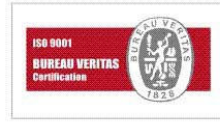




RIS d.o.o.  
Address: Pilepic 10, 51215 Kastav, Croatia  
Phone: +385 51 687 500, Fax: +385 51 687 501  
e-mail: info@ris.hr, web: http://www.ris.hr



Last Update: June 2011



# Thin@ Programmer's Guide v1.51

## 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 (1.51).....	3
II.	Thin@ template options .....	4
III.	Programming functions and methods .....	7
III.1.	Automatically generated by the Thin@ template .....	7
III.1.1.	Example Clarion application with Thin@ support .....	9
III.2.	Manually added to a Thin@ application.....	10
III.2.1.	Adding support for OLE/OCX/ActiveX controls.....	24
III.2.2.	Example application using OCX controls.....	28
IV.	Thin@ class properties.....	32
IV.1.	Thin@ (Server-side) properties.....	32
IV.2.	Thin@ user session properties.....	34
V.	Implementing Thin@ in a 100% hand-coded application .....	35
VI.	Thin@ NetClient application tweaking .....	37
VI.1.	Client window graphical user interface .....	37
VI.2.	Client-side global embed points .....	38
VI.3.	Client multi-language support .....	39
VI.4.	Tweaking compression and decompression routines .....	40
VI.5.	Tweaking the print preview window & making support for custom/3 <sup>rd</sup> party print preview procedures .....	42
VII.	Running a local test environment without installing the Thin@ server.....	45

## **I. List of unsupported standard Clarion features in this version (1.51)**

1. Browse Grid
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 options

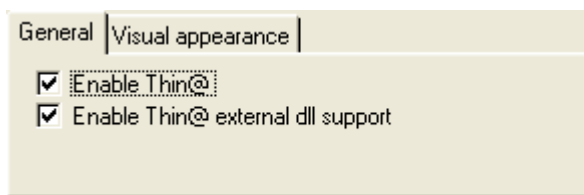
ThinNET is the name of the template shipped with Thin@ which adds Thin@-specific lines of code to a Clarion application thus making it Thin@-ready. In *Chapter 3 – Programming functions and methods* you can find out more about the Thin@-specific lines of code generated automatically by the Thin@ template.

Some Thin@ features can be customized by the Thin@ template options.

This chapter gives an overview of the Thin@ template options.

### a) Thin@ Global Extension options

#### - General Tab



**Enable Thin@** - Enable/disable Thin@ application support

**Enable Thin@ external dll support** – Enable/disable support for Thin@ External Class. (The Thin@ External Class is used by Clarion 3<sup>rd</sup> Party Vendors to help them in making support for Thin@ in their products).

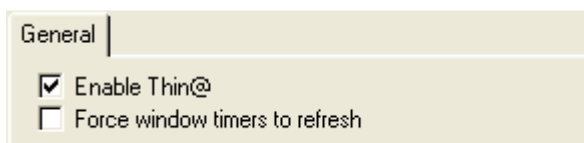
#### - Visual appearance Tab



**SheetTab Style** – The default style of a SHEET control (visible only in Thin@ Clients compiled in Clarion7 & Clarion8)

**Enable Listbox Header Colors** – Listbox header color settings (visible only in Thin@ Clients compiled in Clarion7 & Clarion8)

### b) Thin@ Procedure Extension options



**Enable Thin@** - Enable/disable Thin@ for the selected procedure

**Force window timers to refresh** – If checked, timer events will cause the window to refresh

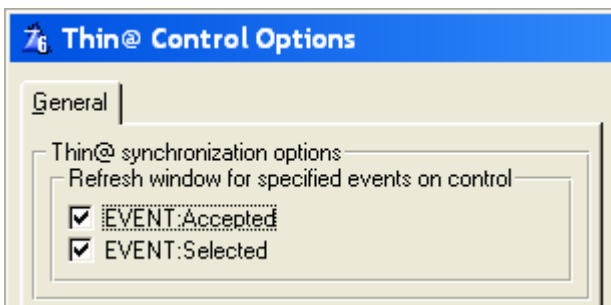
### c) Thin@ Control Options

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.



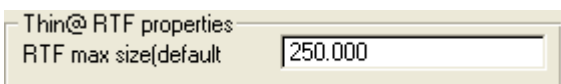
**Thin@ - ignore this control** – If checked, the control will not be visible in Thin@-mode.

### a) Thin@ synchronization options



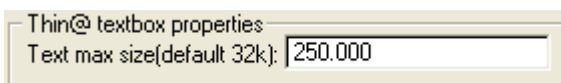
Thin@ synchronization options exist on every Clarion control that can trigger EVENT:Accepted and EVENT:Selected. By default, the Thin@ Client will communicate with the Thin@ Server on every such event. However, it's possible to disable this Thin@ Client-Server communication on EVENT:Accepted and/or EVENT:Selected 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.

### b) Thin@ RTF control properties



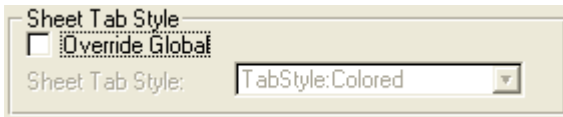
**RTF max size (default 250kB)** - The maximum size of a RTF control (in bytes) in Thin@-mode.

