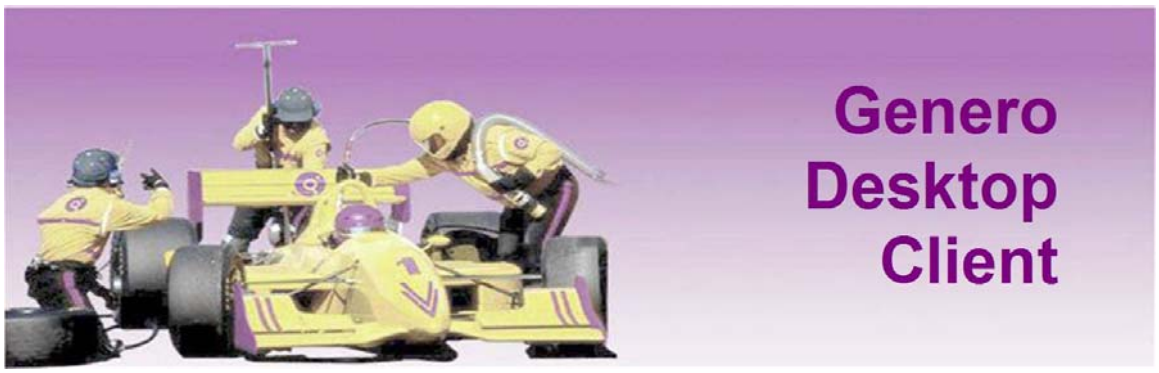



# User Guide
# Version 2.11

**Note:** This documentation is for Genero 2.11. See the corresponding on-line documentation at the Web site **http://www.4js.com/online_documentation** for the latest updates. Please contact your nearest support center if you encounter problems or errors in the on-line documentation.

# Table Of Contents

# Overview

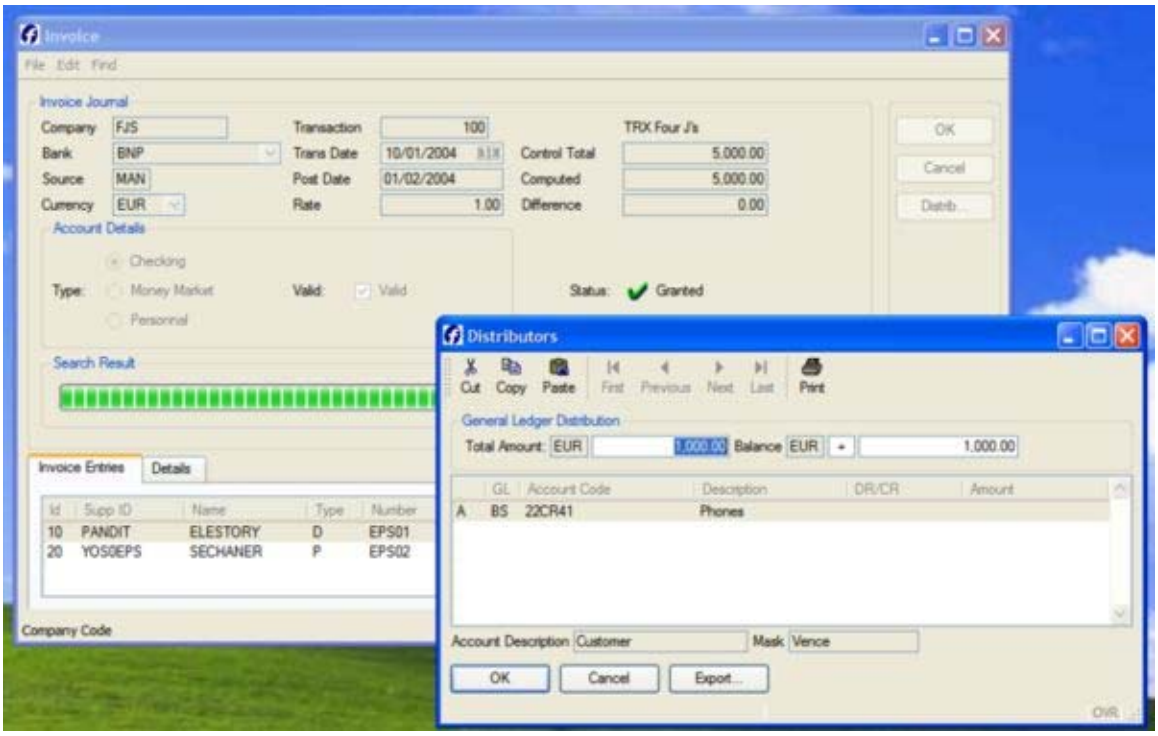
The **Genero Desktop Client** (**GDC**) is a graphical Front End for the Genero Runtime System. The Genero Desktop Client is multi-platform and can run under:

- Windows systems
- Mac OS X (10.2)
- Linux (X11 systems)

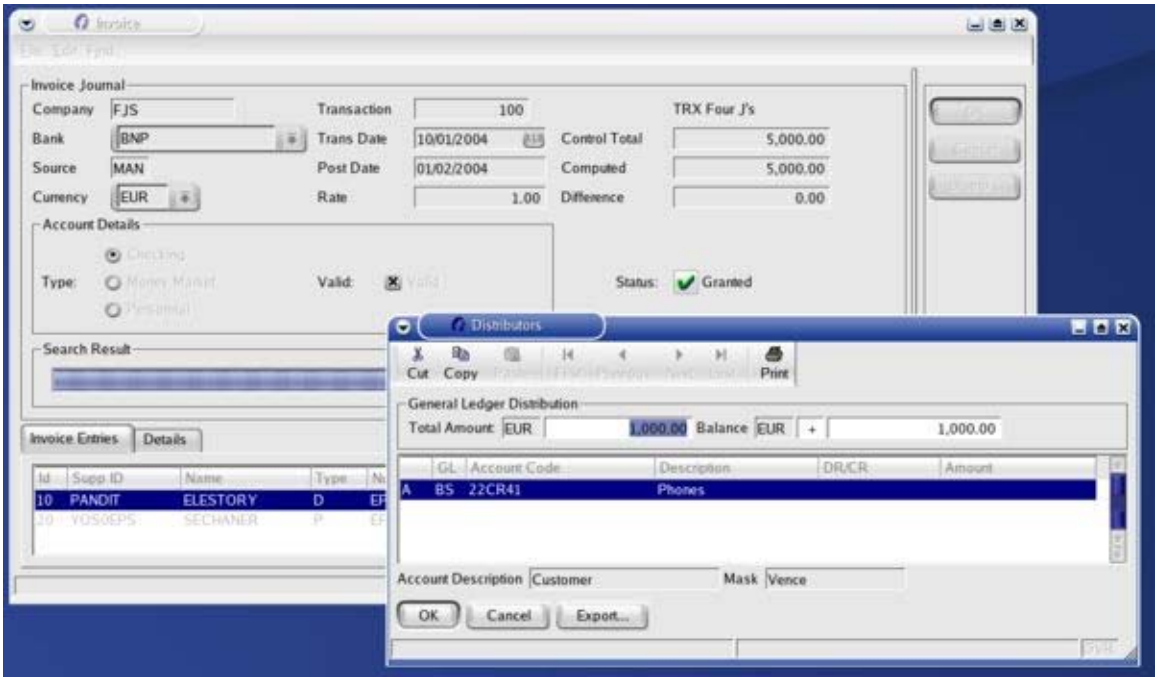
The Genero Desktop Client can also be embedded in an HTML page using Active X. This solution works on Windows systems using the Internet Explorer browser.

The following screen shots show the same application running on each of the three supported platforms.

### Windows version:



## Linux version (using dedicated KDE 3.3 version):




## Mac OS X version:

# Installation

The installation process installs the Front End on the Workstation.

