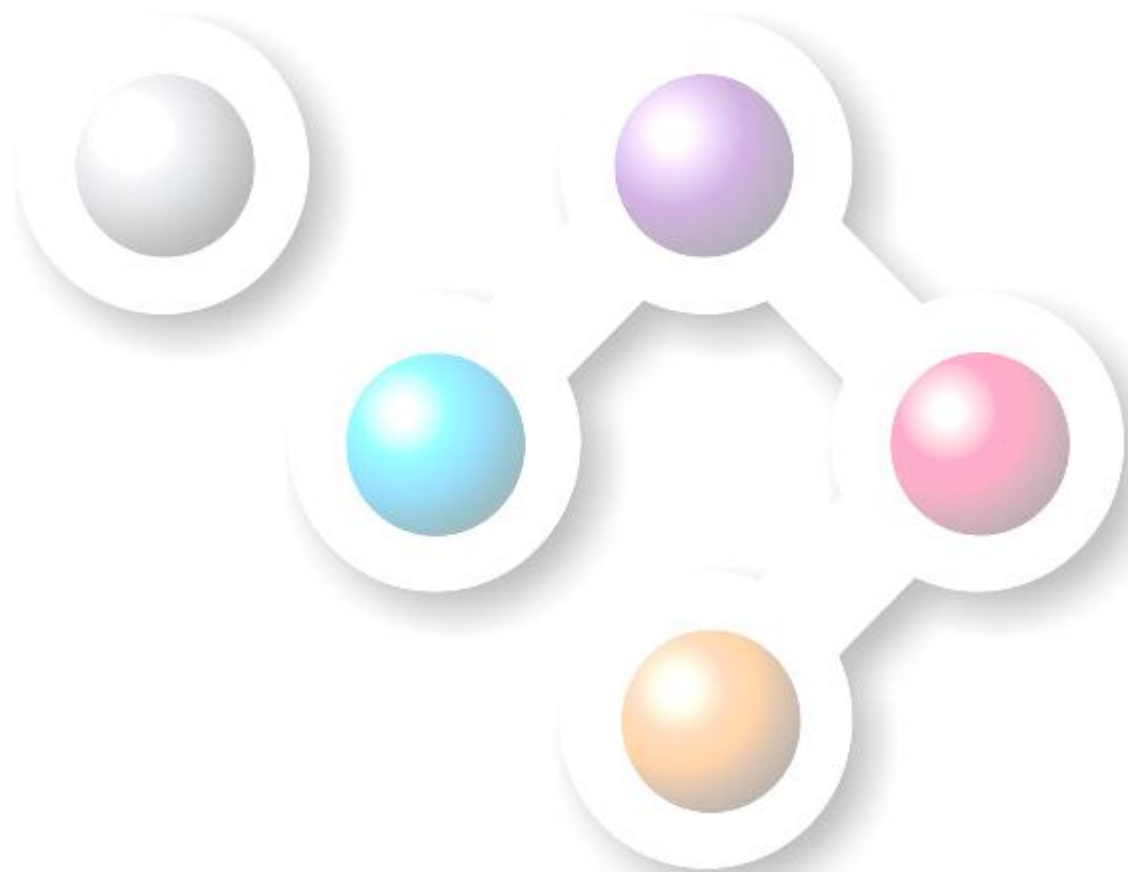


'Hydra4GL'™ Development Suite



'Hydra4GL' Phoenix Developers Guide
Version 4.3 – January 2008
Revision number 1.35



Phoenix Windows thin-client

Developer Guide

Anthony Davey

Technical Writer

Last Updated 02 January 2008

Phoenix

Developer Guide

‘Hydra4GL’

Second Edition

January 2008

Part Number: 005-043-135-008

Phoenix™ Developer Guide

Copyright © 2003-2008 Querix™ Ltd. All rights reserved.