### c) Thin@ textbox properties



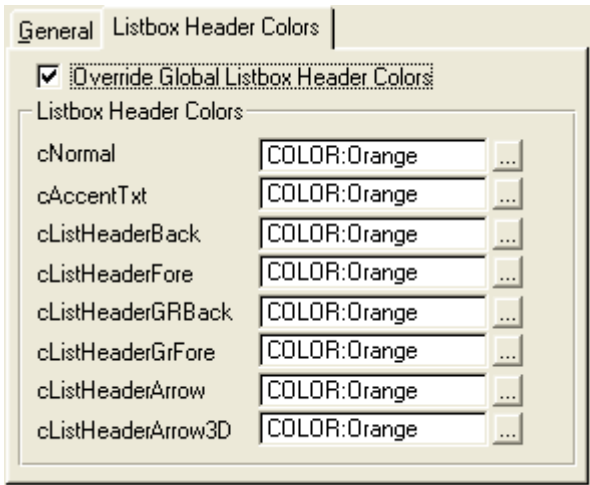
**RTF max size (default 32kB)** - The maximum size of a RTF control (in bytes) in Thin@-mode.

#### d) Sheet Tab Style



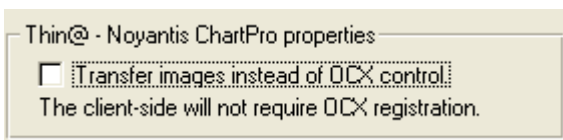
You can override the global tab style settings per control.

#### e) Listbox Header Colors



You can override the global listbox header colors per control.

#### f) Noyantis ChartPro properties



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.

### 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 the ones that are automatically generated by the Thin@ template.

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

#### III.1. Automatically generated by the Thin@ template

The lines of code automatically generated by the Thin@ template are necessary to make an application work as a thin client application. Although you won't be using much of them (except if you're hand-coding your applications), it's good to know what the template does for you.

##### a) Including the Thin@ library in your application

Generated at embed point 'After global includes':

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

##### b) Declaring the Thin@ ThinNetMgr class

Generated at embed point 'Global data':

```
ThinNetMgr NetManager
```

##### c) Starting Thin@ communication (\*)

Generated at embed point 'Program setup':

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

**\*Note:** Calling this method is optional and it's used only if you want to start the Thin@ communication before opening the 1<sup>st</sup> window.

#### d) 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.

#### e) Taking application events

Generated at the end of every ACCEPT LOOP:

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

#### f) Making Thin@ aware of a listbox

Generated at window opening window for each list box add:

```
IF ThinNetMgr.Active THEN  
    ThinNetMgr.AddListControl (<WindowName>, <ListFreq>, <ListQueueSource>)  
END !IF
```

---

NOTE: The Thin@ template can generate other Thin@-related code as well, but that code is either optional or it is generated in special cases and it is not included in this list.

Examples of optionally generated code include setting listbox header colors (template option) using RisNet:SetListboxHeaderColors, setting tab style (template option) using RisNet:SetTabStyle etc.

Examples of code generated in special cases include ThinNetMgr.StartLongRunningProcess(), which is generated for every report to override the default timeout; ThinNETMgr.DownloadFiles which is used in several occasions to support 3<sup>rd</sup> party templates etc.

### III.1.1. Example 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.2. Manually added to a Thin@ application

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.

#### 1. Checking if Thin@ is active

```
ThinNetMgr.Active
```

**Return value:** Returns 1 if Thin@ is active, 0 if it's not active.

**Example:**

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

#### 2. Creating listboxes manually

If you create your listboxes (includes combo and drop boxes) manually, with the *create* command or change listbox queues with the *prop:from* command, you need to instruct Thin@ how to read such controls.

Use this function if your listbox has a queue as its source data:

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

**WindowHandle** – the name of the window object

**Feq** – the control number (for example *?List*)

**ListQueue** – the name of the listbox queue

Use this function if your listbox has a string expression as its source data:

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

**WindowHandle** - the name of the window object

**Feq** - the control number (for example *?List*)

**DataString** - source data in form of a string expression

### 3. FileDialog

The standard Clarion FileDialog function is supported.

NOTE: when an application is run in the Thin@ environment the file dialog will contain the directory and files on the client side.

### 4. Printer dialog

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.

### 5. Downloading files

- **Downloading a single file:**

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

**FileName** – full path name on the application server

**Sign** – use 'File<-4->1' in this field to force the client to open the file dialog and prompt for a download destination

**FilePath** – enter the default download path on the client side; if omitted, the files are stored in the default temporary folder on the client side, readable with the *ThinNetMgr.ClientPath* variable.

**TargetName** – you can optionally set a new filename for the downloaded file.

**ChangeCheck** – if set to 1, Thin@ will check if there is already a file on the client with the same name, and if there is, it will compare it with the file which is preparing for download. The file will be downloaded only if there is a difference between the two.