**Topics**

- System Requirements
- Prerequisites
- Windows Systems
- Mac OS X
- X11 Systems

---

## System Requirements

The client system must meet the following requirements:

**Windows systems:**

- Windows 98, Windows ME, Windows NT4, Windows 2000, Windows XP
- Windows TCP/IP system installed

**Linux systems:**

- TCP/IP system installed
- kde's kmfclient tool if you want standard function call "shellexec" to work.
- KDE 3.3 (or higher) for the KDE specific version.

**Mac systems:**

- Mac OS X 10.2
- TCP/IP system installed

## Prerequisites

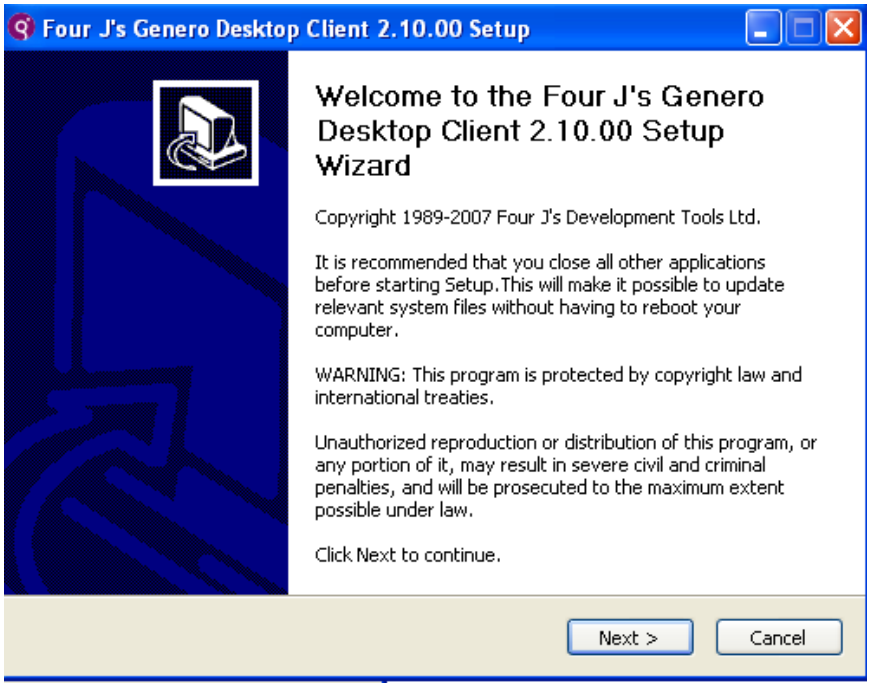
Genero Desktop Client will work only with a Genero Runtime System, and is not compatible with Four J's BDL Runtime System.

---

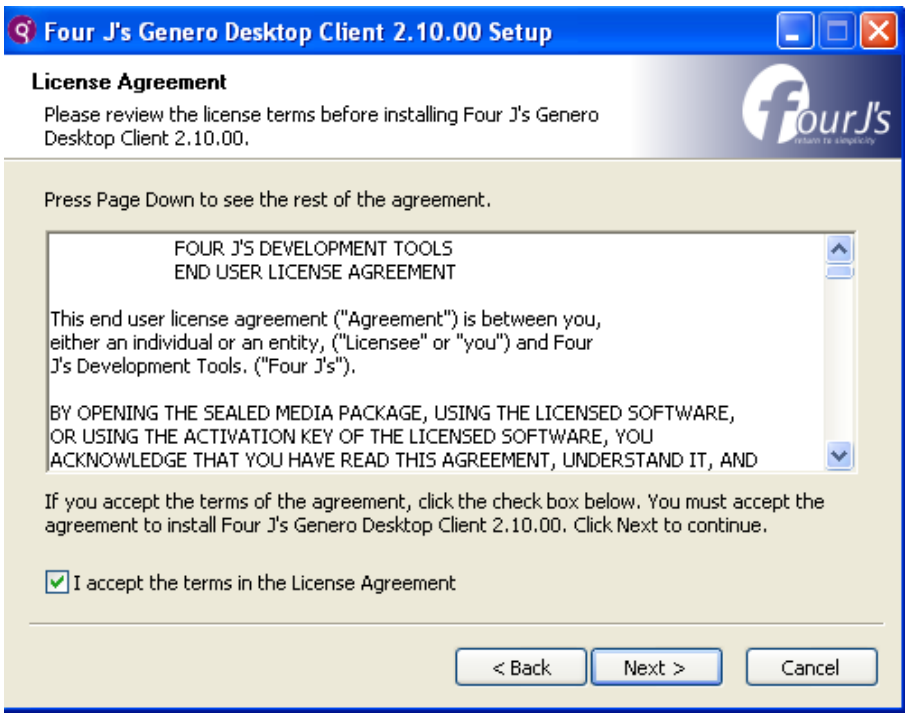
## Windows Systems

**To install:**

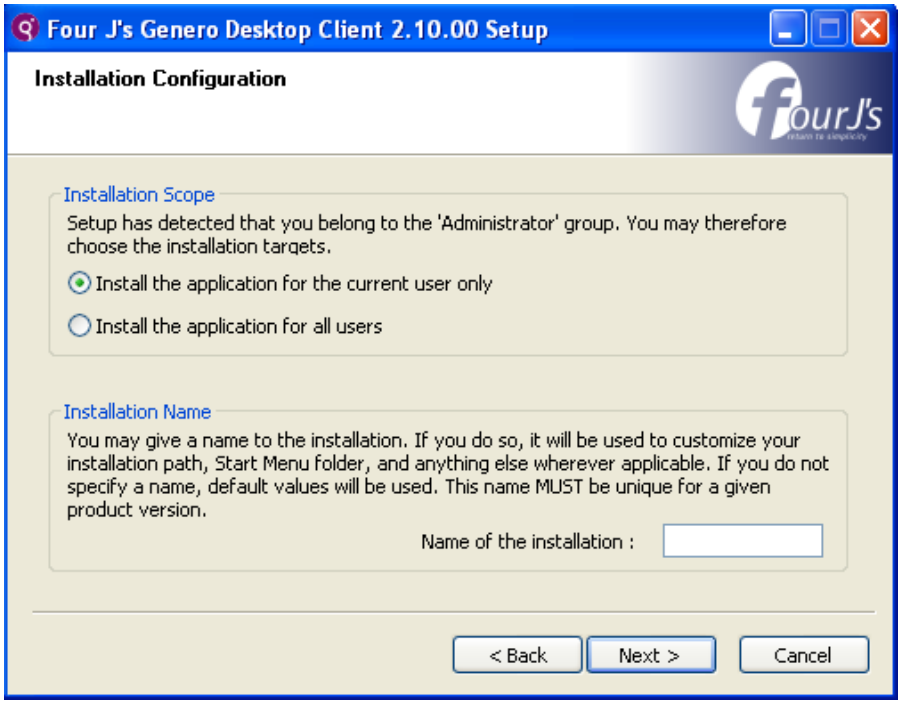
1. Close all running applications and execute the installation program.