January 2008, Part Number: 005-043-135-008

Published by:

Querix Ltd, 50 The Avenue, Southampton, SO17 1XQ, UK

Printing History:

December 2003: Phoenix V4.1 Beta2 version

May 2004: First Edition

May 2005: Updated for version 4.2

January 2008 Updated for version 4.3

Documentation written by:

Anthony Davey/ Sean Sunderland

Last Updated: 02 January 2008

Notices

The information contained within this document is subject to change without notice. If you find any problems in the documentation please submit your comments to documentation@querix.com . No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose without the express permission of Querix Ltd.

Other products or company names used within this document are for identification purposes only, and may be trademarks of their respective owners.

Table of Contents

About This Guide	1
<i>Who Should Read This Guide.....</i>	<i>1</i>
<i>Organisation of this manual.....</i>	<i>2</i>
<i>Conventions used in this book</i>	<i>3</i>
Typefaces and Icons.....	3
Numbered Instructions	4
Examples	4
What is Phoenix?	5
<i>What is Phoenix?.....</i>	<i>6</i>
Using Phoenix	7
<i>Connection Dialog.....</i>	<i>8</i>
1 - Connection Dialog	9
2 - 'HydraStudio'	10
3 - Run a .qxs File.....	10
4 - From the Command Line	10
ActiveX Controls & Advanced Features	11
<i>ActiveX controls.....</i>	<i>12</i>
<i>Using ActiveX controls.....</i>	<i>12</i>
Specifying the control resource	12
Defining the standard interface properties.....	13
Defining user properties.....	14
User objects as controls	14
Buttons.....	15
Background images	16
Adding a user object.....	16
Adding a background image.....	16
<i>Advanced Deployment Options for Developers.....</i>	<i>18</i>
Media	19
Links.....	19
Desktop	20
Controls	21
User Fonts	21
Index.....	23

About This Guide

Who Should Read This Guide

This manual is intended for use by developers who want to deploy 4GL applications using the Querix™ thin-client Phoenix™, to access databases. This guide contains only details that are specific to the Phoenix thin client. Information that is also relevant to other Querix thin clients, such as scripting, is contained in the Graphical Client Reference Guide. Details of Querix 4GL are contained in the volumes of the Querix 4GL Reference.

Readers are advised to select their reading carefully. Sections aimed at 4GL/C application developers will assume a working knowledge of 4GL and C programming.

This guide describes the features and capabilities of Phoenix; the thin-client that forms part of the 'Hydra4GL' compiler suite.

If you have any comments on how to improve this guide, please address them to documentation@querix.com.



Full installation instructions are provided in the 'Hydra4GL' Getting Started Guide.

Organisation of this manual

Chapter 1

A brief overview of the Phoenix advanced thin-client.

Chapter 2

Using Phoenix; how to use Phoenix and run 4GL programs from the 'Hydra4GL' GUI Server.

Chapter 3

How to use ActiveX controls and manipulate BLOBs.


Conventions used in this book

Typefaces and Icons

A monospace typeface is used for code examples and fragments, or command line functions.

The same typeface is used in bold for emphasis.

It is also used in italics to represent variables such as file names, or operating systems.

	<p>This icon indicates a suggestion or note, which may be important.</p>
---	--

Code fragments are generally shown like this, using the monospace typeface:

```

database cms
# lines of code with a hash symbol at the
# beginning are comments, and are not read as
# code
main
    display "Hello World"
    lines of code that should be read as \
continuous will have a backslash(\) at the end
end main
    
```

Where a program needs a code string to be typed as a continuous line but, for neatness or lack of space on these pages, this is not possible, a backslash (\) is used as the last character on the code line. A backslash indicates, in 4GL, that the following line of code should read as if it is joined to it.

Code examples shown in this document may be used within any Querix applications – no special permission is required.

Numbered Instructions

Within numbered step-by-step instructions, buttons to be used on a GUI are highlighted in bold. For example:

- 1) Click the **OK** button.

When a numbered step causes a new action to take place the change will be described below the step in a paragraph like this.

