ThinNetMgr.AddImageLibrary

49. Registering a custom resources file (for images, icons, etc.)

This function registers a custom archive file (in the application directory on the Server) that contains application resources (images, icons, executables etc.).
The file can be password protected to protect intellectual property.

Prototype

```
AddImageLibrary PROCEDURE(STRING ImageLibrary,<STRING
ImageLibrarySubdirectory>,<STRING ImageLibraryPassword>,BYTE
ForceDownloadCheck=0)
```

Parameters

Name	Description
ImageLibrary	Full PATH to the resource file on the Server.
<ImageLibrarySubdirectory>	Name of the resource subfolder that is automatically created in the Windows temporary folder on the Server. Resource files are decompressed in this subfolder (e.g. C:\Documents and Settings\<UserName>\Local Settings\Temp\ImageLibrarySubdirectory\)
<ImageLibraryPassword>	Optional password for the resource file.
ForceDownloadCheck	If set to 1, the function will compare files in the ImageLibrary by name and date with files that are already downloaded. If the files on the Client are older than the files on the Server, they will be overwritten. Default is 0.

Usage notes
In some cases you might want to distribute all your resource files inside a resource archive file, and optionally protect it with a password.

Example
This example requires that the ‘Thin.7z’ resource file exists in the specified folder and 7z.exe must be present in the application folder. It will decompress the resource files to the ‘SubFolder’ folder using the password ‘MyPassword’, and Thin@ will continue to use the resource files from that folder.

```
ThinNETMgr.AddImageLibrary('D:\DemoApp\Thin.7z','SubFolder','MyPassword')
```

See also
ThinNetMgr.AddImageFolder

50. Forcing immediate WINDOW refresh

This function forces immediate WINDOW refresh on the Client.

Prototype

```
Display PROCEDURE(<*Window WindowPtr>,BYTE ForceEnd=0, LONG vWindowHandle =
0),BYTE
```

Parameters

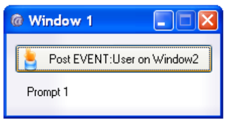
Name	Description
<WindowHandle>	Name of the WINDOW object. If omitted, the current WINDOW is refreshed.
ForceEnd	Internal use. Default is 0.
vWindowHandle	Internal use. Default is 0.

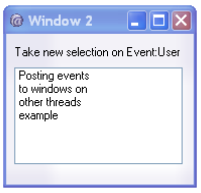
Usage notes

If you want to refresh the current window immediately, without waiting for the end of all window events, you can do it with this function.

Example

This example demonstrates one of the situations where immediate window refresh can be useful. We have two windows, each on its own THREAD.




Window 2
Take new selection on Event:User
Posting events to windows on other threads example

If used in a non-Thin@ application, the following code on the *Window 1* button would a) change the *?Prompt1* text; b) switch thread to *Window 2;* and c) POST Event:User on *Window 2.*

```
?Prompt1{prop:text} = 'Changed text...' !Try to change the text of ?Prompt1
ThinNetMgr.CurrentThread = WinThread[2] !Change current thread to Window 2
POST(Event:User,,WinThread[2]) !Post Event:User on Window 2
```

However, in a Thin@ application the text of *?Prompt1* on *Window 1* would not change because Thin@ would switch the current THREAD to *Window 2* before that (by setting the *ThinNetMgr.CurrentThread* property). In order to force immediate display of *Window 1* before the THREAD is actually switched to *Window 2*, we would modify the code and add the ThinNetMgr.Display function before the switching the THREAD:

```
?Prompt1{prop:text} = 'Changed text...'
ThinNETMgr.Display(Window1) ! Force window display

ThinNetMgr.CurrentThread = WinThread[2]
POST(Event:User,,WinThread[2])
```

See also

ThinNetMgr.DisplayThread

51. Informing Thin@ that the PROP:EDIT property of a LIST box changed

This function is used to inform Thin@ that the PROP:EDIT property of a LIST box changed value. Note that the Thin@ template automatically generates this function call.

Prototype

```
AddListEIPControl      PROCEDURE(LONG pListFeq,LONG Column,LONG ControlFeq)
```

Parameters

Name	Description
pListFeq	FEQ of the LIST box control.
Column	Element number of the PROP:EDIT array which indicates the column number to edit.
ControlFeq	FEQ of the control to perform edit-in-place for a LIST box column.

Usage notes

The Thin@ template automatically generates this function call for the *ResetColumn* and *ClearColumn* class methods of the EIPManager.
However, if you're hand-coding the LIST box control and EIP entry you need to inform Thin@ about any PROP:EDIT assignment that takes place.

Example

```
Win1 WINDOW('List Edit In Place'),AT(0,1,308,172),SYSTEM
   LIST,AT(6,6,120,90),USE(?List),COLUMN,FORMAT('60L@s15@60L@s15@'), FROM(Q)
 END
?EditEntry EQUATE(100)

CODE
OPEN(Win1)
CREATE(?EditEntry,CREATE:Entry)

ACCEPT
 CASE FIELD()
!...
 OF ?EditEntry
   CASE EVENT()
   OF EVENT:Accepted
     PUT(Q)
     ?List{PROP:edit,?List{PROP:column}} = 0
     IF ThinNETMgr.Active THEN
         ThinNETMgr.AddListEIPControl(?List,?List{PROP:column},0)
     END !IF
    END !CASE EVENT
 END !CASE FIELD
END !ACCEPT
```

See also

ThinNetMgr.AddListControl (string expression as the data source)
ThinNetMgr.AddListControl (QUEUE as the data source)

52. Assigning various Thin@ options

This function can be used to assign various Thin@ options to WINDOW and CONTROL objects.

Prototype

```
AddOption   PROCEDURE(*Window WindowHandle,LONG Id,? OptionVar)
```

Parameters

Name	Description
WindowHandle	Name of the WINDOW object.
Id	FEQ of the CONTROL object that the option is assigned to. If set to 0, the option is assigned to the whole WINDOW.
OptionVar	This field can contain an option name and assigned value, in the following format: <Option name><Option value>.

These are the valid option names:

Option name	Applies to	Description
Size=	TEXT CONTROL	Maximum size of the control (number of characters).
ForcedTimerRefresh=	WINDOW	If set to 1, Thin@ will force the TIMER to refresh on a WINDOW
ForcedHide=	WINDOW	If set to 1, the WINDOW will be hidden on the Client side.
TabBarStyle=	SHEET CONTROL	MDI Tab bar style. Possible values are:
TabOrientation=	SHEET CONTROL	Tab orientation.
ParsePipe=	CONTROL	If set to 1, Thin@ will correctly parse to pipe (|) character on a LIST box. Also configurable as a template option.

Value	Description
-1	None
1	Black and white
2	Colored
3 (default)	Squared
4	Boxed

This field can also contain an option value, without a name.

These are the valid option values:

Option value	Description
1	Refresh the CONTROL on EVENT:ACCEPTED
2	Refresh the CONTROL on EVENT:SELECTED
3	Refresh the CONTROL on both EVENT:ACCEPTED and EVENT:SELECTED
4	Hide the CONTROL on the Client.

NOTE: All of these options are configurable through the Thin@ template.

Usage notes

These options are usually configured through the Thin@ template options. Individual use is rare, except in hand-coded applications.

Example

In this example we instruct Thin@ to force the refreshing of the WINDOW timer.

```
ThinNETMgr.AddOption(QuickWindow,0,'forcedtimerrefresh=1')
```

53. Forcing image refresh of an image file that was modified during runtime

This function is used to force image refresh of an image file that was modified during runtime.

Prototype

```
DisplayControlImage    PROCEDURE(LONG Feq)
```

Parameters

Name	Description
Feq	FEQ of the image CONTROL.

Usage notes

If an image file gets modified during runtime, Thin@ normally does not refresh the IMAGE. You can use this function to force image refresh.
It will refresh the IMAGE only if the file was modified.

54. Informing Thin@ that a WINDOW contains an RTF Control

This function is used to inform Thin@ that a WINDOW contains an RTF Control.
Note that the Thin@ template automatically generates this function call.

Prototype

```
AddRtfControl  PROCEDURE(*Window WindowHandle, LONG Feq, *RTFControlClass
RTFCClass)
```

Parameters

Name	Description
WindowHandle	Name of the WINDOW object.
Feq	FEQ of the RTF Control.
RTFControlClass	Object name of the RTF Control Class.

Example
In this typical example the RTF Control is initialized and the required Thin@ function is called to inform Thin@ about the RTF Control.

```
! RTF ?RTFTextBox Initialize
  RTFControl19.Init(?RTFTextBox)

  ! Thin@ - ?RTFTextBox RTF control
  IF ThinNetMgr.Active THEN
      ThinNetMgr.AddRTFControl(Window, ?RTFTextBox, RTFControl19).
  END
```

55. Adding a file to the Thin@ Download List

This function can be used to add a file to the Thin@ download list.

Prototype

```
AddFileToDownloadList PROCEDURE(*? FilePath, BYTE Compression=0, BYTE Recheck
= 0)
```

Parameters

Name	Description
FilePath	Full path to the file on the Server.
Compression	Deprecated.
Recheck	If 1, Thin@ will recheck if the file changed.

Usage notes

This function is implicitly called on every WINDOW that has images and icons, so it's rarely needed in programming. However, it is needed in certain situations. For example, if you're using OCX controls that have images/icons, those images/icons will not be scanned by Thin@ and you have to explicitly call this function.