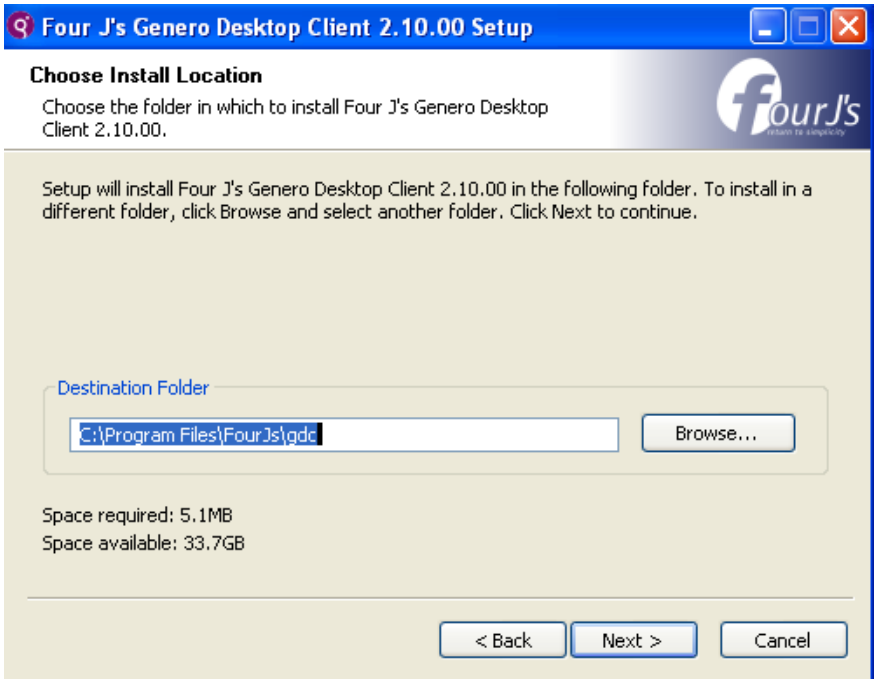

2. Press 'Next' to start the installation.



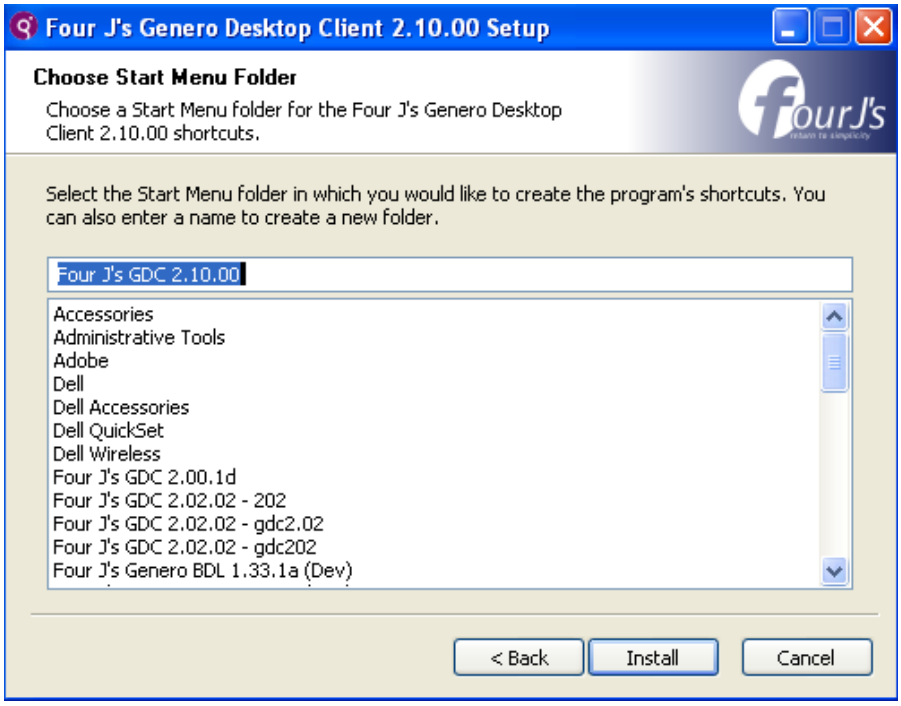
3. If you accept the agreement, check 'I accept the terms in the License Agreement' and click 'Next" to continue.



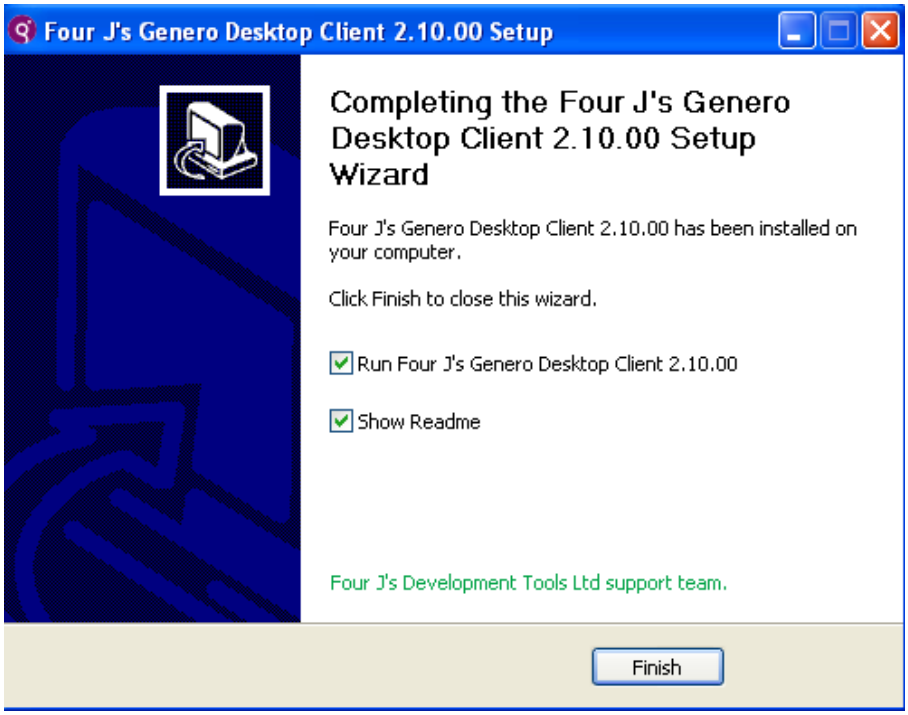

4. Depending on your system, the installation system will ask you if you want to install GDC for all users or for the current user only.



5. Now select the folder in which you want to install GDC.




6. Select the StartMenu folder in which GDC shortcuts will be set.
7. All files will be copied ; the installation is now complete:



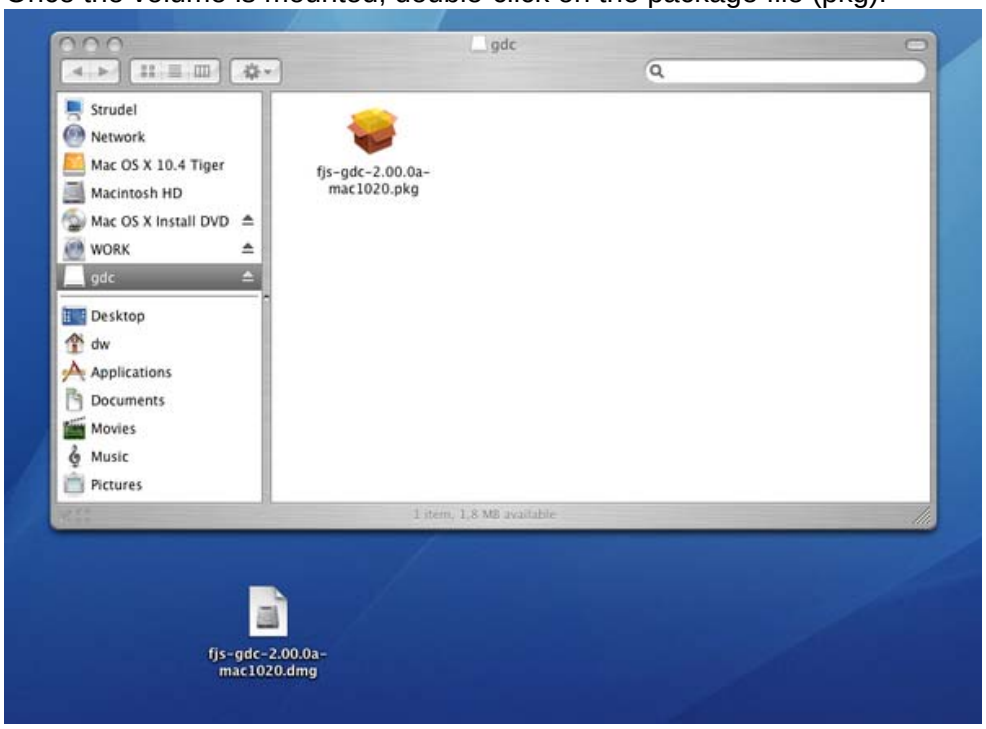

You can now run GDC.