If further information is needed, about a particular instruction, such as alternative courses of action, this will be provided in a paragraph like this one.

Examples

Throughout this document references are made to a series of examples. These examples all refer to the CMS Demonstration or other example applications that are packaged with 'Hydra4GL'.

CHAPTER 1

What is Phoenix?

Phoenix is the Querix advanced thin-client for Windows® environments that allows users to render and use 4GL applications in a familiar style.

This guide should be used to become familiar with the Phoenix thin-client application.

A standard installation of Phoenix is very simple and should be familiar to anybody having previously installed software on a Windows PC. Full details of how to install Phoenix on Windows and Unix®/Linux® platforms can be found in the 'Hydra4GL' Getting Started Guide.

What is Phoenix?

Phoenix is an advanced thin-client that enables 4GL applications, developed or compiled using the 'Hydra4GL' compiler suite, to be displayed in a format that is familiar to users of PCs with a Windows operating system and related software packages.

More specifically, Phoenix has been developed to make database usage as familiar and user-friendly as possible for Windows users. Phoenix interacts with 4GL programs to enable easy access to most of the common database systems including:

- Informix® (v4.1 or later)
- Oracle® (v8i or later)
- IBM DB2™ (v8.1 or later)
- Microsoft® SQL Server™ (v7 or later)
- Pervasive® (v8.5 or later)
- PostgreSQL
- MySQL
- Other RDBMSs via ODBC without Dynamic SQL translation

Phoenix is supported on most of the recent Windows OS platforms including:

- Windows 2000 Series
- Windows XP
- Windows 2003 Series

Phoenix can be used with existing applications with no change to existing source code. All that is needed is that the application be compiled using 'Hydra4GL'.

CHAPTER 2

Using Phoenix

This chapter focuses on the basic use of Phoenix, and demonstrates how to run a 4GL program from the 'Hydra4GL' listener service using Phoenix.

Examples used are based on the CMS Demonstration and other example applications, which are packaged with 'Hydra4GL'.

Connection Dialog

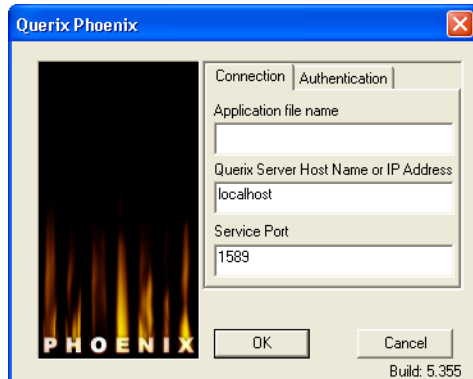
There are four ways in which to connect to or start an application.

- 1) Start the application via the Windows Start menu. Click Start, choose All Programs, then Querix and select Phoenix. A connection dialog box appears, allowing the operator to enter the connection parameters for the application server, server port and application name. There is also an Authentication tab in which you are able to enter your user name and password.
- 2) If you are developing programs using 'HydraStudio', you can start the application via the Run Program dialog. Double-click the icon for a built program in 'HydraStudio', select the Run with Phoenix radio button and click the Run button.
- 3) Run a .qxs file. Windows manages and associates file extensions with programs. After Phoenix is installed, all files with the extension .qxs are associated with Phoenix.
- 4) Start the application from the command line.

1 - Connection Dialog

The connection dialog provides an interface to calling and running 4GL applications within Phoenix.

In this connection dialog it is possible to specify a host, port and application to be connected to. If there is an appropriate script on the server with the same name as the application (except the .qxs file extension) and located in the same directory as the program, it will be downloaded and used as the application starts.



The connection dialog

The following features:

- Access to the Phoenix object-name information facility, which is particularly useful for script file configuration
- Access to the connection console window, which reports errors and problems in the script file, as well as showing all 4GL DISPLAY (not "DISPLAY AT") statements

2 - 'HydraStudio'

In 'HydraStudio', once a program has been compiled and built, double-click the corresponding program name in the project browser. The Run Program dialog is displayed. Choose the Run with Phoenix radio button and click the Run button.