Example

```
! RTF ?RTFTextBox Initialize
  RTFControl19.Init(?RTFTextBox)

  ! Thin@ - ?RTFTextBox RTF control
  IF ThinNetMgr.Active THEN
      ThinNetMgr.AddRTFControl(Window, ?RTFTextBox, RTFControl19).
  END
```

See also

RisNet:DownloadFile

56. Returning the Thin@ Client-side library state of <CONTROL>{prop} values

This function can be used to return Client-side CONTROL properties from the Thin@ library.

Prototype

```
GetControlOption       PROCEDURE(LONG Feq,STRING pOption),STRING
```

Parameters

Name	Description
Feq	FEQ of the Control.
pOption	Property name.

Return value: STRING
The function returns a property value for the given CONTROL.

Usage notes
With this function you can check a property of a CONTROL on the Client side, even before it is assigned. Rarely used.

III.2.6. Adding support for OLE/OCX/ActiveX controls

a) Creating OLE/OCX/ActiveX controls

This method assigns a PROP:Create statement to the Client-side OLE control.
Use this method in your application instead of the *<OLE Control>{PROP:Create}* statement.

Prototype

```
ThinNetMgr.AddOCXControl PROCEDURE(*Window WindowHandle, LONG Feq, STRING
CreateStatement)
```

Parameters

Name	Description
WindowHandle	Name of the WINDOW object.
Feq	FEQ of the OLE control.
CreateStatement	Name of the OCX file.

Example

```
IF ThinNetMgr.Active THEN
   ThinNetMgr.AddOcxControl(Window,?ChartPro,'Codejock.ChartPro.v15.0.2.ocx')
ELSE
   ?ChartPro{PROP:Create} = 'Codejock.ChartPro.v15.0.2.ocx'
END
```

See also
RisNet:OCXRegister

b) Assigning properties to OLE/OCX/ActiveX controls

This method sets a property of an OCX/OLE control.
Use this method in your application instead of the *<OLE Control>{<Property>}=* statement.

Prototype

```
RisNet:SetOCXProperty PROCEDURE(LONG Feq,STRING Expression,<STRING
SetValue>,<*WINDOW WindowHandle>)
```

Parameters

Name	Description
Feq	FEQ of the OLE control.
Expression	This parameter can take one of the two: - a name of an OCX property - a PROP attribute of the OLE control
<SetValue>	This parameter can be used in two different ways: - a value that you want to assign to the OCX property - a value of the PROP attribute
<WindowHandle>	Name of the WINDOW object which contains the OLE control. If omitted, the current window is used.

Usage notes

This method can be used for two different things:

- To set a property of a Client-side OCX control
- To set a PROP attribute of a Client-side OLE control

Example 1: Setting the value of an OCX property

In this example we set the ‘Value’ property of an OCX control:

```
IF ThinNetMgr.Active THEN
      Risnet:SetOCXProperty(?OCXControl,'Value',FORMAT(TODAY(),@d6))
ELSE
      ?OCXControl{'Value'} = FORMAT(TODAY(),@D6) !Set control to TODAY
END
```

Example 2: Setting an OLE property

In this example we set the PROP:DoVerb attribute of an OLE control:

```
IF ThinNetMgr.Active THEN
      Risnet:SetOCXProperty(?OCXControl, PROP:DoVerb , DOVERB:Primary)
ELSE
      ?OCXControl{PROP:DoVerb} = DOVERB:Primary
END
```

See also

RisNet:GetOCXProperty

c) Retrieving properties from OLE/OCX/ActiveX controls

This method gets a property of an OCX/OLE control.
Use this method in your application instead of checking the value of *<OLE Control>{<Property>}*

Prototype

```
RisNet:GetOCXProperty PROCEDURE(LONG Feq,STRING Property),STRING
```

Parameters

Name	Description
Feq	FEQ of the OLE control.
Property	This parameter can take one of the two: - a name of an OCX property - a PROP attribute of the OLE control
<WindowHandle>	Name of the WINDOW object which contains the OLE control. If omitted, the current window is used.

Return value: STRING

- If the Property parameter is a name of an OCX property then the value of the specified OCX property is returned.
- If the Property attribute is a PROP attribute then the value of the specified PROP attribute is returned.

Example 1: Getting a property of an OCX control
The following example gets the 'Value' property of an OCX control.

```
IF ThinNetMgr.Active THEN
    DateField = RisNet:GetOCXProperty(?OCXControl,'Value')
ELSE
    DateField = ?OCXControl{'Value'}
END
```

Example 2: Getting a PROP attribute of an OLE control
The following example gets PROP:Ctrl attribute of an OLE control.

```
IF ThinNetMgr.Active THEN
    DateField = RisNet:GetOCXProperty(?OCXControl,PROP:Ctrl)
ELSE
    DateField = ?OCXControl{PROP:Ctrl}
END
```

See also
RisNet:SetOCXProperty

d) Registering an OCX callback event procedure

This function installs an OCX event callback procedure for the control.

Prototype

```
RisNet:OCXRegisterEventProc PROCEDURE(LONG Feq,LONG EventProcAddress)
```

Parameters

Name	Description
Feq	FEQ of the OLE control.
EventProcAddress	Memory address of the event processing callback procedure for the control.

Usage notes
The callback procedure is called whenever an event is posted by the operating system to the control. It should be used in Thin@ mode instead of the OCXREGISTEREVENTPROC method.

Example

```
!Register Event processing Callback
IF ThinNetMgr.Active THEN
      RisNet:OCXRegisterEventProc(?OcxObject,ADDRESS(EventFunc))
ELSE
      OCXREGISTEREVENTPROC(?OcxObject,EventFunc)
END
```

See also
RisNet:OCXGetLastEventName
RisNet:OCXGetParamCount
RisNet:OCXGetParam

e) Getting the name of the last event sent to an OCX control

The function returns the last EVENT name sent to the OCX control.

Prototype

```
RisNet:OCXGetLastEventName PROCEDURE(SHORT Reference),STRING
```

Parameters

Name	Description
Reference	The label of the first parameter of the event processing callback procedure.

Return value: STRING
The last EVENT name sent to the OCX control is returned.

Example

```
IF ThinNetMgr.Active THEN
      Res = 'Event: ' & RisNet:OCXGetLastEventName(Reference)
ELSE
      Res = 'Event: ' & OleControl{PROP:LastEventName}
END
```

See also
RisNet:OCXRegisterEventProc
RisNet:OCXGetParamCount
RisNet:OCXGetParam

f) Returning the number of parameters associated with the current OCX event

This function returns the number of parameters associated with the current OCX event.

Prototype

```
RisNet:OCXGetParamCount PROCEDURE(SHORT Reference),STRING
```

Parameters

Name	Description
Reference	The label of the first parameter of the event processing callback procedure.

Return value: STRING
The number of parameters associated with the current OCX event is returned.

Usage notes
This procedure is only valid when the OCX event processing callback function is active**.** Thin@ equivalent of OCXGETPARAMCOUNT.

Example

```
!Cycle through all parameters
IF ThinNetMgr.Active THEN
      LOOP Count = 1 TO RisNet:OCXGETPARAMCOUNT(Reference).
ELSE
      LOOP Count = 1 TO OCXGETPARAMCOUNT(Reference).
END
```

See also
RisNet:OCXRegisterEventProc
RisNet:OCXGetLastEventName
RisNet:OCXGetParam

g) Returning the value of a parameter associated with the current OCX event

This function returns the value of a parameter associated with the current OCX event.

Prototype

```
RisNet:OCXGetParam PROCEDURE(SHORT Reference,LONG Count),STRING
```

Parameters

Name	Description
Reference	The label of the first parameter of the event processing callback procedure.
Count	The number of the parameter to retrieve.

Return value
The value a parameter associated with the current OCX event is returned.

Usage notes
This procedure is only valid when the OCX event processing callback function is active. Thin@ equivalent of OCXGETPARAM.

Example

```
IF ThinNetMgr.Active THEN
      Parm = RisNet:OCXGETPARAM(Reference,Count)
ELSE
      Parm = OCXGETPARAM(Reference,Count)
END
```

See also
RisNet:OCXRegisterEventProc
RisNet:OCXGetLastEventName
RisNet:OCXGetParamCount

h) Registering an OCX control with the Client OS

This function registers the specified OCX control with the Client-side OS.

Prototype

```
RisNet:OCXRegister PROCEDURE(STRING FileName,BYTE ForceRegistration = 0)
```

Parameters

Name	Description
FileName	Full path to the OCX file you want to register.
ForceRegistration	If set to 1 it will attempt to register the OCX control even if it is already registered. Default is 0.

Usage notes
The registration procedure will first check if the OCX control is registered on the client side.
If the OCX control is not registered on the Client, or the ForceRegistration parameter is set to 1, the procedure will attempt to register the OCX file on the Client following these steps:

1. It will first check for the existence of the OCX file on the server in the root application folder, \Resource subfolder and all image folders.
2. If the file is found on the server and the file does not exist on the client or if the client-side file is older than the server-side file, the procedure will download the new file from the server and register it on the client.
3. If the file on the client is newer it will register the file already on the client.

See also
ThinNetMgr.AddOCXControl

i) Binding an OCX variable

This function binds an OCX variable so that it can be used in dynamic expressions with Thin@ OCX functions.

Prototype

```
RisNet:OCXBind PROCEDURE(STRING BindName, <*? Variable>, <BYTE BindType>)
```

Parameters

Name	Description
BindName	A string constant containing the identifier used in the dynamic expression. This may be the same as *variable*.
Variable	The label of the temporary variable on the Client side.
BindType	If set to 1, the temporary variable on the Client side will be type LONG. If set to 2, the temporary variable on the Client side will be type STRING.

Usage notes
See Clarion help on the BIND function for more information.

j) Ignoring an OCX event

This function is used to add an OCX event to the Thin@ OCX Event Ignore List in order to improve performance.