#### 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:\')
```

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

- **Downloading multiple files:**

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

**SourceQueue** – the queue name containing multiple files

**FileName** – the queue field containing the full pathname

**Sign** – use 'File<-4->1' in this field to force the client to open the file dialog and prompt for a download destination

**FilePath** – enter the default download path on the client side; if omitted, the files are stored in the default temporary folder on the client side, readable with the *ThinNetMgr.ClientPath* variable.

Note: Both RisNet:DownloadFile and RisNet:DownloadFiles will compress the designated files and download them to the client machine.

## 6. Uploading files

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

**FileName** – the full pathname of the uploaded file on the client side

**FilePath** – the uploading 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).

NOTE: 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:\')
```

## 7. Running programs on the client side

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

**ExecuteString** – run expression

**WaitFlag** – advise the client to wait for an instruction to end (0 or 1)

**ShowFlag** – advise the client to hide / show activated process (0 or 1) (default 0)

**Return value:** 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](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](http://msdn.microsoft.com/en-us/library/ms681382(v=vs.85).aspx)

The RisNet:RunOnClient function can start a process on the client side. You can use it for executing client-side programs, scripts, etc.

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.

#### 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
```

## 8. Canceling window close

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

Sometimes in programming there is a need to cancel the window termination even after event:closewindow is processed. In standard client / server environment it is done by simply calling the CYCLE() function before accept termination. However, in the Thin@ environment there is a need to call the *RisNet:CancelCloseWindow()* function prior to the CYCLE() call.

#### 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
```

## 9. 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
```

This code will automatically switch to the current thread and window and notify its activity. It will not wait for a new window to open as well.

## 10. Initiating long running server jobs

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

a) After each user action, the client sends the request to the server application, which starts to process it. If the server application is unresponsive for longer than 40 seconds (due to a long running transaction or user inactivity), the client will lose server connection and the reconnect window will appear.

However, the client will reconnect to the server side application but only after the long running process has ended.

The solution to this is to use the Long Running function to warn the client of an active batch process. It is advisable to use this program function on statements which execute time is expected to last longer than 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.

So, if there is a need to initiate a long running job on the server side, use these statements in your program:

**On process start:** `ThinNetMgr.StartLongRunningProcess ()`

**On process end:** `ThinNetMgr.EndLongRunningProcess ()`

**NOTE:** The Thin@ template automatically generates a long running call for reports.

b) You might want to limit the maximum amount of time an application user can be inactive, after which he/she is kicked out of the application. The server application setting *<timeout period>* can be used to implement a non-activity timeout. Its default value is set to 45 minutes.

## 11. Showing progress dialog window

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

You can use this function if you want to show a progress dialog window containing a progress bar that is automatically populated and refreshed every 3 seconds.

The input parameters for your progress bar are:

**pDone** – Percentage of job done (for example, 10)

**pTotal** – Total job percentage (for example 100)

**pCancel** – a value of 1 will show a Cancel button on the window

**pTitle** – Window title

**Return value:** The procedure returns 1 if the process is completed.

**Example:**

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

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
```

## 12. Setting the client printer

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

**PrinterName** – The name of the printer on the client side.

This procedure sets the client printer (readable by ThinNetMgr.ClientPrinter).

**Example:**

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

### 13. Executing a code only on the client side

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

**Var1...Var7** – Input strings that you can use to execute specific blocks of code on the client side or pass parameters to the client.

**Return value:** The function can return any string value from the client back to the server.

This procedure will execute a source code on the client side.

To use it, you need to add a function call to your (server side) application, and you need to add a source code to your Thin@ client that will execute on your client machine.

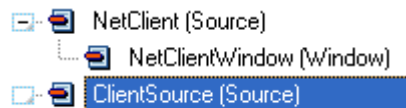
#### Example:

Add this line of code to your server side application:

```
IF ThinNetMgr.Active THEN  
  ThinNetMgr.ExecuteClientSource ('1')  
END
```

In order for the code to be executed on the client side, add this line of code in the ClientSource procedure in your Thin@ client application:

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



Notice that the procedure returns a value, so you can use it also to send messages from the client to the server. For example, if you would want to detect the client's machine screen resolution and send that information to the server you could write a piece of code such as:

#### Example 2 (Client returning a value to the server):

On the server side:

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

On the client side:

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

#### 14. Manually starting the Thin@ manager

```
ThinNetMgr.Start()
```

#### 15. Getting the client folder path

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

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

#### 16. Setting the client folder path

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

**PathExpression** – a folder path value

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

**Example:**

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

#### 17. Returning the Thin@ library state of the <control>{prop} values

```
ThinNetMgr.GetControlOption(LONG Feq, STRING pOption), STRING
```

#### 18. Forcing window refresh for specific/all threads in current execution loop

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

NOTE: The windows will refresh even if not focused.

#### 19. Force redraw of the control image if the image changed but the image filename is unchanged

```
ThinNetMgr.DisplayControlImage(LONG Feq)
```

#### 20. Delay application on the server without using processor time

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

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.

## 21. Posting events to windows on other threads and forcing them to display

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

**ThreadNo** – the thread number, returned by THREAD()

Although Thin@ allows posting events to windows on other threads, the results of that event won't be displayed on the other window until the window gains focus.

If you want to force display of other threaded windows while the focus is still on the current window, you can do it with the ThinNetMgr.DisplayThread() function. It forces window refresh of another threaded window but the focus returns to the current window.

### Example:

In the following example there are two windows, each on its own thread.