3 – Run a .qxs File

The Phoenix installer associates all files with the extension .qxs with Phoenix. This means double clicking a .qxs file from the file browser or linking it to a short cut or start program menu item will launch Phoenix with the corresponding application connection information. This requires the .qxs file to include the necessary connection information.

default.host: <host name or IP Address of the Querix Application Server>

default.port: <Querix GUI Application Server Port>

default.command: <Application name>

4 – From the Command Line

Phoenix can be invoked to run specific applications from the command line if no other method is possible. Use the command:

```
phoenix -h host -p port application
```

Where *host* is the name of the application server, *port* is the number of the connection port, and *application* is the name of the application executable file to be run.

Phoenix can also be started from the command line by using as an argument the corresponding .qxs file which includes the connection properties.

```
Phoenix c:\start.qxs
```

CHAPTER 3

ActiveX Controls & Advanced Features

This chapter details the use of Microsoft ActiveX controls with Phoenix; it also covers the use of user-level access control.

ActiveX controls

Prior to version 4.2 you would have to have had user ActiveX controls to create GUI controls such as radio buttons, image controls, and checkboxes if you did not want to modify the 4GL source code. With v4.2 and beyond, all common internal GUI client controls can be created either with form widgets or by using the scripting language. This means ActiveX controls are now only required if you want to develop your own graphical controls.

Phoenix, and the Java client Chimera, include a few sample ActiveX and JavaBean controls with identical control and property names for demonstration purpose.

Microsoft ActiveX technology is a standard mechanism by which arbitrary controls can be interfaced to user programs. ActiveX provides mechanisms for all communication between the application and the user control.

By using ActiveX, any object within a Phoenix window can be changed into a custom control. One of the easiest ways to write an ActiveX control is by using Visual Basic (VB). VB provides a standard toolkit for writing such controls, which implements much of the required functionality with a minimum of effort.

Using ActiveX controls

There are three steps required to use ActiveX controls with Phoenix:

- Specifying the control resource
- Defining the standard interface properties
- Defining any user properties

Specifying the control resource

The first step in using an ActiveX control is to request that Phoenix load this control in place of the control it would normally use to display the object. To do this the control resource must be set; the value of this resource is the ActiveX control's object name. The object name is a name of the form "QxControls.qximage", and is used by the ActiveX framework to determine what control to load. In the case of Visual Basic controls this name will be the name of the project/control.

For our example we are going to use the combo box demonstration control. This is described in more detail later in this section. The control is set as follows:

```
?.?.?.customer.state.control: QxControls.QxComboBox
```

Defining the standard interface properties

The next thing is to specify the interface to the custom control; there are a number of control parameters that keep track of the control state, these are summarised below:

Property	Default Value	Meaning
modifyProperty	modified	Property used to keep track of whether the control has been modified. This is both set and read by the framework.
valueProperty	text	Property used to store the actual value the control is representing. This is both set and read by the framework.
enableProperty	enabled	Property used to determine whether the control should be drawn as being enabled. This is set only by the framework.
lockProperty	locked	Property used to determine whether the control should allow input. This is set only by the framework
colorProperty	foreColor	Property used to determine the text colour of the control. It is set only by the framework.
backColorProperty	backColor	Property used to determine the background colour of the control. It is set only by the framework.
cursorPositionProperty	cursorPosition	Property used to determine the current cursor position within a field. Support for this property is not required, but offers more consistent behaviour if it is implemented. It is both set and read by the framework.
visibleProperty	visible	Property which determines whether the control should be visible. It is set only by the framework.

The values of these interface properties are all case insensitive. In the case of the demonstration Combo Box control, the default values will suffice for these properties, so none of them need to be set.

Defining user properties

In order to properly configure a control, it is often necessary to set other values associated with it. For example, to configure a checkbox, we must tell the checkbox the text value that represents checked, and the value that represents unchecked. To do this, Phoenix provides the facility to set any control properties to desired values at the time the checkbox is created.