Prototype

```
RisNet:OCXAddSkipEvent PROCEDURE(LONG Feq, LONG pEvent=0, <STRING
EventExpression>,<*WINDOW WindowHandle>)
```

Parameters

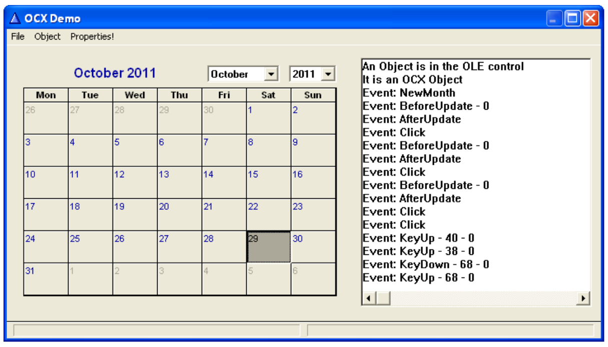
Name	Description
Feq	FEQ of the OLE control.
pEvent	Event number. Default is 0.
<EventExpression>	If set to 1, the temporary variable on the Client side will be type LONG. If set to 2, the temporary variable on the Client side will be type STRING.
<WindowHandle>	Name of the WINDOW object. If omitted, the CONTROL FEQ must be found on the current window.

Usage notes
Sometimes an OCX control performs events that trigger frequent Client-Server interactions but that are not essential and often serve only minor aesthetic purposes.
However, due to frequent Client-Server interactions these events can have a significant impact on speed and overall performance of the application in a Thin@ environment.
This function can be used to disable the event.

Example
In this example we disable certain events that not essential, but trigger frequent Client-Server interactions.

```
! CommandBars speed optimization
 RisNet:OCXAddSkipEvent(SELF.OCXCtrl, 2, ,NoyantisWind)
 RisNet:OCXAddSkipEvent(SELF.OCXCtrl, 0, 'ControlSelected',NoyantisWind)
 RisNet:OCXAddSkipEvent(SELF.OCXCtrl, 0, 'TrackingModeChanged',NoyantisWind)
 RisNet:OCXAddSkipEvent(SELF.OCXCtrl, 0, 'InitCommandsPopup',NoyantisWind)
 RisNet:OCXAddSkipEvent(SELF.OCXCtrl, 0, 'Update',NoyantisWind))
```

III.2.7. Sample application with OCX



```
An Object is in the OLE control
It is an OCX Object
Event: NewMonth
Event: BeforeUpdate - 0
Event: AfterUpdate
Event: Click
Event: BeforeUpdate - 0
Event: AfterUpdate
Event: Click
Event: BeforeUpdate - 0
Event: AfterUpdate
Event: Click
Event: Click
Event: KeyUp - 40 - 0
Event: KeyUp - 38 - 0
Event: KeyDown - 68 - 0
Event: KeyUp - 68 - 0
```

The following example application demonstrates Thin@ support for the Calendar OCX control. It shows OCX control registration, initialization, setting parameters to and getting parameters from the OCX control and adding an event processing callback function.
The lines of code specific to a Thin@ implementation are highlighted.

```
! This program uses the Calendar OCX that Microsoft ships with its Access95
! product (specifically, the one in MS Office Professional for Windows 95).

 PROGRAM

 MAP
 INCLUDE('OCX.CLW')
EventFunc  PROCEDURE(*SHORT Reference,SIGNED OleControl,LONG
CurrentEvent),LONG
 END
 INCLUDE('OCXEVENT.CLW') !Constants that OCX events use
 INCLUDE('ERRORS.CLW') !Include errorcode constants
 INCLUDE('ThinN@.inc') !Include Thin@ methods and properties

ThinNetMgr NetManager !Thin@ class

!Event and change display queue
GlobalQue QUEUE
F1         STRING(255)
          END

SaveDate FILE,DRIVER('TopSpeed'),PRE(SAV),CREATE
Record    RECORD
DateField  STRING(10)
          END
         END

! Main Window definition
MainWin WINDOW('OCX Demo'),AT(,,360,167),STATUS(-1,-1),SYSTEM,GRAY,MAX,RESIZE
       MENUBAR,USE(?Menubar)
         MENU('&File'),USE(?FileMenu)
           ITEM('Save Date to File'),USE(?SaveObjectValue)
           ITEM('Retrieve Saved Date'),USE(?GetObject)
           ITEM('E&xit'),USE(?exit)
         END
         MENU('&Object'),USE(?MenuObject)
           ITEM('About Box'),USE(?AboutObject)
           ITEM('Set Date to TODAY'),USE(?SetObjectValueToday)
           ITEM('Set Date to 1st of Month'),USE(?SetObjectValueFirst)
         END
         ITEM('&Properties!'),USE(?ActiveObj)
       END
       LIST,AT(215,8,139,150),USE(?List1),HVSCROLL,FROM(GlobalQue)
       OLE,AT(5,8,200,150),USE(?OcxObject)
       END
     END
```

```
CODE
 ThinNetMgr.Start()

 OPEN(SaveDate)
 IF ERRORCODE()                                  !Check for error on Open
  IF ERRORCODE() = NoFileErr                     !if the file doesn't exist
   CREATE(SaveDate)                              !create it
   IF ERRORCODE() THEN HALT(,ERROR()) END
   OPEN(SaveDate)                                !then open it for use
   IF ERRORCODE() THEN HALT(,ERROR()) END
  ELSE
   HALT(,ERROR())
  END
 END
 OPEN(MainWin)

 !Register OCX control on the system
 RisNet:OCXRegister('C:\Program files\Microsoft office\Office12\MSCAL.OCX')

 IF ThinNetMgr.Active THEN
   ThinNetMgr.AddOcxControl(MainWin, ?OcxObject,'MSCAL.Calendar.7')
   ThinNetMgr.AddListControl(MainWin, ?List1, GlobalQue)
 ELSE
   ?OcxObject{PROP:Create} = 'MSCAL.Calendar.7' !MS Access 95 Cal OCX control
 END

 IF RECORDS(SaveDate)                      !Check for existing saved record
  SET(SaveDate)                            !and get it
  NEXT(SaveDate)
  IF ERRORCODE() THEN STOP(ERROR()).
  POST(EVENT:Accepted,?GetObject)
 ELSE
  ADD(SaveDate)                             !or add one
  IF ERRORCODE() THEN STOP(ERROR()).
 END
 IF ?OcxObject{PROP:OLE}                    !Check for an OLE Object
  GlobalQue     = 'An Object is in the OLE control'
  ADD(GlobalQue)

  IF ?OcxObject{PROP:Ctrl}                   !See if Object is an OCX
   GlobalQue     = 'It is an OCX Object'
   ADD(GlobalQue)
  END
 END
 DISPLAY

!Register Event processing Callback
 IF ThinNetMgr.Active THEN
   RisNet:OCXRegisterEventProc(?OcxObject,ADDRESS(EventFunc))
 ELSE
   OCXREGISTEREVENTPROC(?OcxObject,EventFunc)
 END

 ?OcxObject{PROP:ReportException} = 1 !Enable the OCX's error reporting
```

```
ACCEPT
  CASE EVENT()
  OF EVENT:Accepted
   CASE FIELD()
   OF ?Exit
    POST(EVENT:CloseWindow)
   OF ?AboutObject
    ?OcxObject{'AboutBox'}                         !Display control's About Box
   OF ?SetObjectValueToday
    IF ThinNetMgr.Active THEN
      Risnet:SetOCXProperty(?OCXObject,'Value',FORMAT(TODAY(),@d6))
    ELSE
      ?OcxObject{'Value'} = FORMAT(TODAY(),@D6)   !Set control to TODAY's date
    END
   OF ?SetObjectValueFirst
    IF ThinNetMgr.Active THEN
    ELSE
      ?OcxObject{'Value'} = MONTH(TODAY()) & '/1/' & SUB(YEAR(TODAY()),3,2)
    END
   OF ?SaveObjectValue                           !Save control's value to file
    IF ThinNetMgr.Active THEN
       SAV:DateField = RisNet:GetOCXProperty(?OcxObject,'Value')
    ELSE
      SAV:DateField = ?OcxObject{'Value'}
    END
    PUT(SaveDate)
    IF ERRORCODE() THEN STOP(ERROR()).

   OF ?GetObject                                !Set control's value from file
    IF ThinNetMgr.Active THEN
    ELSE
      ?OcxObject{'Value'} = SAV:DateField
    END
   OF ?ActiveObj
    !?OcxObject{PROP:DoVerb} = 0        !Activate control's property dialog
    ?OcxObject{PROP:DOVERB} = 0         !Activate control's property dialog
   END
  END
  IF ThinNetMgr.Active THEN ThinNetMgr.TakeEvent(MainWin).
 END
```

```
!Event processing callback function
EventFunc    PROCEDURE(*SHORT Reference,SIGNED OleControl,LONG CurrentEvent)
Count        LONG
Res          CSTRING(200)
Parm         CSTRING(30)
 CODE

 IF CurrentEvent <> OCXEVENT:MouseMove           !Eliminate mouse move events
    IF ThinNetMgr.Active THEN
       Res = 'Event: ' & RisNet:OCXGetLastEventName(Reference)
       !Cycle through all parameters
       LOOP Count = 1 TO RisNet:OCXGETPARAMCOUNT(Reference)
          !Get each parameter name
          Parm = RisNet:OCXGETPARAM(Reference,Count)
          !and concatenate them together
          Res = CLIP(Res) & ' - ' & Parm
       END
     ELSE
        Res = 'Event: ' & OleControl{PROP:LastEventName}
        !Cycle through all parameters
        LOOP Count = 1 TO OCXGETPARAMCOUNT(Reference)
           !Get each parameter name
           Parm = OCXGETPARAM(Reference,Count)
           !and concatenate them together
           Res = CLIP(Res) & ' - ' & Parm
        END
     END
  GlobalQue = Res                                 !Assign to a global QUEUE
  ADD(GlobalQue)                                  !and add the entry
  DISPLAY
 END
 RETURN(True)
```

IV. Thin@ Class Properties

This chapter covers the Thin@ class properties.