Window 2 saves its thread number after opening:

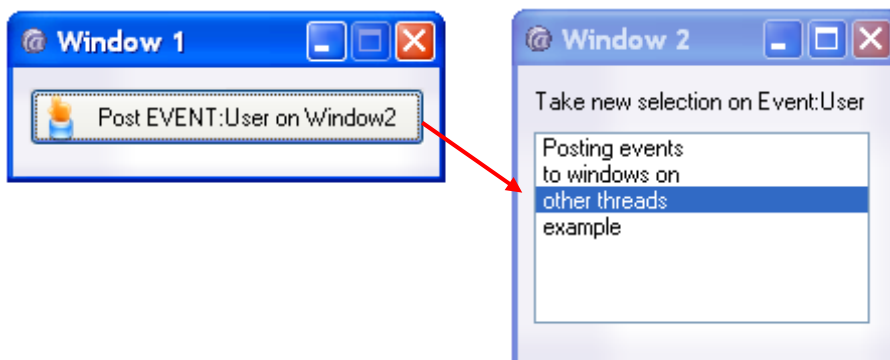
```
! Embed point: After Opening the Window  
Winthread[2] = THREAD() ! Save thread 2
```

Window 2 has some code on EVENT:User:

```
! 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])
```



## 22. Forcing immediate window refresh

```
ThinNetMgr.Display PROCEDURE (<*Window WindowHandle>, BYTE ForceEnd=0), BYTE
```

**WindowHandle** – the name of the Window object

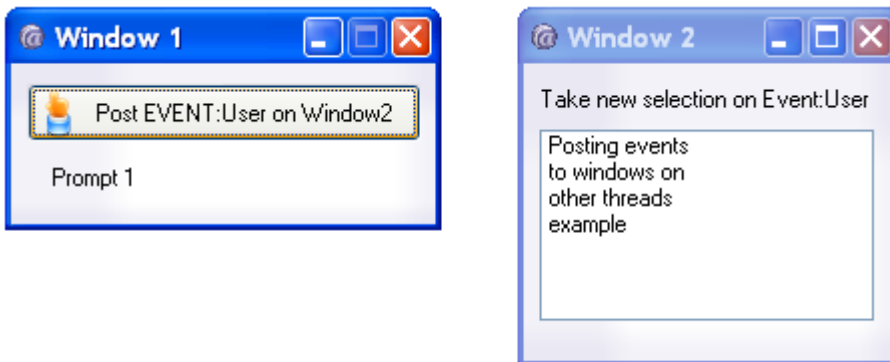
**ForceEnd** – internal use

If you want to refresh the current window immediately, without waiting for the end of all window events, you can do it with the ThinNetMgr.Display(<Window>) function.

### Example:

The following example demonstrates one of the situations where immediate window refresh can be useful.

In our example we have two windows, each on its own thread.



Adding the following code on Window 1 button would attempt to change the ?Prompt1 text, then switch thread to Window 2 and then 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, the text of ?Prompt1 on Window 1 would not change because before it happens Thin@ would switch the current thread to Window 2 (by setting the ThinNetMgr.CurrentThread variable). 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 switching threads:

```
?Prompt1{prop:text} = 'Changed text...'  
ThinNETMgr.Display(Window1) ! Force window display  
  
ThinNetMgr.CurrentThread = WinThread[2]  
POST(Event:User,,WinThread[2])
```

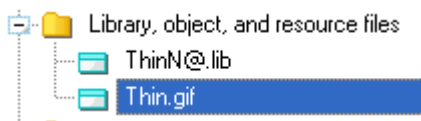
### 23. Adding resource folder (images, icons, etc.)

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

**ImagePath** – Full pathname to the resource folder on the server

**OnTop** – set to 1 if you want to add this resource folder on top of scanning list (possible speed optimization).

In some cases you want to include resource files in your application so that you don't have to ship those files with your application.



In a Thin@ application, all images must be found on the server. Thin@ by default scans the root application folder and the \Images subfolder (if it exists), and if a resource file you're using in your application is not found on those folders they won't display.

The solution is to register a Thin@ resource folder with the AddImageFolder function.

#### Example:

The following function registers an additional image folder on the server in which Thin@ will search for resource files.

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

### 24. Adding resource library (images, icons, etc.)

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

**ImageLibrary** – Full pathname to the 7-zip compressed resource file on the server

**ImageLibrarySubdirectory** – Name of the resource subdirectory automatically created in the Temp folder on the server where the resource files are decompressed (e.g. C:\Documents and Settings\\Local Settings\Temp\ImageLibrarySubdirectory\)

**ForceDownloadCheck** – Set to 1 to compare the files in the ImageLibrary with already downloaded files with the same name on the client, and overwrite the files on the client if the files on the server are newer

In some cases you might want to distribute all your resource files inside a resource archive, and optionally protect it with a password. You can do it Thin@ by using the AddImageLibrary function.