User properties are defined by setting resources on the object properties. Each resource is assumed to correspond to a property on the ActiveX control, which Phoenix will attempt to set at creation time. In general there is no side effect to attempting to set a property that does not exist. However, if the control generates an exception as a result of setting a property to an invalid value, the entire control creation process will be aborted. This is most notable with the control `QxControls.qximage`. This is the default image control as used in previous sections.

In the case of a user combo box, you need to configure the list of possible values the combo box can take. To do this, the `SelectionList` resource must be set:

```
?.?.?.customer.state.properties.options: \  
Alaska,Alabama,Arkansas
```

Assuming the `.OCX` file for your combo box user control has successfully been installed, you can now run the Phoenix demonstration with your new script, and instead of getting the old state input box, a drop down list of possible states is presented.

To re-use this selection list of US States in other contexts it would be best to use a template:

```
#Define the templates  
!state_sel.properties.options: \  
Alaska,Alabama,Arkansas  
  
!state_sel.control: QxControls.QxComboBoxDemoctrl.combobox.1  
  
# Use the templates  
?.w_invoice.?.customer.state.template: !state_sel  
?.w_contact.?.another.state_field.template: !state_sel
```

User objects as controls

Elsewhere you will see how to replace a user object with an ActiveX control. This is a very frequent usage for ActiveX controls, as it allows the application to create background images, or add standard buttons to a page on a form-by-form basis. However, buttons and images differ from each other in one notable respect; buttons expect input from the user (mouse clicks), images do not.

ActiveX controls have two states, an active state, in which they expect input and will process it accordingly, and an inactive state, in which they do not. In the inactive case they use up significantly fewer system resources, since the only procedure the application has to ask them to perform is to draw

itself. Phoenix creates most controls as inactive objects until such time as they actually become active. For ActiveX controls involved in an input statement this is not a problem, since as soon as Phoenix receives the request to start inputting from the control they become active. For user objects, however, there is no such procedure.

As such when you require a user object to be a control which can accept input from the user, it is necessary to request Phoenix to make it active. This is done by setting the active resource:

```
guidemo.Screen.userObject.button: Button
guidemo.Screen.?.button.active: true
guidemo.Screen.?.button.control: \
QxControls.QxImageButton
guidemo.Screen.?.button.properties.key: F1
```

Note, that setting the active property may have unexpected side effects for Image objects. In particular when the image is made active, the order in which it is drawn will be changed. This means that if you have an active image object covering your entire window (as a background texture), then none of your labels will appear.

Buttons

Like adding an image, embedding button controls into your application makes use of the native framework. Querix provides a default button control for use in your Phoenix applications. This control can perform one of two functions - either return a single key press, or execute a command on the client platform (i.e. spawn a given application).

The first step towards creating a button, much like adding an image, is to declare the user object. This is done in the same way for buttons as for other images to be used. As an example, in the guidemo GUI script file: `gd_71_userobject.qxs` you will see the code:

```
guidemo.w_control_demo.f_controls.userObject.\
SmallBackgroundImage: MySmallBackgroundImage
```

Using the same format, it would be possible to create a button in guidemo called `Mybutton` with the following code:

```
guidemo.w_english.f_a_english.userObject.Mybutton: Button
```

Note that the string resource "Button" is the text string displayed in place of the button. Now the object has been declared, its key properties can be set:

```
guidemo.w_english.f_a_english.Mybutton.control:
QxControls.QxImageButton
```

GUI Script File `gd_63_button_replacement.qxs` creates a button, 12 characters wide, within the 'simple_button' template, by using an ActiveX control for Phoenix with the code lines:

```
# Create and assign a template
```

```
guidemo.w_color_chart.?.Input.template: !simple_button

# Assign the width property
!simple_button.width: 12

# We replace 4gl Prompt field with the button
# For this, we will move the button horizontally
guidemo.w_color_chart.?.Input.X: 66
```

Background images

Background images could be either corporate logos, or background textures. They can be used to enhance the appearance of an application.