IV.1. General Thin@ class properties

1. ThinNetMgr.Active

Declaration

```
ThinNetMgr.Active BYTE
```

This property is used to check if the application is running in Thin@-mode.
Returns 1 if Thin@ is active, 0 if it is not active.

Example

```
IF ThinNetMgr.Active THEN
      ! <YOUR CODE HERE>
END
```

2. ThinNetMgr.ClientPrinter

Declaration

```
ThinNetMgr.ClientPrinter CSTRING
```

This is the default printer on the client side.

3. ThinNetMgr.State

Declaration

```
ThinNetMgr.State BYTE
```

This is the state of the Thin@ server.

It can be in one of these three states:
0 – Thin@ server waiting for the Client
1 – Thin@ sending a response to the Client
2 – Thin@ server in transition phase

4. ThinNetMgr.CurrentThread

Declaration

```
ThinNetMgr.CurrentThread LONG
```

This is the current thread in the Thin@ library. It can be different from THREAD().

Usage note: Exiting the window prior to opening

Sometimes in programming there is a need to exit a window even before it is opened. For example, if authorization checks are implemented directly in the WINDOW code. After the window is initiated, Thin@ will wait for the window to open. If the opening does not happen because of our forced RETURN procedure code, then the application will probably hang. It is advised to use this code before the OPEN(WINDOW) statement:

```
IF ThinNetMgr.Active THEN
  ThinNetMgr.CurrentThread=0{prop:thread}
  NOTIFY(401h,0{prop:thread})
 END
```

5. ThinNetMgr.ThreadBusy

Declaration

```
ThinNetMgr.ThreadBusy BYTE
```

Thread number that is currently communicating with the Client.

6. ThinNetMgr.ClientPath

Declaration

```
ThinNetMgr.ClientPath CSTRING
```

This is the default temporary folder on the client side.
The RisNet:DownloadFile function uses this as default if no other download path is specified.
On Windows platforms it is usually: *"C:\Documents and Settings\<UserName>\Local Settings\Temp"*

7. ThinNetMgr.FilePath

Declaration

```
ThinNetMgr.FilePath CSTRING
```

This is the default temporary folder on the server side.

On Windows platforms it is usually: *"C:\Documents and Settings\<UserName>\Local Settings\Temp"*

8. ThinNetMgr.FileDirPath

Declaration

```
ThinNetMgr.FileDirPath CSTRING
```

This is the default temporary folder created on the server for storing files of each user. The folder name is unique and is determined by the serial number of the client machine (visible from the NetSetup utility).

If *RisNet:UploadFile* function is used without specifying an upload path, this folder is created automatically and the file is stored in it.

However, if you want to use this variable and the folder is not already automatically created by *RisNet:UploadFile*, you can create it manually by executing the following function:

ApiCreateDirectory(ThinNetMgr.FileDirPath)

IV.2. Thin@ user session properties

The Thin@ user session properties contain a lot of useful information about a Thin@ user session.

```
ThinNetMgr.Stats.ProcessId         ULONG
ThinNetMgr.Stats.AppName           CSTRING(51)
ThinNetMgr.Stats.UserName          CSTRING(51)
ThinNetMgr.Stats.Password          CSTRING(51)
ThinNetMgr.Stats.ClientIpAddress   CSTRING(51)
ThinNetMgr.Stats.ClientWebAddress  CSTRING(51)
ThinNetMgr.Stats.ClientSerial      CSTRING(51)
ThinNetMgr.Stats.ClientType        CSTRING(51) ! Values: WinClient6,
WinClient7, Java Client
ThinNetMgr.Stats.ClientResolution  CSTRING(51) ! Format: 1920x1080
ThinNetMgr.Stats.ClientOSVersion   CSTRING(51) ! Values:
Win3.1,Win95,Win98,WinME,WinXP64Bit,WinXP,NT_3.51,NT_4.0,Win2000,WinServer200
3,Win2000,WinVista,WinServer2008,Win7,WinServer2008R2
ThinNetMgr.Stats.LastRefreshTime   LONG
ThinNetMgr.Stats.LastRefreshDate   LONG
ThinNetMgr.Stats.LastRefreshTimea  LONG
ThinNetMgr.Stats.LastRefreshDatea  LONG
ThinNetMgr.Stats.StartDate         LONG
ThinNetMgr.Stats.StartTime         LONG
ThinNetMgr.Stats.LongRunning       BYTE
ThinNetMgr.Stats.ActiveTime        LONG
ThinNetMgr.Stats.ClientType           CSTRING(51) ! Possible values:
WinClient6, WinClient7
ThinNetMgr.Stats.ClientResolution  CSTRING(51) ! Format: 1920x1080
ThinNetMgr.Stats.ClientOSVersion   CSTRING(51) ! Possible values: Win3.1,
Win95, Win98, WinME, WinXP64Bit, WinXP, NT_3.51, NT_4.0, Win2000,
WinServer2003, Win2000, WinVista, WinServer2008, Win7, WinServer2008R2
```

Usage notes

For example, if you would like to know the Thin@ username of the currently logged user, you would use the following code:

```
Username" = ThinNetMgr.Stats.Username
```

V. Implementing Thin@ in a 100% hand-coded application

Example Hand-coded Window Procedure:

```
PROGRAM

 MAP
handcoded PROCEDURE
 END

INCLUDE('thinn@.inc')

ThinNetMgr NetManager

 CODE
 handcoded

handcoded PROCEDURE

ListQueue QUEUE
Field1 STRING(100)
Field2 STRING(100)
 END

MyWindow  WINDOW('MyWindow'),SYSTEM,AT(,,225,123),FONT('MS Sans Serif', 8,,
FONT:regular),GRAY,Maximize
PROMPT('From Queue:'), AT(7,9), USE(?Prompt1)
LIST, AT(9,20,97,76), USE(?List1), FORMAT('20L(2)|M'), FROM(ListQueue)
LIST, AT(121,20,97,76), USE(?List2), FROM('FirstStringRow|SecondStringRow')
BUTTON('Refresh'), AT(183,102,35,14), USE(?Button)
PROMPT('From String:'), AT(121,9), USE(?Prompt2)
 END

  CODE
 OPEN(MyWindow)
IF ThinNETMgr.Active THEN ThinNETMgr.OpenWindow(MyWindow).

 ListQueue.Field1='FirstQueueRow'; ADD(ListQueue)
 ListQueue.Field2='SecondQueueRow'; ADD(ListQueue)

 !Has to be added after opening the window for every ListBox control
 IF ThinNetMgr.Active THEN
    ThinNetMgr.AddListControl(MyWindow, ?List1,ListQueue)
    ThinNetMgr.AddListControl(MyWindow, ?List2, ?List2{PROP:From})
 END !IF

 ACCEPT
    IF EVENT()=EVENT:Accepted AND FIELD()=?Button THEN
        ?List2{PROP:From} = ?List2{Prop:From} & '|NewStringRow'
!Has to be called every time you change a string which is used to fill a
!ListBox
        IF ThinNetMgr.Active THEN
            ThinNetMgr.AddListControl(MyWindow, ?List2, ?List2{PROP:From})
        END !IF
    END !IF

    IF ThinNetMgr.Active THEN !Has to be added in the end of the ACCEPT loop
        ThinNetMgr.TakeEvent(MyWindow)
    END !IF
 END !ACCEPT
IF ThinNETMgr.Active THEN ThinNETMgr.CloseWindow(MyWindow).
CLOSE(MyWindow)
```

This is an example hand-coded application. Thin@ calls are highlighted. The first highlighted block adds support for two LIST box controls. If the LIST box is populated from a queue, it is enough to add one line of code for each LIST box after opening the window.

However, if the LIST box is populated from a string, a line of code is necessary to refresh the LIST box after every change. Thus, the second block adds support for dynamically changing the {PROP:From} of a LIST box filled from a string.

The third block is mandatory for every window and it goes in the end of the window ACCEPT LOOP.

VI. Thin@ NetClient application tweaking

Being a Smart Client environment, the Thin@ developer installation package contains the full source code of the default Thin@ Client application (NetClient.app).