**Example:**

The following 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')
```

## 25. Maximizing windows and getting windows maximized state

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

**WindowHandle** – The name of the Window object

**ReturnValue:** 1 if the window is maximized, 0 if it's not

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

**WindowHandle** – The name of the Window object

**MaxState:** 1 = maximize, 0 = normal

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, the window is resized to the maximum size determined by her local OS resolution, meaning that the window on the server gets resized to that size. In other words, 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, and you set a window to maximized using that attribute.

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:**

```
!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
```

## 26. Changing listbox header colors

```
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)
```

**Enabled** – Enable/disable ListboxHeaderColors (1 = enabled, 0 = disabled)

**cListHeaderBack** – header background color

**cListHeaderFore** – header foreground (text) color

**cListHeaderGRBack** – header group background color

**cListHeaderGrFore** – header group foreground (text) color

**cListHeaderArrow** – header (sort) arrow color

**cListHeaderArrow3D** – header (sort) 3D arrow color

RisNet:SetDefaultListboxHeaderColors enables you to set a default header color for all the listbox headers in an application. NOTE: Also available as a Thin@ template option.

```
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)
```

**Window** – name of the window object

**FEQ** – FEQ of the listbox control

NOTE: The rest of the properties are the same as in RisNet:SetDefaultListboxHeaderColors

RisNet:SetListboxHeaderColors enables you to set header colors for a specific listbox and therefore override the default settings. NOTE: Also available as a Thin@ template option.

## 27. Changing sheet tab styles

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

**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)

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. RisNet:SetDefaultTabStyle sets the default tab style for the application. NOTE: Also available as a Thin@ template option.

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

**Window** – name of the window object

**FEQ** – FEQ of the sheet control

**pTabStyle** – numeric value indicating the style of the tab

RisNet:SetListboxHeaderColors enables you to set the tab style a specific sheet control and therefore override the default settings. NOTE: Also available as a Thin@ template option.

## 28. Starting a Thin@ application programmatically

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

**ApplicationName** – name of the Thin@ application registered in a Thin@ server that you wish to run

It is possible to run other Thin@ applications from an already running Thin@ application instance using the RisNet:RunThinApplication function. 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.

### III.2.1. Adding support for OLE/OCX/ActiveX controls

#### a) Creating OLE/OCX/ActiveX controls

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

**WindowHandle** – the name of the window object.

**Feq** – The FEQ of the OLE control

**CreateStatement** – the name of the OLE control to create

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.

#### 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
```

#### b) Assigning properties to OLE/OCX/ActiveX controls

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

**Feq** – the FEQ of the OLE control

**Expression** – a) the name of the OLE property you want to assign a value to OR

b) a PROP attribute of the OLE control you want to change

**SetValue** – the value you want to assign to the property OR to the PROP attribute

This method sets a property of the client-side OLE control OR it can be used to set a PROP attribute of an OLE control. Use this method in your application instead of the <OLE Control>{<Property>}= statement.

#### Example 1: Setting an OLE property

The following example sets the 'Value' property of an OLE 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

The following example sets 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
```

### c) Retrieving properties from OLE/OCX/ActiveX controls

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

**Feq** – the FEQ of the OLE control

**Property** – a) the name of an OLE property OR  
b) a PROP attribute of the OLE control

**Return value:**

- If the Property attribute is a name of an OLE property then the function returns the value of the specified OLE property.
- If the Property attribute is in fact a PROP attribute then the function returns the value of the specified PROP attribute.

### Example 1: Getting an OLE property

The following example gets the 'Value' property of an OLE control.

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

### Example 2: Getting a PROP attribute of the 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
```

### d) Installing OCX callback event procedure

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

**Feq** – The FEQ of the OLE control to affect.

**EventProcAddress** – The memory address of the event processing callback procedure for the control.

RisNet:OCXRegisterEventProc installs an OCX event callback procedure for the control. The callback procedure is called when any event is posted by the operating system for 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
```

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

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

PROP>LastEventName.

**Example:**

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

**f) Getting the name of the last event sent to an .OCX control**

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

**Reference** – The label of the first parameter of the event processing callback procedure.

RisNet:OCXGetParamCount returns the number of parameters associated with the current .OCX event. 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
```

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

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

**Reference** – The label of the first parameter of the event processing callback procedure.

**Count** – The number of the parameter to retrieve.

RisNet:OCXGetParam returns the value of the number parameter associated with the current .OCX event. 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
```

### h) Registering the OCX control with the client OS

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

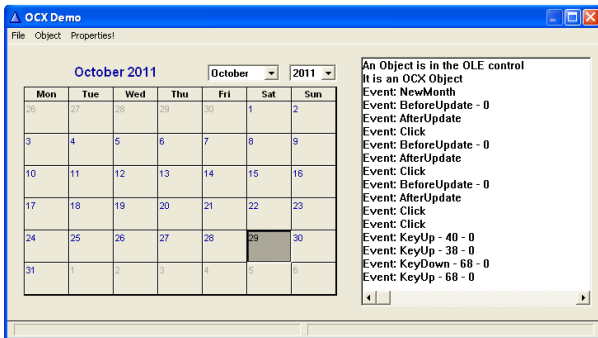
**FileName** – The full path to the OCX file you want to register

**ForceRegistration** – if set to 1 it will attempt to register the OCX control even if it is already registered.

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 ForceRegistration 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.

### III.2.2. Example application using OCX controls



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. Thin@ (Server-side) properties

```
ThinNetMgr.ClientPrinter CSTRING
```

This is the default printer on the client side.

```
ThinNetMgr.State BYTE
```

You can check 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

```
ThinNetMgr.CurrentThread LONG
```

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

```
ThinNetMgr.ThreadBusy BYTE
```

Thread number that is currently communicating with the client.