Both of these options require the use of the **user object** component. A user object is an object that is created by the user in the script file rather than the back-end client. User objects can be created either when a window is created, or when a form is placed in the window. By default a user object appears as a simple text label, and this can then be placed with a user control. In Phoenix this would be an **ActiveX control**.

Adding a user object

Adding a user object to a window requires a number of directives. Firstly, the client must be instructed to create the user object; then commands need to be added that determine the size and position of the object within the window.

```
# The value of the userObject resource is the
# text string which will be displayed
cms.Screen.userObject.picture1: Picture1
#Set the location and size of the object
cms.Screen.userObject.picture1.x: 0
cms.Screen.userObject.picture1.y: 0
cms.Screen.userObject.picture1.width: 60
cms.Screen.userObject.picture1.height: 3
```

This code sample creates an object anchored in the top left corner of the window (with x and y coordinates of 0), of size 60 characters wide and 3 lines high.

Adding a background image

Now that a user object has been added, it must be replaced with a control which draws graphics:

```
cms.Screen.userObject.picture1.control: \
Querix.Image.1
cms.Screen.userObject.picture1.protected: true
cms.Screen.userObject.picture1.active: false
cms.Screen.userObject.picture1.properties.\
picture: c:\mypics\mylogo.bmp
```

This instructs the client to load the image from the file C:\mypics\mylogo.bmp. If the client cannot find the image file specified it will revert back to the old text string, displaying the message "Unable to load Active/X control".

Advanced Deployment Options for Developers

In its simplest form the installation of Phoenix is suitable for installations to be used by developers, in a development environment. Phoenix is intended for use by end users, and it needs to be configured to meet their specific needs. Those developers that will be configuring Phoenix for distribution to end users can set up the advanced thin-client with the use of a number of configuration files, which are described in this section.



This advanced deployment version of Phoenix is only available with an .msi version of the installation program. This version is only available for registered customers on request from Querix.

As well as the advanced thin-client itself, it is possible to allow additional media to be transferred from the server to the client machine. It is also possible to configure the destination platform. This is done using a number of install configuration files, which have the .qxi extension.

If you are using Phoenix with an application that is resident on your machine you will need to install all of these configuration files. If you are using Phoenix on your machine to access an application stored on a remote server you will only need to the last four listed here.

- Media – Define additional files (such as scripts, images etc.) to be copied during the installation process
- Links – Application links to be created within the start menu
- Desktop – Desktop shortcuts to be placed on the target platform
- Controls – User controls to be copied and registered on the target platform
- Fonts – User fonts to be copied and registered on the client platform

Media

Supplementary media files required for the client installation should be listed in the file media.qxi. Media files can be of any format, but are usually used for client scripts and image files. 'Hydra4GL'4.1 and later support server side resource distribution which Querix strongly recommends that you use. Ideally you should only use client side resources for desktop and start menu icons.

The format of the file is to list one file name per line. For example:

```
images\mainlogo.bmp
images\smalllogo.bmp
resources\myprog.exe
```

The media files to be copied should reside in the directory from which the installer is invoked. The list of files will be hierarchically copied to a media subdirectory of the Phoenix installation directory. Therefore, the files in the above example (assuming the installer is located in E:\) will be copied in the following manner:

Base source directory	Base destination
E:\	C:\Program Files\Phoenix

File Source File	Destination
E:\images\mainlogo.bmp	C:\Program Files\Phoenix\media\images\mainlogo.bmp
E:\images\smalllogo.bmp	C:\Program Files\Phoenix\media\images\smalllogo.bmp
E:\resources\myprog.exe	C:\Program Files\Phoenix\media\resources\myprog.exe

Links

The file links.qxi contains directives for those additional application links that are to be placed on the Windows Start menu. The format of the file is separate directives on individual lines, which should each take the following format:

```
link_file;folder;link_name;link_icon
```

Where the link_file parameter is the file to which a link is being created; and the link_icon parameter is the icon file which will be used in the Start menu. The folder parameter specifies the program group under the Start menu, into which the program link is to be installed. The link_name parameter specifies the optional text to be used in place of the icon.