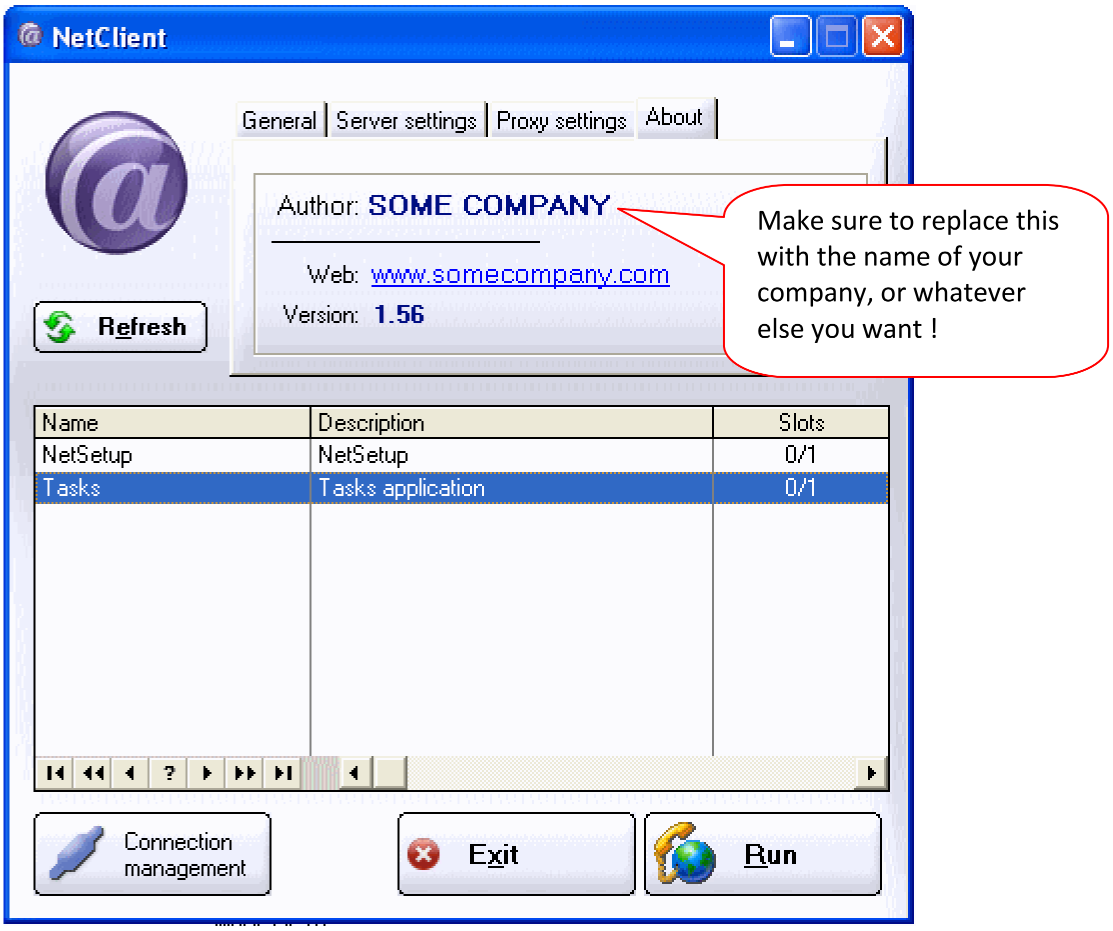
This chapter covers the most common reasons for modifying the Thin@ Client.

The chapter is divided in 5 sections, and those are:

- Modifying the Thin@ Client GUI
- Thin@ Client global embed points
- Adding Multilanguage support to the Thin@ Client
- Tweaking compression and decompression routines
- Tweaking the print preview window & making support for custom/3rd party print preview procedures

Note that there are other possible reasons for wanting to modify the default Thin@ Client that are not covered in this chapter. Examples include adding support for external devices (e.g. scanners), support for certain 3rd party products, calling Client-side Windows API functions etc.

VI.1. Modifying the Thin@ Client GUI



Author: SOME COMPANY

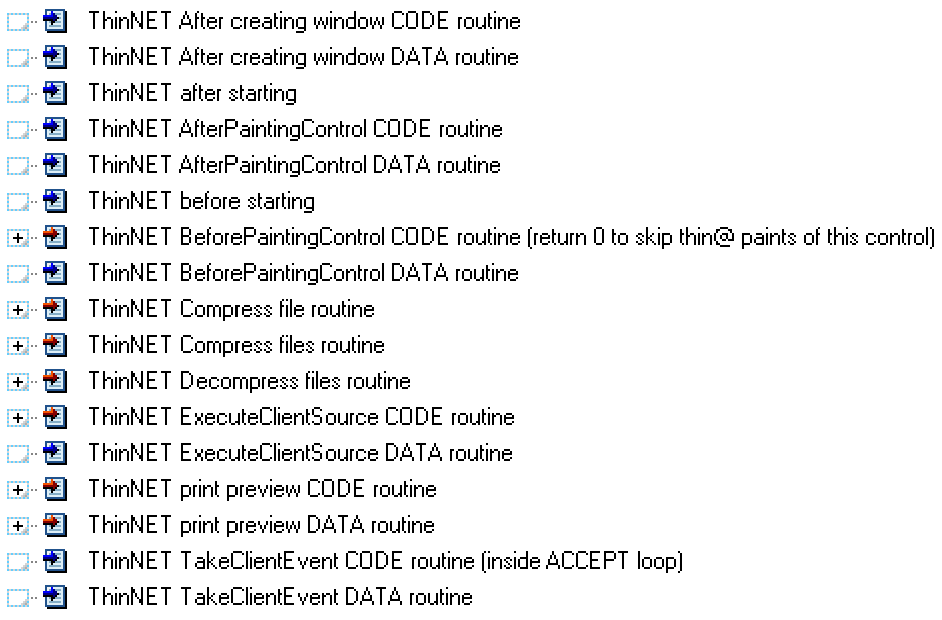
Web: www.somecompany.com

Version: 1.56

Name	Description	Slots
NetSetup	NetSetup	0/1
Tasks	Tasks application	0/1

Feel free to completely modify the look & feel of your client application, including the window size and appearance, the position and choice of various elements such as images, buttons, sheets, etc.

VI.2. Thin@ Client global embed points


ThinNET After creating window CODE routine
ThinNET After creating window DATA routine
ThinNET after starting
ThinNET AfterPaintingControl CODE routine
ThinNET AfterPaintingControl DATA routine
ThinNET before starting
ThinNET BeforePaintingControl CODE routine (return 0 to skip thin@ paints of this control)
ThinNET BeforePaintingControl DATA routine
ThinNET Compress file routine
ThinNET Compress files routine
ThinNET Decompress files routine
ThinNET ExecuteClientSource CODE routine
ThinNET ExecuteClientSource DATA routine
ThinNET print preview CODE routine
ThinNET print preview DATA routine
ThinNET TakeClientEvent CODE routine (inside ACCEPT loop)
ThinNET TakeClientEvent DATA routine

a) AfterCreatingWindow

This function is called each time a WINDOW is opened.

Prototype

```
AfterCreatingWindow    PROCEDURE(),VIRTUAL
```

Usage notes
You can add your own code that you wish to be executed every time after a WINDOW is opened on the Client side.

b) BeforePaintingControl

This function is called before a CONTROL is drawn on the Client side.

Prototype

```
BeforePaintingControl  PROCEDURE(LONG Feq,BYTE Created=0,BYTE
Last=0),BYTE,VIRTUAL
```

Usage notes
Rarely used.

Return value
If the derived function returns 0 the library will skip the paint process for that CONTROL.

c) AfterPaintingControl

This function is called after a CONTROL is drawn on the Client side.

Prototype

```
AfterPaintingControl   PROCEDURE(LONG Feq,BYTE Created=0,BYTE Last=0),VIRTUAL
```

Usage notes
Rarely used.

d) TakeClientEvent

This function is called for each event generated inside an ACCEPT loop.

Prototype

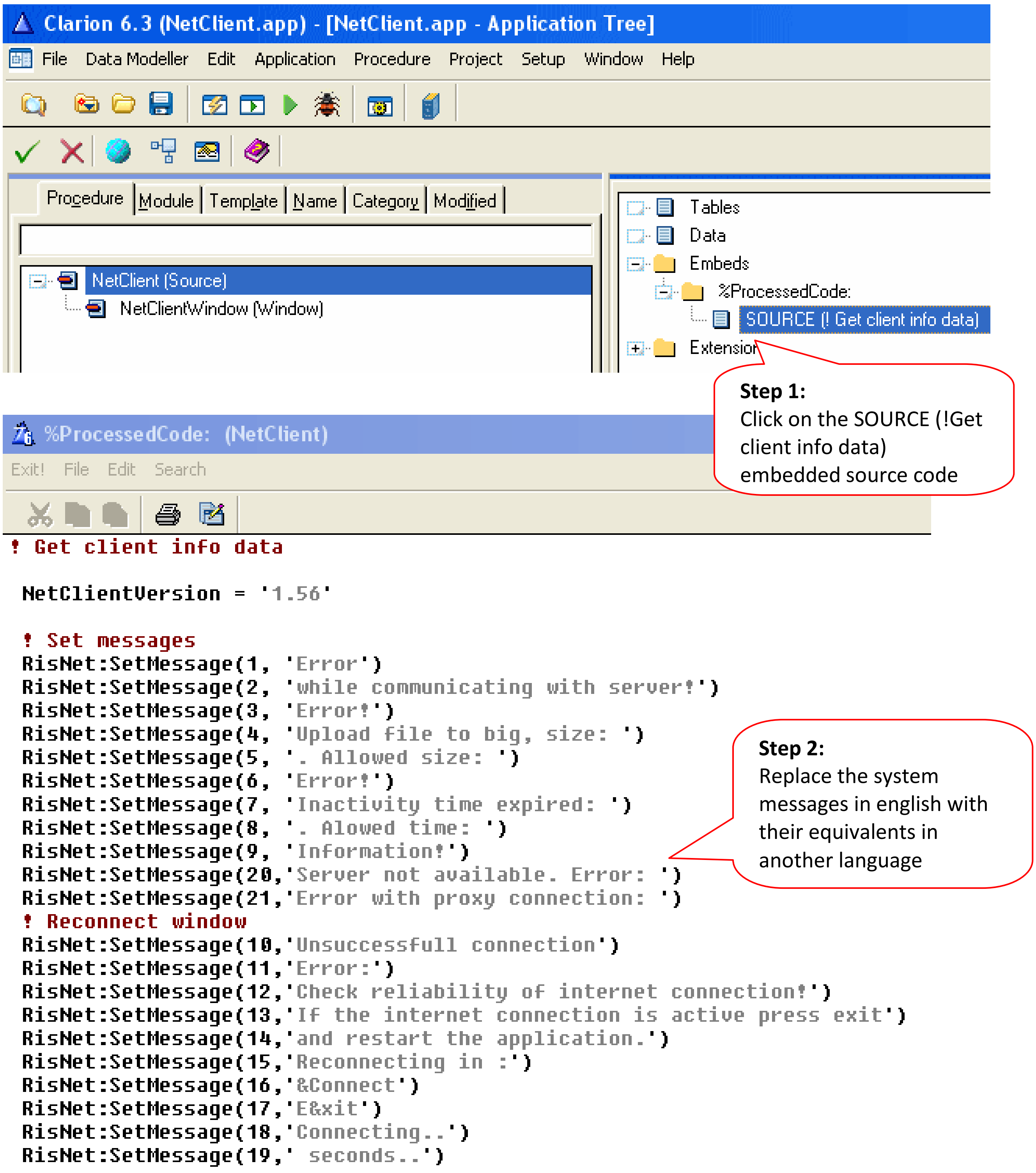
```
TakeClientEvent        PROCEDURE(*Window WindowHandle),BYTE,PROC,VIRTUAL
```

Return value
If the derived function returns 0 the library will skip sending the event to the Server side.