## Mac OS X

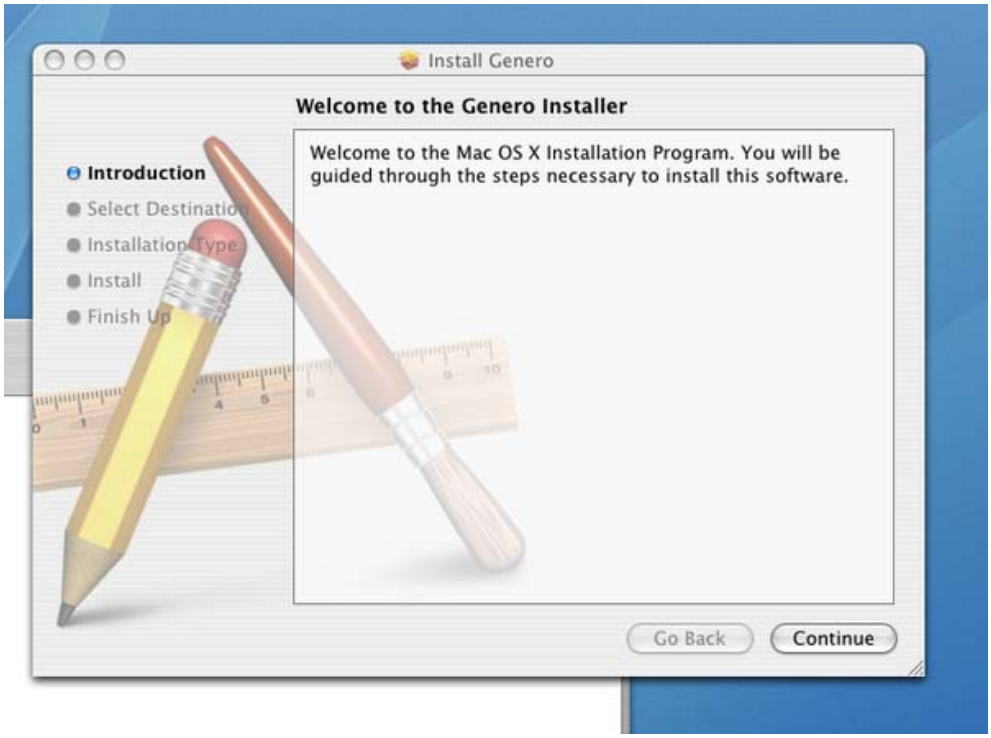
**To install:**

1. Close all applications:

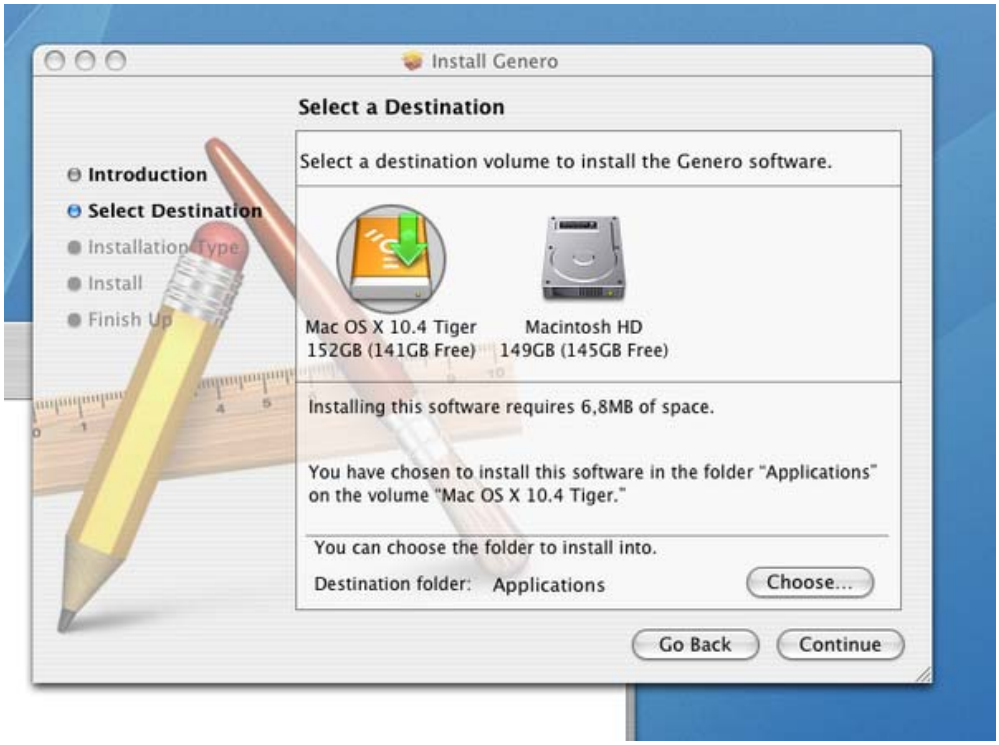

2. Double-click on the drive image file (.dmg). The image file will be mounted by Mac Os X.
   Once the volume is mounted, double-click on the package file (pkg):



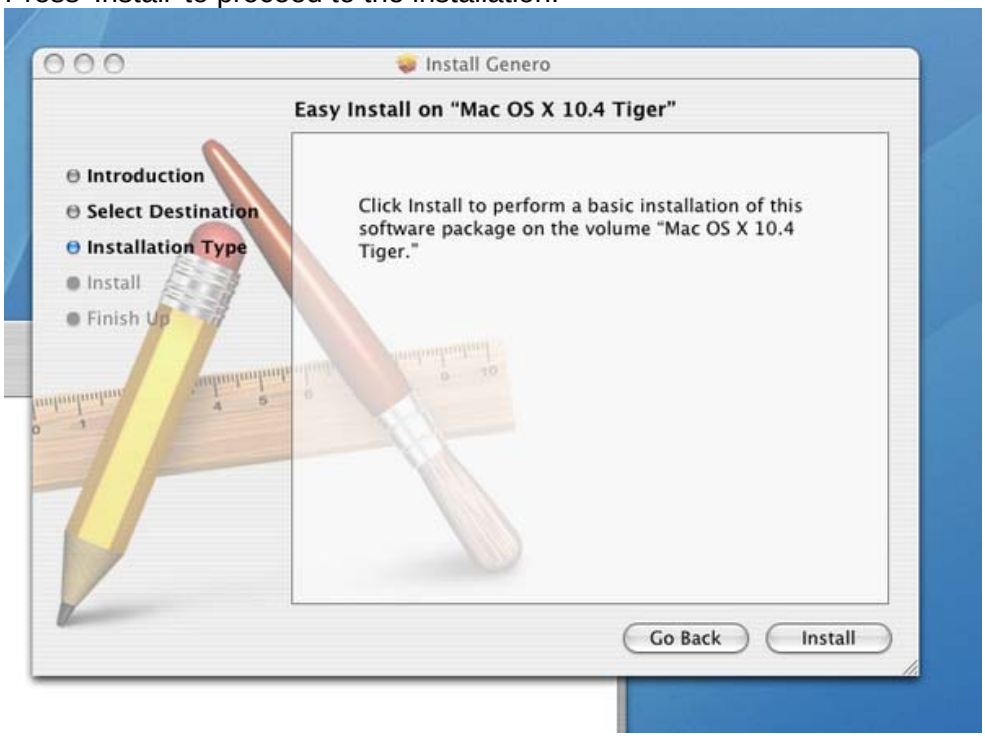
3. Press 'Continue' to start the installation:




4. Select the drive, and then the Destination Folder (by default 'Applications'), in which you want to install the GDC.
   Press 'Continue' to continue the installation:

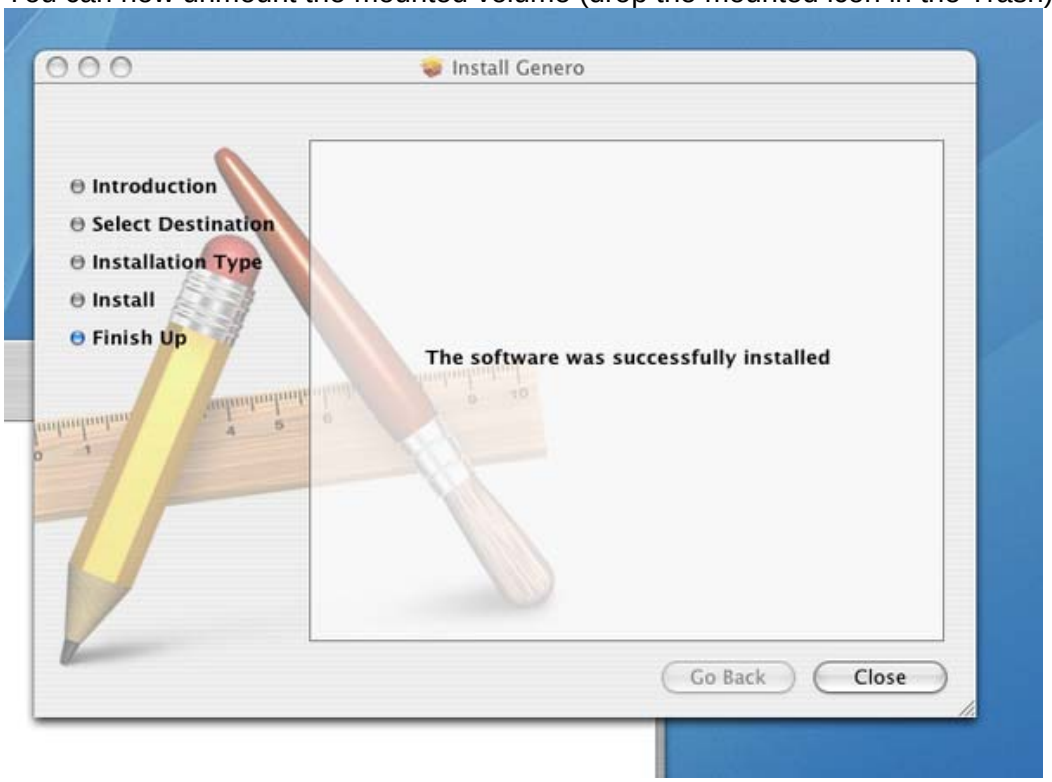


5. Press 'Install' to proceed to the installation:

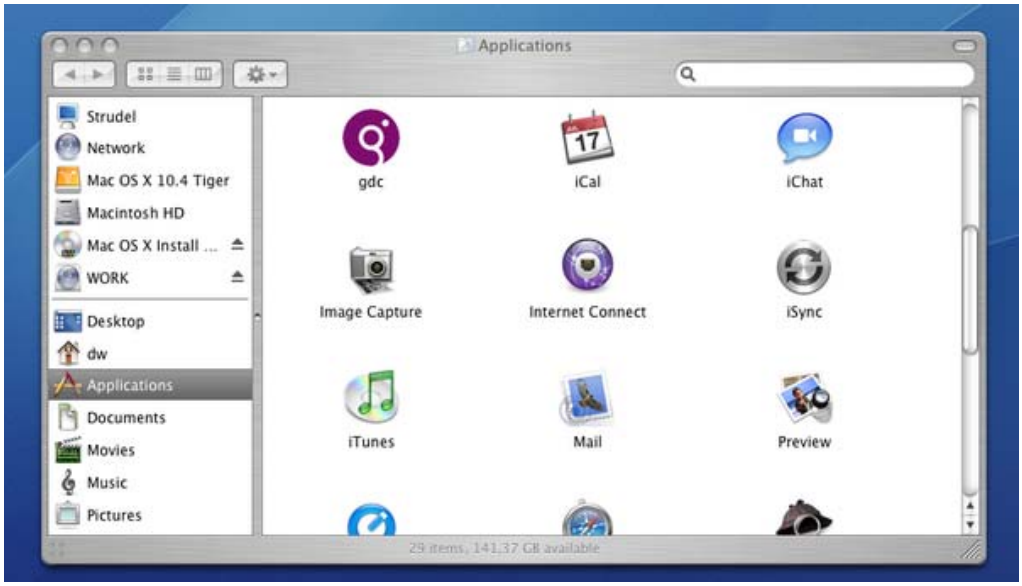



6. Enter the root password. Administrator rights are needed for RLOGIN and to install GDC in the *Application* directory.

7. Now GDC is installed on your Mac OS X. Press 'Close' to finish the installation. You can now unmount the mounted volume (drop the mounted icon in the Trash)




8. To start GDC, double-click the gdc icon in the Destination Folder (by default 'Applications').



## X11 Systems

**To install:**

1. Close all applications.
2. Execute the installation shell using the following command:
   ```
   sh installation-script.sh -i
   ```
3. Follow the directions that appear.

**Warning! By default, the script un-compresses data into `$DBTEMP` ; please make sure you've the correct rights in this directory.**
**If not, either unset `$DBTEMP` (/tmp will be used), or use `-w <tempdir>` option to specify the temp directory you want.**

---

# Starting and Configuring the GDC

### Topics



---

## Starting GDC

Under Windows systems, you can use the shortcut on the Start Menu.
Under X11 systems, performing envgdc shell will add the Genero Desktop Client binary directory to your path; you will be able to start with the following command : `gdc`.
Under OS X systems, the installer will create an entry in the "Application" directory.

By default, GDC will listen for Runtime System connections on port 6400. You can specify the port by starting GDC with the parameter -p.

If the port is not available, GDC will try the next port, continuing until it finds the first available one.

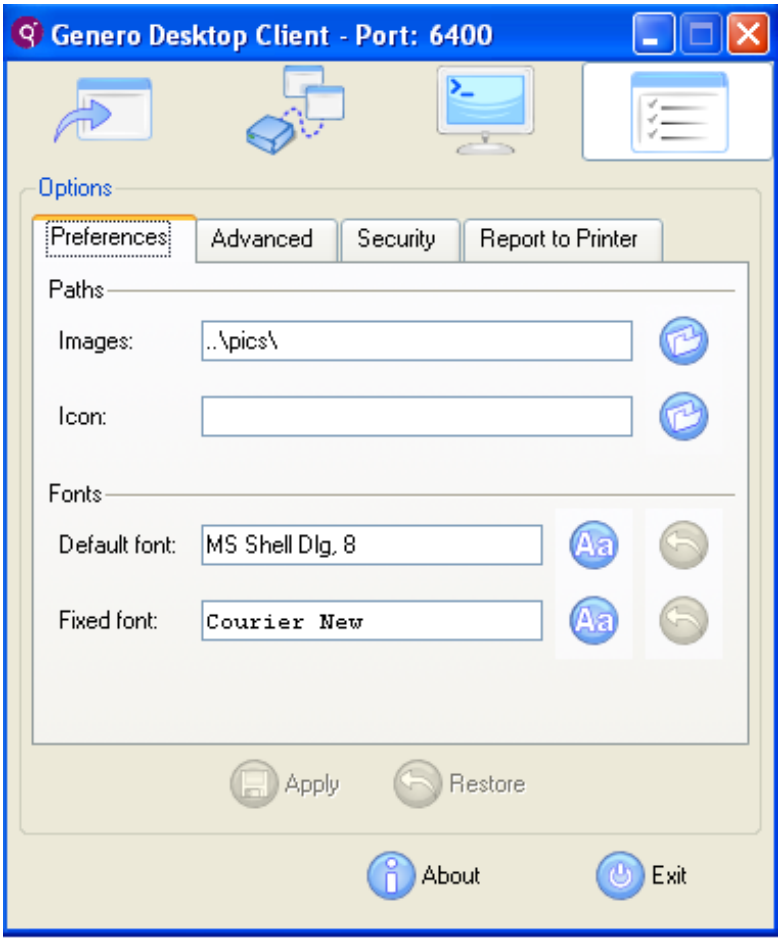
See Command Line for a list of all command line options.