```
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"

```
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"

**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
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
```

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

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

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

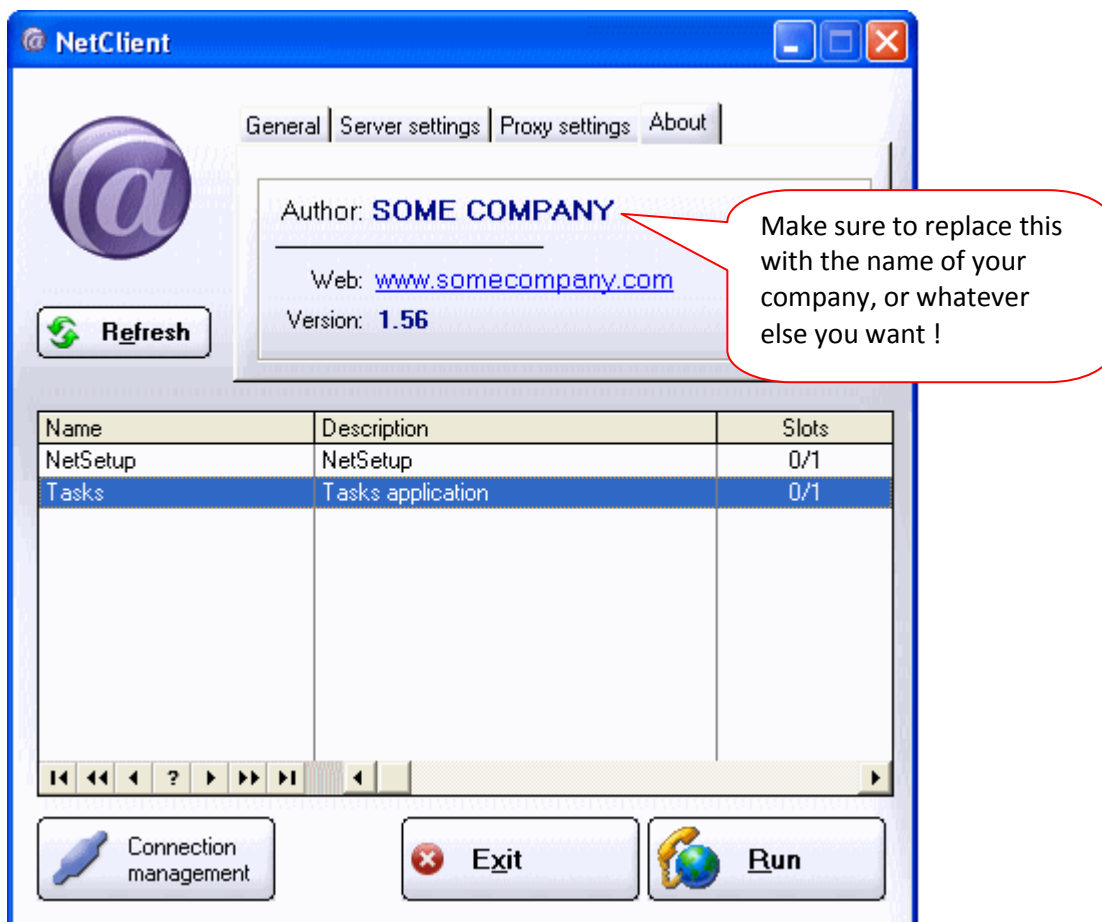
The third block is mandatory for every window and it goes in the end of the window ACCEPT LOOP.

## VI. Thin@ NetClient application tweaking

The Thin@ developer installation package contains the full source code of the default client application. If you install it, you will be able to change the client\NetClient.app application.


















Here is a list of some common features you might want to modify to better suit your needs:

### VI.1. Client window graphical user interface



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. Client-side 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) Called each time the window is opened:

```
AfterCreatingWindow
```

b) Called before each control paint process. If derived function returns 0 the library will skip the paint process for that control:

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

c) Called after each control paint process. If derived function returns 0 the library will skip the paint process for that control:

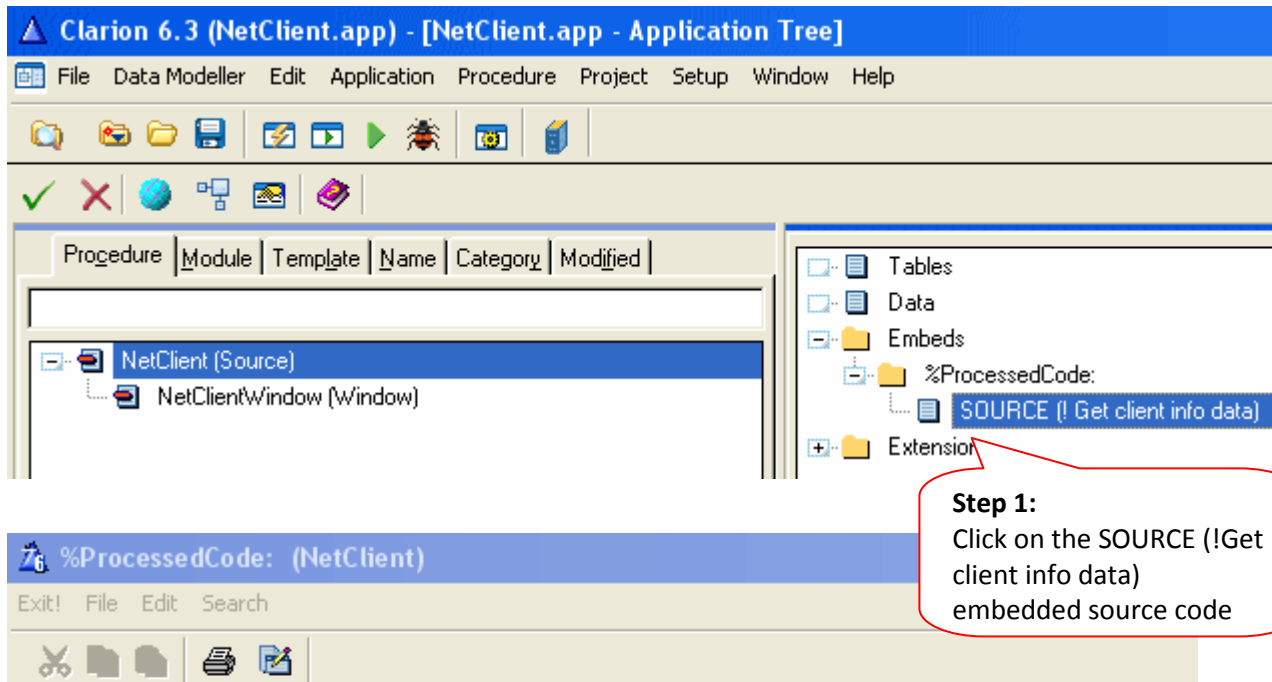
```
AfterPaintingControl (LONG Feq, BYTE Created=0, BYTE Last=0)
```

d) **TakeClientEvent (\*Window WindowHandle)** – called for each event generated inside an ACCEPT loop. If derived function returns 0 the library will skip sending the event to the server side.

```
TakeClientEvent (*Window WindowHandle)
```

### VI.3. Client multi-language support

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



**! 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, '. Allowed 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..')
```

**Step 2:**

Replace the system messages in english with their equivalents in another language

## VI.4. Tweaking compression and decompression routines

Thin@ uses two standard file compressor programs:

7zip - [www.7zip.com](http://www.7zip.com)

or

WinRar - [www.winrar.com](http://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
```

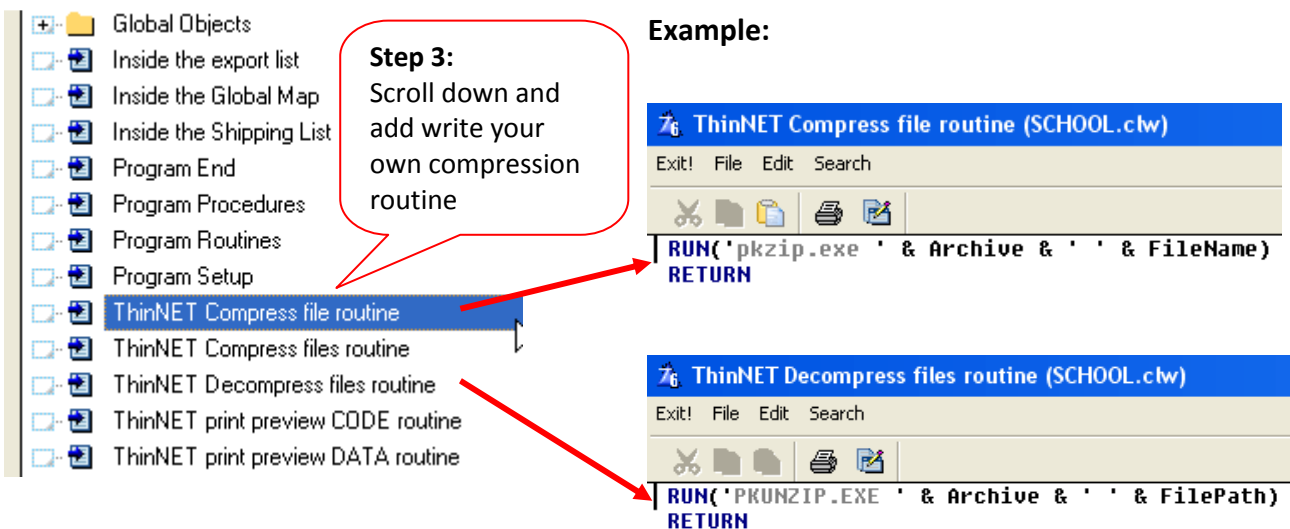
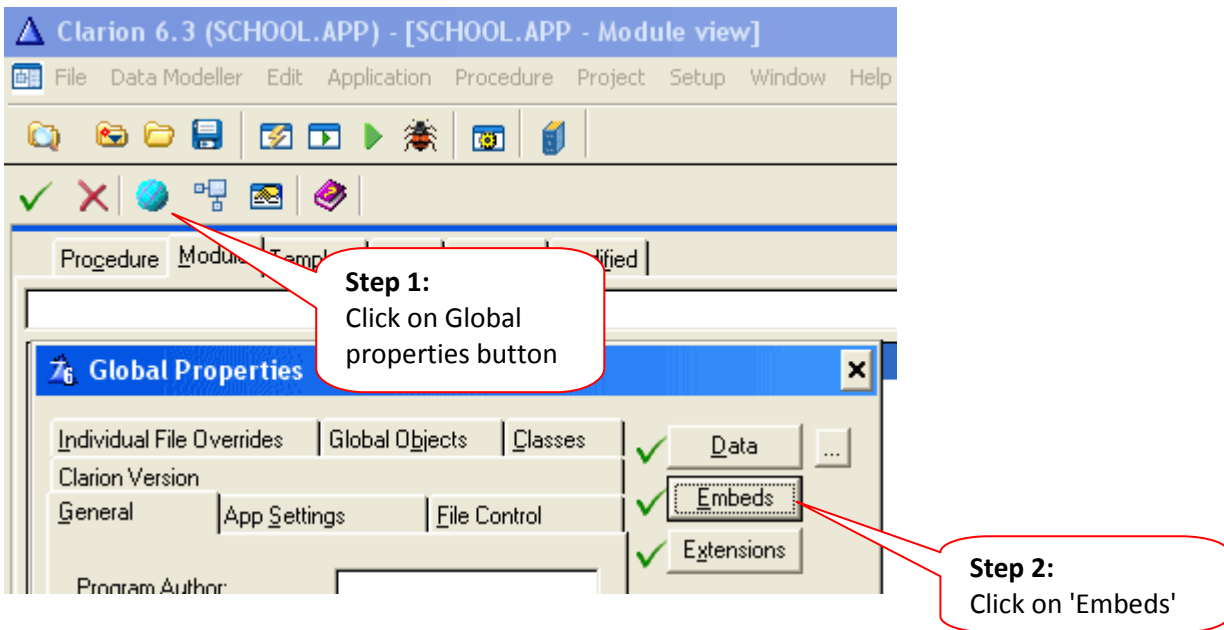
FileName – the name of the file you want to compress