Usage notes
Rarely used.

VI.3. Adding Multilanguage support to the Thin@ Client

The language of the client system messages can be easily changed in the client source code:


Clarion 6.3 (NetClient.app) - [NetClient.app - Application Tree]
File Data Modeller Edit Application Procedure Project Setup Window Help
Procedure Module Template Name Category Modified
NetClient (Source)
NetClientWindow (Window)
Tables
Data
Embeds
%ProcessedCode:
SOURCE (! Get client info data)
Extension
Step 1: Click on the SOURCE (!Get client info data) embedded source code
%ProcessedCode: (NetClient)
Exit! File Edit Search
Step 2: Replace the system messages in english with their equivalents in another language

```
! Get client info data

 NetClientVersion = '1.56'

 ! Set messages
 RisNet:SetMessage(1, 'Error')
 RisNet:SetMessage(2, 'while communicating with server!')
 RisNet:SetMessage(3, 'Error!')
 RisNet:SetMessage(4, 'Upload file to big, size: ')
 RisNet:SetMessage(5, '. Allowed size: ')
 RisNet:SetMessage(6, 'Error!')
 RisNet:SetMessage(7, 'Inactivity time expired: ')
 RisNet:SetMessage(8, '. Alowed time: ')
 RisNet:SetMessage(9, 'Information!')
 RisNet:SetMessage(20,'Server not available. Error: ')
 RisNet:SetMessage(21,'Error with proxy connection: ')
 ! Reconnect window
 RisNet:SetMessage(10,'Unsuccessfull connection')
 RisNet:SetMessage(11,'Error:')
 RisNet:SetMessage(12,'Check reliability of internet connection!')
 RisNet:SetMessage(13,'If the internet connection is active press exit')
 RisNet:SetMessage(14,'and restart the application.')
 RisNet:SetMessage(15,'Reconnecting in :')
 RisNet:SetMessage(16,'&Connect')
 RisNet:SetMessage(17,'E&xit')
 RisNet:SetMessage(18,'Connecting..')
 RisNet:SetMessage(19,' seconds..')
```

VI.4. Tweaking compression and decompression routines

Thin@ uses two standard file compressor programs:
7zip - www.7zip.com
or
WinRar - www.winrar.com

7zip compressor is set as default, but you can change this by changing the value of the *ThinNetMgr.CompressionType* variable. Simply add this program line somewhere in your source code:

Using rar.exe compression:

```
ThinNetMgr.CompressionType = 0
```

Using 7z.exe compression:

```
ThinNetMgr.CompressionType = 1
```

If it is set to 0 (False) on the client and server side, Thin@ will use the '7z.exe' command line tool for file compression and decompression.

If it is set to 1 (True) on the client and server side, Thin@ will use the 'rar.exe' command line tool for file compression and decompression.

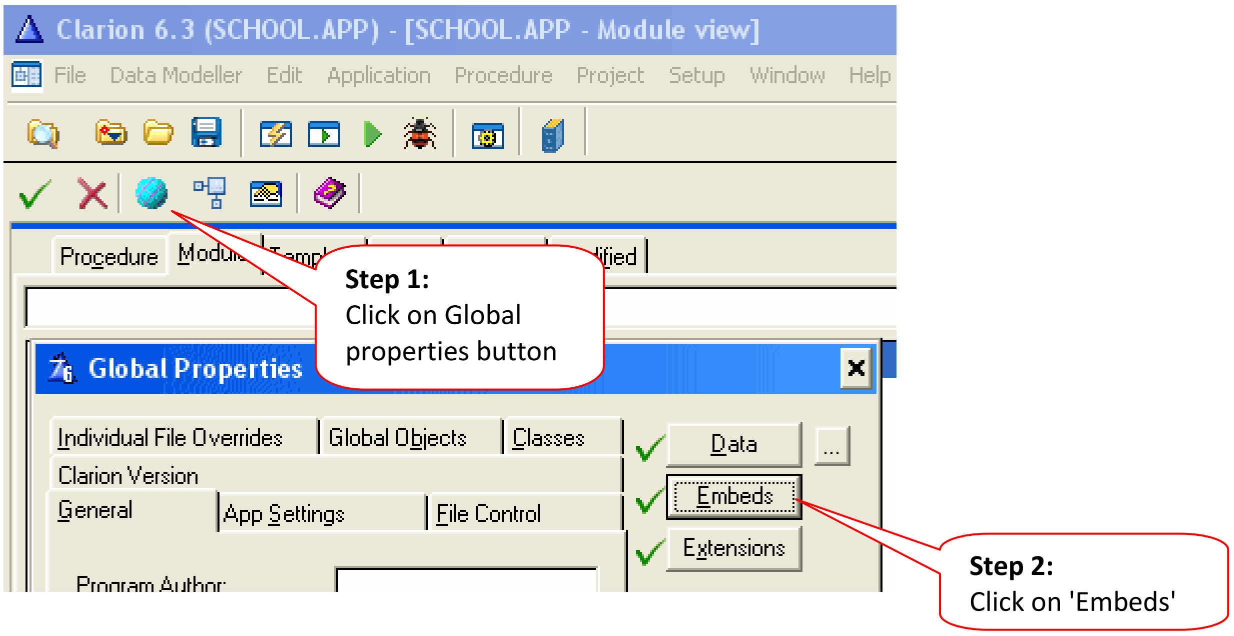
When using the above mentioned programs (7z.exe or rar.exe), they need to be available both on the server side and on the client side.

If there is a need (for some reason) to replace the internal routines with your own compression method, it is advisable to use the defined virtual routines. The *ThinNetMgr* class is defined in this way:

```
ThinNETMgr    CLASS(NetManager)
CompressFile    PROCEDURE(STRING FileName,STRING Archive),DERIVED
CompressFiles   PROCEDURE(*QUEUE QF,*? FileName,STRING Archive),DERIVED
DecompressFile  PROCEDURE(STRING Archive,STRING FilePath),DERIVED
    END
```

FileName – the name of the file you want to compress
Archive – the name of the archive you want to create
QF – the name of the queue which contains a list of files you want to compress
FilePath - the path in which the file archive will be decompressed

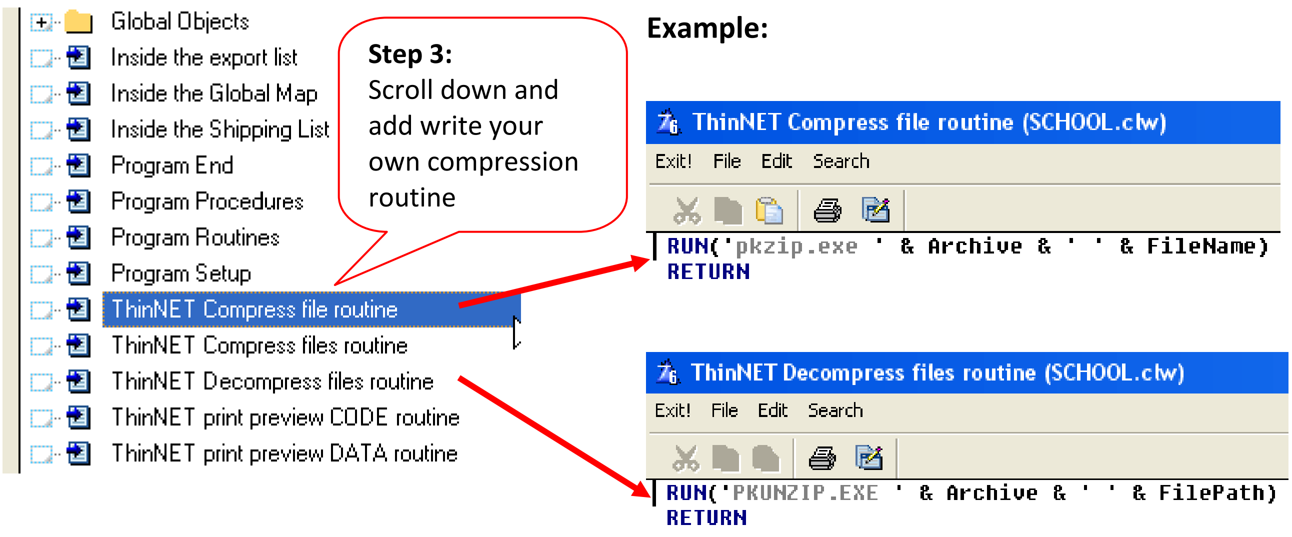
How to add your own compression and decompression calls in the defined routines:


Clarion 6.3 (SCHOOL.APP) - [SCHOOL.APP - Module view]
File Data Modeller Edit Application Procedure Project Setup Window Help
Procedure
Global Properties
Individual File Overrides
Global Objects
Classes
Clarion Version
General
App Settings
File Control
Data
Embeds
Extensions
Program Author

Step 1: Click on Global properties button

Step 2: Click on 'Embeds'

Example:


Global Objects
Inside the export list
Inside the Global Map
Inside the Shipping List
Program End
Program Procedures
Program Routines
Program Setup
ThinNET Compress file routine
ThinNET Compress files routine
ThinNET Decompress files routine
ThinNET print preview CODE routine
ThinNET print preview DATA routine
ThinNET Compress file routine (SCHOOL.clw)
Exit! File Edit Search
ThinNET Decompress files routine (SCHOOL.clw)
Exit! File Edit Search

Step 3: Scroll down and add write your own compression routine

```
RUN('pkzip.exe ' & Archive & ' ' & FileName)
RETURN
```

```
RUN('PKUNZIP.EXE ' & Archive & ' ' & FilePath)
RETURN
```

Don’t forget to finish your statements with a ‘RETURN’ call or the standard parent routines will be called as well!