---

## Configuring GDC

Click the Options icon to display the configuration options panel. The configuration options are organized across four tabs: Preferences, Advanced, Security, and Report to Printer.

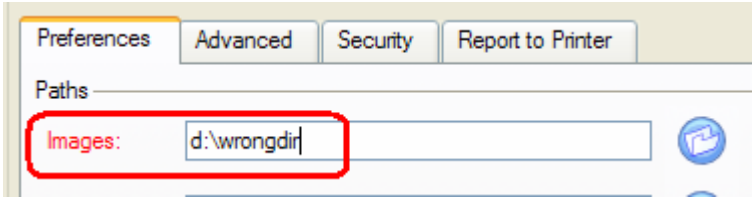
## Preferences




The following options can be configured in the *preferences* panel:

**Path to local images:** Specifies the path for gdc to search when an image is needed. GDC will first check if the name provided corresponds to an absolute file name; then it will look in the path you have specified here, then in the /pics directory. If it has still not found the image, it will draw a "..." picture.

If you enter the wrong path, the label turns red to warn you:




**Default Icon**: Specifies the default icon for GDC. This is the default icon used for the taskbar, the systray icon (under Windows systems), the shortcuts, the Terminals and applications.

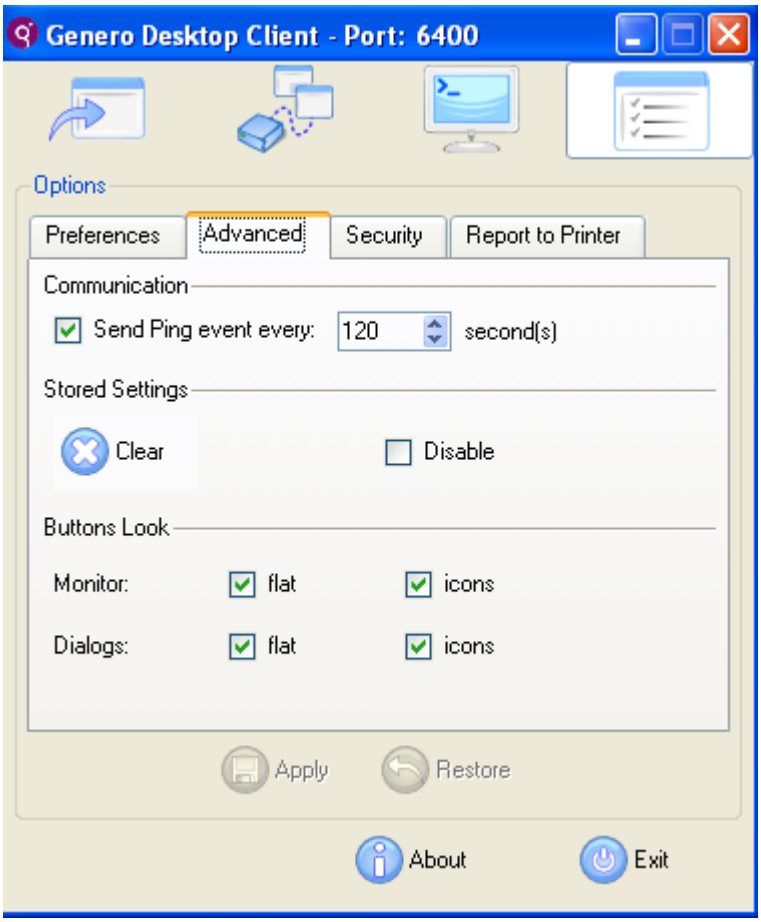
**Default Font**: Specifies the default font for GDC. This font will be used everywhere in your applications.

**Fixed Font:** Specifies the default fixed font for GDC. This font will be used when the fixed font attribute is defined.

**Warning! Changes will not be applied until the "Apply" button is clicked.**

---

## Advanced options

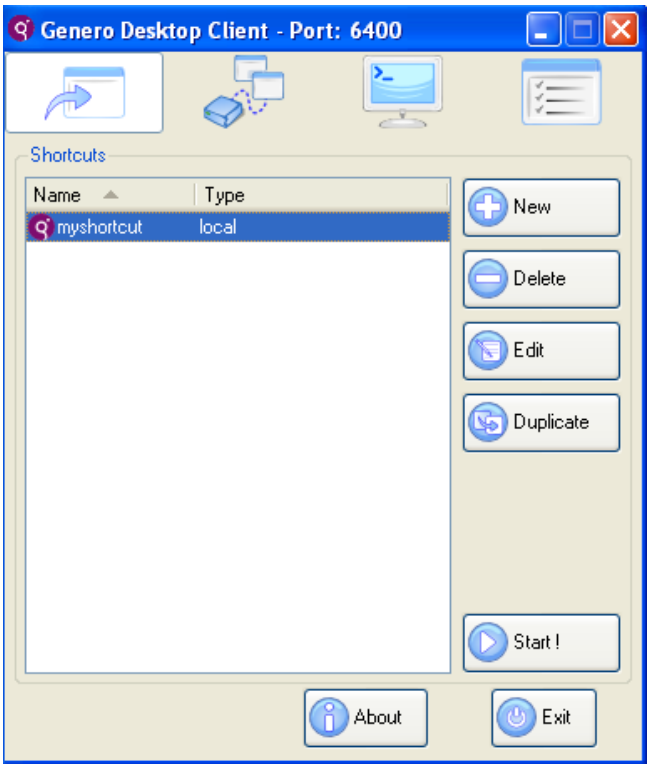


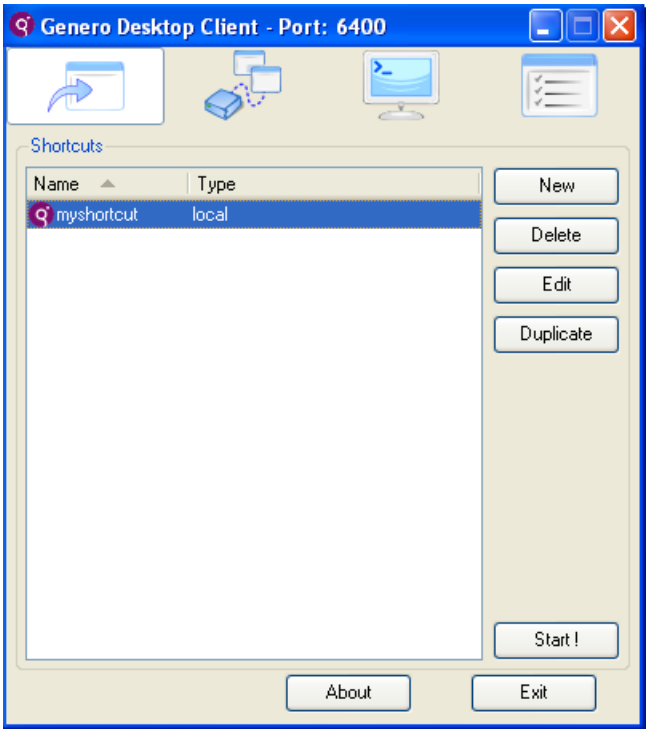

The following options can be configured in the *Advanced* panel:

**Communications (Ping Timeout:** To check whether the connection with the runtime system or the application server is still alive, GDC sends a "ping" signal over the network. The signal is sent by default every two minutes; this interval can be changed here.