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:



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/3<sup>rd</sup> 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).
```

```
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
```

-  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

**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!

### Step 1:

Make a shortcut for your application



### 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.

### 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.



### Step 5:

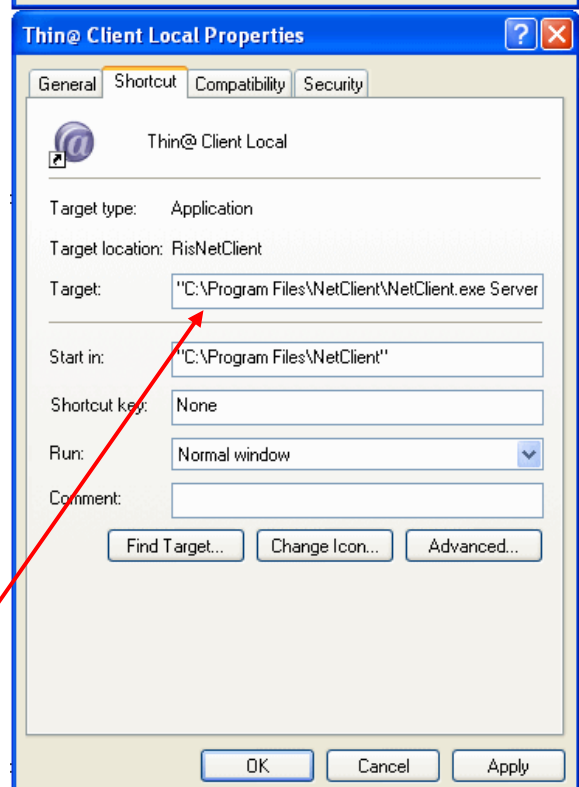
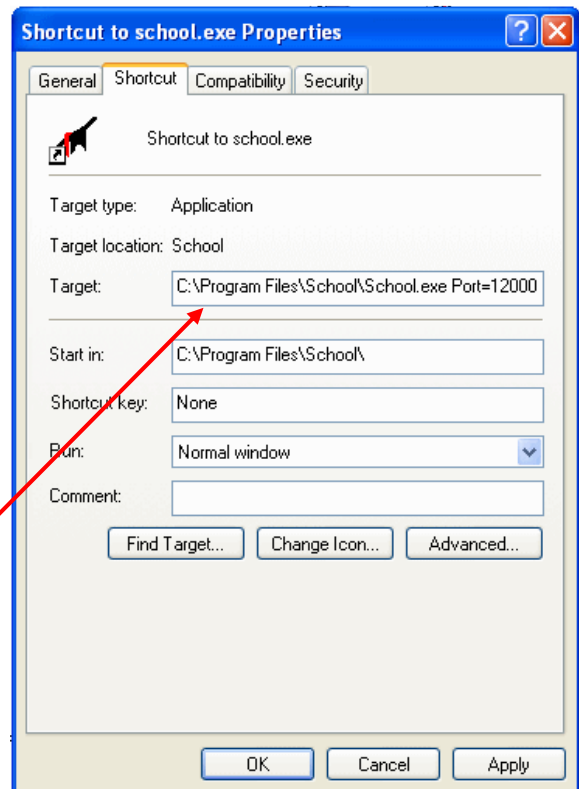
Right click on the shortcut icon and select Properties.

### Step 6:

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

### 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
```