VI.5. Tweaking the print preview window & making support for custom/3rd party print preview procedures

Thin@ uses the standard print preview dialog on the client side (PDF creating is included). If there is a need to replace the standard print preview dialog with something else, it is advisable to use the defined virtual functions. The *ThinNetMgr* class is defined in this way:

```
ThinNETMgr   CLASS(NetManager)
PrintPreview   PROCEDURE(*QUEUE PrintPreviewQueue,*? FileName, LONG
pLandscape), BYTE, DERIVED
  END
```

PrintPreviewQueue – a queue that contains all .wmf files downloaded and generated from the server side
FileName – a queue variable containing the exact file names on the client side
pLandscape – a parameter which indicates that the report should be previewed in landscape mode

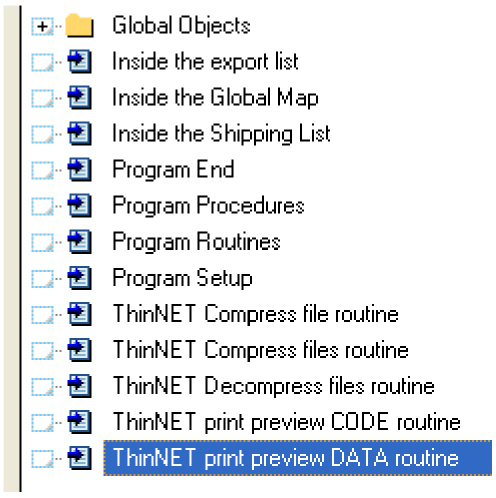
By supplying your own function call to this routine and supplying the RETURN call, the client program can be tweaked with additional preview routines. The function is using PrintPreviewQueue, FileName and Landscape parameters.

If you want to add your own print preview routine, you can add your own code to the *ThinNET print preview CODE routine* and to *the ThinNET print preview DATA routine* embed points.

This is how the original Thin@ print preview window routine looks like:

```
NetManager.PrintPreview PROCEDURE(*QUEUE
PrintPreviewQueue,*? FileName,LONG pLandscape)

Previewer       CLASS(PrintPreviewClass)
 END
TargetSelector  &ReportTargetSelectorClass
pWMFParser      &WMFDocumentParser
PDFReporter     CLASS(PDFReportGenerator)
 END
pReport REPORT
        END
ReportQueue     QUEUE(PrintPreviewFileQueue).
```



Global Objects
Inside the export list
Inside the Global Map
Inside the Shipping List
Program End
Program Procedures
Program Routines
Program Setup
ThinNET Compress file routine
ThinNET Compress files routine
ThinNET Decompress files routine
ThinNET print preview CODE routine
ThinNET print preview DATA routine

```
CODE
   FREE(ReportQueue)
   LOOP I#=1 TO RECORDS(PrintPreviewQueue)
     GET(PrintPreviewQueue,I#)
     ReportQueue.FileName = FileName
     ADD(ReportQueue)
   END
   pWMFParser &= NEW WMFDocumentParser
   IF TargetSelector &= NULL THEN
     TargetSelector &= NEW
ReportTargetSelectorClass
   END

TargetSelector.AddItem(PDFReporter.IReportGene
rator)
   Previewer.AllowUserZoom=True
   Previewer.Maximize=True

Previewer.Init(ReportQueue,TargetSelector,pWMF
Parser)
   IF Previewer.Display() THEN
     OPEN(pReport)
     IF pLandscape THEN
       pReport{PROP:Landscape}=pLandscape
     END

pReport{PROP:Preview}=ReportQueue.FileName
     ENDPAGE(pReport)
     pReport{PROP:FlushPreview} = True
   ELSE
     OPEN(pReport)

pReport{PROP:Preview}=ReportQueue.FileName
     ENDPAGE(pReport)
     pReport{PROP:FlushPreview} = False
   END
   CLOSE(pReport)
   FREE(ReportQueue)
   DISPOSE(pWmfParser)
   DISPOSE(TargetSelector)
   RETURN 1
```

Example:

The following code is an example of how to implement a 3rd party Print Preview procedure, in this case CPCS Reports (http://www.cpcs-inc.com/).
This code is already included in the global embeds section(PrintPreview routine) in the NetClient.app shipped with Thin@.

Data section:

```
!ReportQueue     QUEUE(PrintPreviewFileQueue)
```

Code section:

```
!! Custom Print Preview Procedure Example:
!! The following code is an example of how to implement a 3rd party Print
Preview procedure, in this case CPCS Reports.
!! You can use this section to make support for your 3rd party or custom
print preview procedure, or you can uncomment the following
!! code to use the CPCS Print Preview procedure

!!***************************************************************************
***********
!! CPCS Reports print preview support (uncomment to apply)
!!***************************************************************************
***********

! IF SkipPreview THEN
!   OPEN(pReport)
!   IF pLandscape THEN
!     pReport{PROP:Landscape}=pLandscape
!   END
!   pReport{PROP:Preview}=ReportQueue.FileName
!   ENDPAGE(pReport)
!   PRINTER{PROPPRINT:COPIES}=1 !Previewer.Copies
!   pReport{PROPPRINT:COPIES}=1 !Previewer.Copies
!   pReport{PROP:FlushPreview} = True
!   Printed# = 1
!   CLOSE(pReport)
!   FREE(ReportQueue)
!   RETURN Printed#
! ELSE
!   OPEN(pReport)
!   IF pLandscape THEN
!      pReport{PROP:Landscape}=pLandscape
!   END
!   pReport{PROP:Preview}=ReportQueue.FileName
!   ENDPAGE(pReport)
!   PRINTER{PROPPRINT:COPIES}=1 !Previewer.Copies
!   pReport{PROPPRINT:COPIES}=1 !Previewer.Copies
!   PreviewOptions = BOR(PreviewOptions,10000000b)

!! --- Print preview procedure call
!   Printed# =
PrintPreview(PrintPreviewQueue,100,'cpcs.ini',pReport,PreviewOptions,,,'','',
```

VII. Running a local test environment without installing the Thin@ server

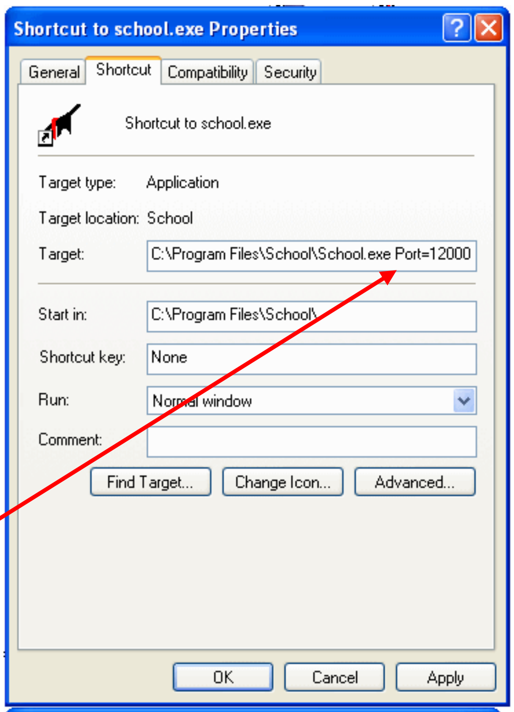
It is possible to test and run your application locally without the need to install the Thin@ application server environment.
Note that the steps covered in this chapter are not necessary for Clarion 7 and Clarion 8 environments, where the Thin@ Addin can be used instead.

Step 1:
Make a shortcut for your application

Shortcut to school.exe

Step 2:
Right click on the shortcut icon and select Properties.

Step 3:
On the Shortcut tab, edit the Target field and add the port parameter, as shown in the picture on the right.



Shortcut to school.exe

Target type: Application
Target location: School
Target: C:\Program Files\School\School.exe Port=12000
Start in: C:\Program Files\School\
Shortcut key: None
Run: Normal window
Comment:

Example:
C:\Program Files\School\School.exe Port=12000

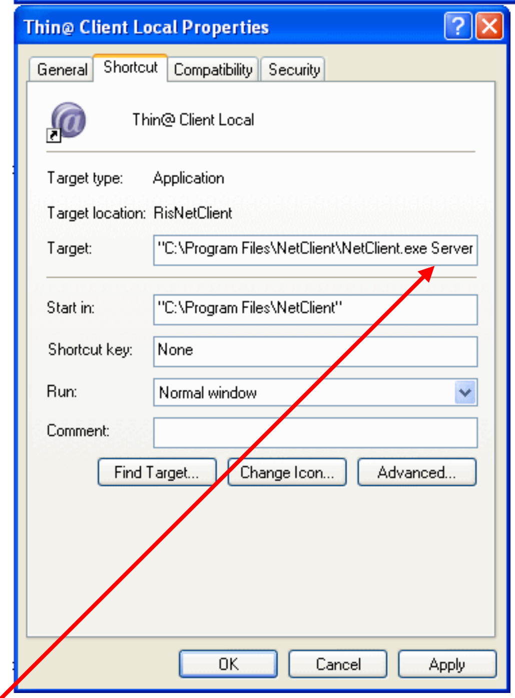
This means that the application will run in the Thin@ 'hidden' mode and that it will open only after you start the client locally.