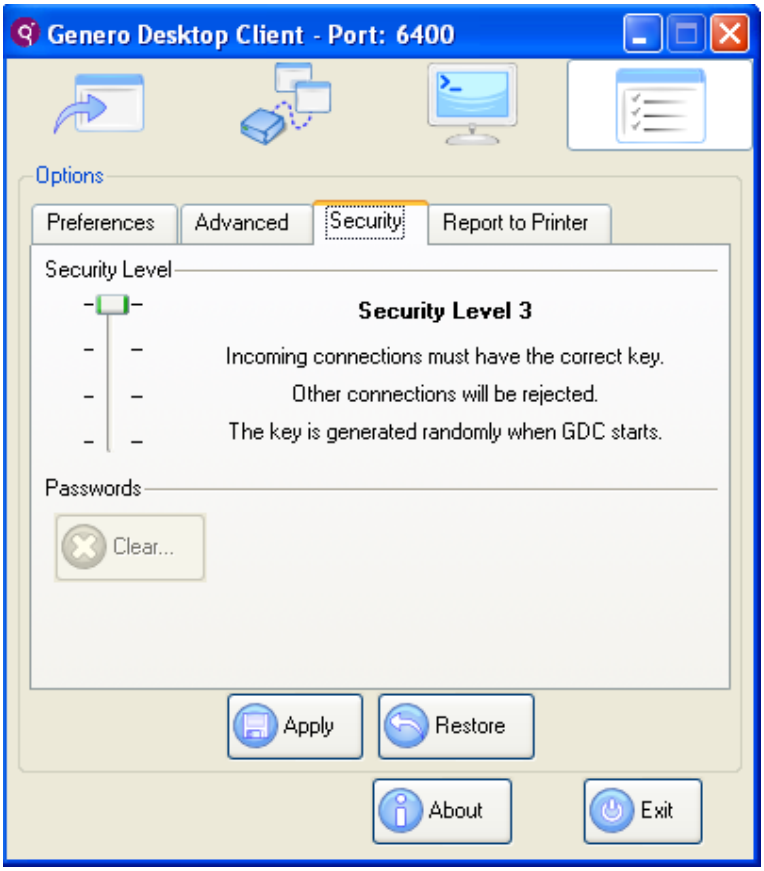
**Stored settings:** Stored Settings can be temporarily disabled by checking "Disable". To clear them, click on "Clear" button.

**Buttons look:** The look of the monitor and dialogs (shortcuts wizard, login, about box, debug console) buttons can be customized to match the look'n'feel of a regular 4gl application. The following screenshots illustrate raised buttons and raised buttons without icons:



## Security options

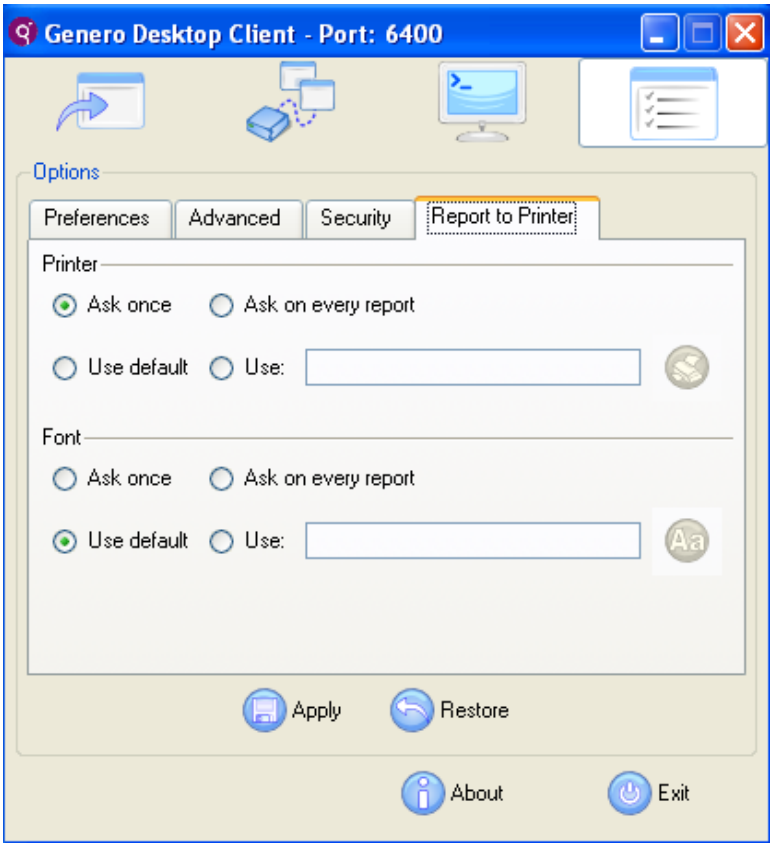



The following options can be configured in the *Security* panel:

**Security Level:** The security level can be changed here. See the "security level" section for more details.

**Clear Password:** Clears the passwords that are stored by GDC.

### Report to Printer options




This panel provides some options for the font and printer used with REPORT TO PRINTER behavior:

***ask once*:** GDC will ask for the parameter once and then keep the choice in memory until it's closed.

***ask on every report*:** GDC will ask every time a report is printed.

***Use default*:** Use the system default printer or GDC's default font.

***Use*:** Use a specified printer or font.

# Frequently Asked Questions

---

## Topics

---

## 1. General Questions

### 1.1 What does GDC mean ?

GDC stands for Genero Desktop Client.

### 1.2 What is the Debug Console ?

The Debug Console shows you the communication between the Front End and the Runtime System. See Debug Console doc. for more details. You can open this window by clicking on "Console" button on the Connections Panel.

### 1.3 I did control-right-click on a form, and a strange window appeared ! What's this ?

You have opened the 'Debug Tree'. This window shows all the Abstract User Interface Tree and the nodes attributes. This window can only appear when the GDC is started in debug mode.

If you are an end-user, you do not care about this window. If you are a programmer, please refer to Genero's documentation for more information about the AUI Tree.

### 1.4 How can I print the current window ?

GDC allows you to print the current window to any available printer.

You can refer to the Screenshots section to see how to use this feature.

### 1.5 How can I change the default icon ?

GDC allows you to change the default icon used in:

the task bar

the systray (Windows systems only)

the shortcuts system (default icon)

the terminal system (default icon, Windows systems only)

each application (default icon)

You can refer to the Configuring GDC section to see how to specify this icon.

### 1.6 What does GDC do when it has to display an image ?

GDC uses the following algorithm to search for an image:

| Step | Test done | Action | Example |
|---|---|---|---|
| 1. | filename starts with "http://" | uses the HTTP protocol to get the image from a web server | http://www.4js.com/fourjs/site/img/template/logo.jpg |
| 2.a. | filename corresponds to a file | uses the given file | "c:/mypictures/logo.jpg"<br>"../../app/pics/img2.bmp" |
| 2.b | filename + extension correspond to a file | | "c:/mypictures/logo.jpg" --> c:/mypictures/logo.jpg<br>"../../app/pics/img2" --> ../../app/pics/img2.bmp |
| 3.a. | look into <userdir> if filename exists | | userdir = "c:\mypictures"<br>"logo.jpg" -->>c:\mypictures\logo.jpg |
| 3.b | look into <userdir> if filename + extension exists | | userdir = "c:\mypictures"<br>"logo" --> c:\mypictures\logo.jpg |
| 4.a. | look into <gdcdir>/pics if filename exists | | "smiley.jpg" --> <gdcdir>/pics/smiley.jpg |
| 4.b | look into <gdcdir>/pics if filename + | | "smiley" --> <gdcdir>/pics/smiley.jpg |

| | | | |
|---|---|---|---|
| | extension exists | | |

GDC supports the following extensions : bmp, gif, png, ico, jpg.

### 1.7 What are these files in the etc directory ?

Some of the files in the *<GDCDIR>/etc/* directory are installed when installing GDC; other are created when needed.

| FileName | Description | Installed |
|---|---|---|
| `gdc_de.qm` | Contains the translation strings used by GDC when started in a **German** environment | X |
| `gdc_fr.qm` | Contains the translation strings used by GDC when started in a **French** environment | X |
| `WinCOM.cst` | Constants for WinCOM. | X |
| `.rhosts` | Contains the list of authorized hosts for the RCP server. | |
| `config.xml` | The main configuration files ; contains GDC parameters, and non "local" shortcuts. | |
| `hosts.xml` | Contains the list of authorized hosts using Security Level 1 | |
| `wincom.cst` | Windows only: list of constants for WinCOM extension (GDC 2.00.1e and later) | |

### 1.8 I plan to install Windows XP Service Pack 2. Is there any known problem with GDC ?

Changes related to Windows XP SP2 are described in the GDC and Windows XP Service Pack 2 section.