The link file is assumed to be in the media subdirectory (or in one of its subdirectories) of the base installation directory, so it is not necessary to specify the 'media' prefix to the file name if the file is one of those listed in the media.qxi file. Similarly, the icon file specified, if not a system icon, is assumed to be located within this directory. Therefore, a sample line of this file might look like this:

```
cms.qxs;Querix Demo;cms Demo Application;\  
phoenix.ico
```

This will direct the installer to create the link cms Demo Application.lnk within the Start menu under the folder 'Querix Demo', which will link to the file
C:\Program Files\Phoenix\media\cms.qxs.

To expand on this example: links to files located in subdirectories only need to have the subdirectory name used as a prefix to the file name. For example, in the following section of media.qxi, to make a link to the script named orders.qxs.

```
...  
scripts\orders.qxs  
images\dollar.ico  
...
```

To enable this link, would simply need an entry within the links.qxi file of the form:

```
scripts\orders.qxs;Customer Orders;images\  
dollar.ico
```

Desktop

To place a desktop link to your application, the file desktop.qxi can be used. This file lists the desktop shortcuts to be placed on the target platform, using the following format:

```
file_name;icon_text;icon
```

The file_name parameter is that of the physical file to link to. If this is a file being installed by the client, the file must also have an entry in the media.qxi file. The icon_text parameter is that of the text label to be applied to the desktop item, and the icon parameter is that of the icon file to be used for the Desktop item.

If the icon file is supplied by the user, then this must also be listed in the media.qxi file.

Controls

The controls.qxi file lists the user controls to be installed on the client machine. Each control will automatically be registered on the destination system.

The files listed in this script will be copied to the local system, so it is not necessary to list controls in the media.qxi file. Below is an example of a controls.qxi file:

```
listbox.ocx  
drilldown.ocx
```

User Fonts

It may be necessary to register the font files on the client system, if there are references to any non-standard fonts in the configuration script. Querix allows the file fonts.qxi to do this. This file should list any fonts that are to be registered on the target platform.

```
file_name;font_name
```

The file name parameter references the TTF font file to be registered (like all other user files, this should also have an entry in media.qxi). The font_name parameter should be the name with which the font is to be installed and registered.

A	
ActiveX	
control resource, specifying	12
controls, user objects as.....	14
controls, using	12
controls, with input statements.....	15
description of	12
interface, defining properties	13
placing labels with	16
user properties, defining	14
Advanced Deployment Options	18
application	
starting with Phoenix	8
B	
button	
controls, default.....	15
properties, setting.....	15
C	
combo box	
values list	14
configuration	
of destination platform	18
controls	
in configuration files.....	18
D	
databases	
compatible	6
E	
end users	
deployment options for	18
errors	
in script files.....	9
F	
file	
fonts.qxi.....	21
links.qxi.....	20
media.qxi	20, 21
files	
configuration, for end users.....	18
icon, default location of.....	20
image	19
media	18, 19
script	18
fonts	
in configuration files.....	18
user	21
I	
images	
background, adding	16
installation	
guide.....	1
overview.....	5
L	
links	
in script files	19
M	
media	
accessing with script files	19
O	
object	
user, declaring	15
objects	
image, setting as active.....	15
size & position	16
user created	16
user, adding	16
user, as controls	14
P	
Phoenix	
running from command line.....	10
running from Hydra Studio	10
running with .qxs file.....	10
platforms	
supported	6
R	
resource	
SelectionList	14
setting user object as active	15
S	
script	
download from server	9
files, errors in	9
selection list	
as a template.....	14
statements	
DISPLAY	9
T	
templates	
selection list as	14