Step 4:
Make a NetClient shortcut.

Thin@ Client Local

Step 5:
Right click on the shortcut icon and select Properties.

Step 6:
On the Shortcut tab, edit the Target field and add the port parameter, as shown in the picture on the right.



Thin@ Client Local

Target type: Application
Target location: RisNetClient
Target: "C:\Program Files\NetClient\NetClient.exe Server
Start in: "C:\Program Files\NetClient"
Shortcut key: None
Run: Normal window
Comment:

Example:
"C:\Program Files\NetClient\NetClient.exe
Servername=localhost ServerPort=12000

This means that the client will try to open the application on port 12000, using 'localhost' instead of an IP or DNS address.

Step 7:
First start your application, and then start the NetClient.

Your application will now be running in your local test environment!

NOTE: The execution of these .exe files can also be done through a single .bat file which executes both statements at once. All you have to do is to create a .bat file which first starts your application, and then starts the NetClient.

For example:

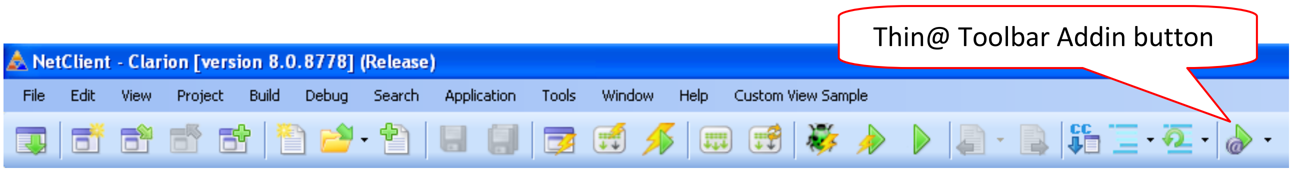
School.bat

```
@start /d"C:\ Program Files\Schools\" school.exe Port=12000
@start /d"C:\ Program Files\NetClient\" NetClient.exe ServerName=localhost ServerPort=12000
```

VIII. Thin@ Addin (C7 & C8 only)

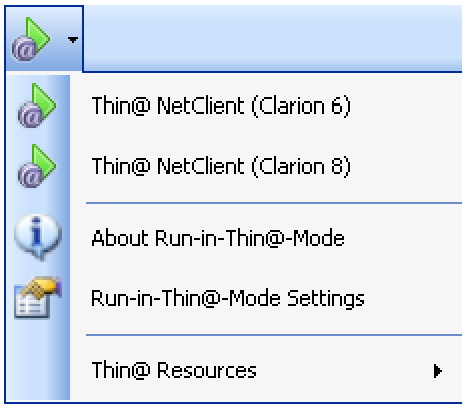
With the Thin@ toolbar addin you can:

1) Quickly test how your application works in Thin@-mode without configuring a Thin@ Server (and uploading the application to the Thin@ Server). This makes testing your Thin@ application in C7 and C8 quicker and simpler than with Clarion 6.
2) Access some of the most useful Thin@ resources such as documentation and other web resources



To quickly test how your application works in Thin@-mode, all you have to do is recompile your application with the Thin@ template and press .

This will use the default Thin@ Client to run the application, the location of which is *"C:\Program Files\Thin@ Client\Clarion7\NetClient.exe".* If you changed this path during the Client installation you need to modify this setting and provide full path to a Thin@ Client executable.

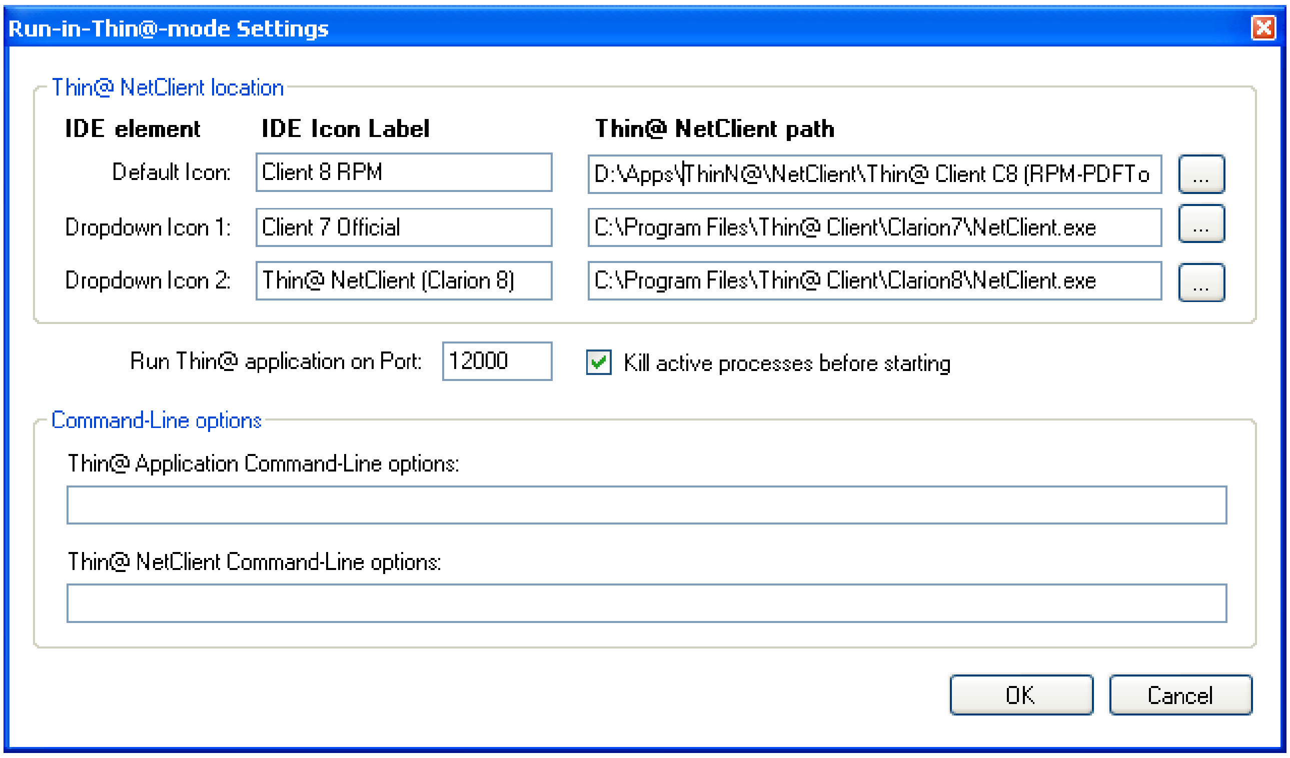

Thin@ NetClient (Clarion 6)
Thin@ NetClient (Clarion 8)
About Run-in-Thin@-Mode
Run-in-Thin@-Mode Settings
Thin@ Resources

The Thin@ Addin lets you configure up to three Thin@ Clients so that you can quickly test your application with different versions of the Client.
By default these are the standard C7, C8 and C6 versions.

Run-In-Thin@-Mode Settings

You can modify the IDE Icon Label, the paths to the client, the port on which the application will run, whether to kill already started application/client instances (recommended) or not and you can pass command line startup parameters to the Thin@ application and Client.

Based on the following configuration, pressing the IDE button will start the application using the custom Thin@ Client for RPM.



Run-in-Thin@-mode Settings

Thin@ NetClient location

IDE element	IDE Icon Label	Thin@ NetClient path
Default Icon:	Client 8 RPM	D:\Apps\ThinN@\NetClient\Thin@ Client C8 (RPM-PDFTo
Dropdown Icon 1:	Client 7 Official	C:\Program Files\Thin@ Client\Clarion7\NetClient.exe
Dropdown Icon 2:	Thin@ NetClient (Clarion 8)	C:\Program Files\Thin@ Client\Clarion8\NetClient.exe

Run Thin@ application on Port: 12000

[✓] Kill active processes before starting

Command-Line options

Thin@ Application Command-Line options:

Thin@ NetClient Command-Line options:

IX. 3rd party products support for Thin@

Thin@ features support for a number of 3rd party products for Clarion.

This is especially important if you're transitioning an existing Clarion application to Thin@ SaaS, because the application probably has a number of 3rd party products.

When it comes to support for Thin@, 3rd party products can be divided into the following categories:

- Products that work with Thin@ out-of-the-box
- Products with integrated Thin@ support
- Products supported by Thin@
- Products that do not support Thin@

3rd party products with integrated Thin@ support:

- ClarionTools

3rd party products supported by Thin@:

- Noyantis CommandBars
- Noyantis ShortcutBar
- Noyantis ChartPro
- Capesoft AnyFont
- Capesoft Insight Graphing
- Capesoft FTP
- Fomin Report Builder
- CPCS Reports
- RPM

X. Known Server-Client version compatibility issues

In case that Clarion 6.2 is used to compile the (Server-side) application and the application contains RTF controls, the Thin@ Client compiled in Clarion 6.2 must also be used.