### 1.9 I plan to install Windows Vista. Is there any known problem with GDC ?

Changes related to Windows Vista are described in the GDC and Windows Vista section.

## 1.10 Since version 2.0, when I close the monitor window, GDC is still running. Is this expected ?

Yes. GDC is now running like a daemon, and the monitor window acts as a "configuration" tool. To close GDC, you'll have to click on the "Exit" button on the main page, or use the systray.

### 1.11 I changed some preferences, but I can't see any effect. Why ?

You have to click on the 'apply' button. Changes won't be applied until this button has been clicked.

## 2. Shortcut Problems

### 2.1 I can't see any shortcuts even if I create new shortcuts; is this normal ?

No, it should not happen, but it may be that the file where the shortcuts are saved is corrupted. This file can be found in *<GDCDIR>/etc/config.xml*. The best solution is to remove it and re-create your shortcuts. You can also edit it, but this operation is reserved for experienced users.

**WARNING!** This file is automatically created by GDC and should not be changed directly. Changes may introduce uncontrollable problems.

### 2.2 When starting a direct shortcut, I got "Error #1, Host not found". What does it mean ?

It means that the host you have specified when creating the shortcut with the Shortcut Wizard can't be reached from your computer. Try to ping the host using the *ping* command. If this command fails, check your network configuration or contact your system administrator. Also check the computer host in your shortcut configuration.

### 2.3 I'm under Windows, I've started a direct Shortcut, and nothing happens. What's wrong ?

First, wait a few seconds. The Naming Resolution System can take a couple of seconds and throw an error (see 2.2). If nothing happens after you've waited, go to the Terminals Panel and see if there is any terminal corresponding to your shortcut. If yes, click on the Show / Hide button to make it appear. There you will be able to see the terminal window, and then see where the problem is. Most of the time, this is due to a problem in your *command line*.

### 2.4 I'm under Windows connecting to the Runtime System using SSH. The terminal closes automatically; is this normal ?

Yes. fgltty uses the SSH protocol to send the command line. Your problem is due to the SSH protocol itself; if all the programs are closed, ssh will close the connection. That's why the terminal window is also closed. To avoid this, you can start a shell at the end of your command (add "bash;" for instance). Then, as the shell will be running, the connection will not be closed.

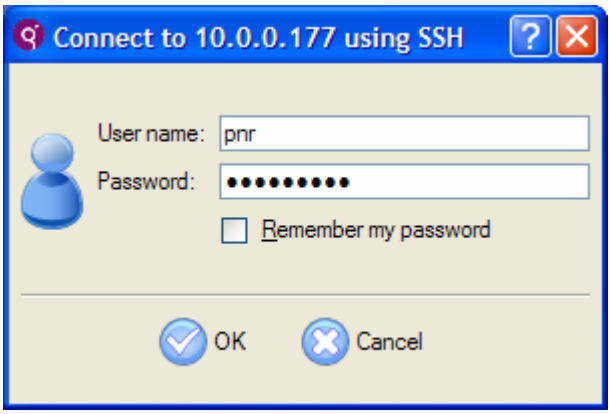
### 2.5 When starting an http shortcut, I got "Error GET : xxx" or "Error POST: xxx"; what does it mean ?

It means that there is a network problem between GDC and Genero Application Server. The message you are likely to see is "Error GET:  Connection Refused", which means that either the computer hosting the Application Server is not reachable, or that the connection was rejected. Please check your shortcut configuration and the Genero Application Server configuration.

### 2.6 I've unchecked "This connection needs a password", but GDC still asks me for a password, why ?

First of all, you have to see which program is asking you for a password.

GDC asking for a password:




fgltty asking for a password:

When you uncheck "This connection needs a password", GDC will not ask you for a password and therefore will not transmit a password to fgltty. But, if your authentication system is not correct, the host will still ask for a password. This is why fgltty asks you for the password.

To correct this, check your authentication system or contact your system administrator.

### 2.7 I have a lot of Terminals that stay in the background even though no application is running. Where is the problem ?

There is no way for GDC to know if the terminal that has been started is related to any application.

You may start shortcut A, then B, and application B starts before application A.

You may start a shortcut called A that starts an application called B.

You may start a shortcut A that starts applications A, B.

So, GDC can't be sure that, if an application is closed, it should close the terminal.

To close the terminal, check the Terminals Panel: there are all your login sessions that can be Terminated by clicking on the corresponding button.

You should either:

1. Add "; exit" to the end of your command string (eg.: @FGL ; cd $FGLDIR/demo ; fglrun ia.42r ; exit) *or*
2. Use exec to replace process (eg. @FGL ; cd $FGLDIR/demo ; exec fglrun ia.42r)

### 2.8 There is no button in the shortcut panel, what should I do to modify my shortcuts ?

You are running GDC in "user" mode. To change your configuration options and your shortcuts, you have to start GDC in admin mode, using the correct command line option.

### 2.9 What does the "Unexpected protocol version sent by the runtime system" message mean ?

The protocol used by the Runtime System and the GDC to communicate may change depending on new features added to Genero. The message "Unexpected protocol version sent by the runtime system" is displayed when the Runtime System and the Front End use protocols that are too different to work perfectly. You can press "Continue", but this is likely to cause problems that may lead to GDC's crash.

We recommend that you always use GDC and a Runtime System version that are designed to work together.

### 2.10 RLogin is not working, any ideas ?

Under Linux / Mac OS systems, fgltty must have root privileges to be able to start rlogin connections. Please refer to the Rlogin Problem section for more information.

---

## 3. Network Problems

### 3.1 Can I change the listening port ?

Yes. -p (or --port) parameters allows you to change the port: `GDC -p 3200` will make GDC listen on port 3200. In every case, if the specified port is not available, GDC will try the next port and listen on the first free one.

### 3.2 I've started GDC on port 6400 and it seems it is listening on port 6401... Why ?

If the port you specify is not chosen by GDC, it means that it was not free. This may be due to:

- another instance of GDC that already listens on this port
- an instance of another Four J's Front End that also listens on this port.

You can use -p command line option to force GDC to stop if the port is unreachable.

### 3.3 As I use SSH, my connection is completely secured, right ?

It depends on the way you've created your shortcut. If you've entered a "forwarded port", SSH tunneling will be set and then your connection is **secured**. If this value is not entered, SSH is only used to secure the connection when GDC needs to start an application on a distant host, and not the complete communication.

## 4. Logging System

### 4.1 Why a logging system ?

The logging system has two main utilities, debugging and demo making.

Debugging: if you notice any problem in the GDC (especially an unexpected crash), you can record the scenario that caused the problem and send it back to us. This will allow us to solve the problem.

Demo Making: you may want to make a demonstration for your application, but you don't want to export the installation of the Database and the Runtime System. Recording

a demo will allow you to create a scenario of your application and move it anywhere. The only requirement is GDC.

**Warning!** Not everything is logged by GDC. See (4.2) for more details.

### 4.2 I have moved some columns on a table during a demo, but the replay does not change the order; is this normal ?

Unfortunately, yes. The logging system only logs the communication between the GDC and the Runtime System. Everything that is completely handled locally (like moving columns) will not be recorded.

---

## 5. Concerning ActiveX

### 5.1 I've un-installed the Active X, but there are some files left on my computer. What's wrong ?

Please be sure you have closed all of your browser windows before un-installing GDC's Active X. If not, an instance of the GDC may still be running, and you will not be able to un-install the application successfully. Therefore, some files may be left on your computer.

### 5.2 What is the difference between ja/r and wa/r for HTTP connections ?

GDC can be used with the Genero Application Server in two modes:

- as a module integrated into the Genero Application Server (as an ActiveX)
- totally independently

**When you run as a module, it works as follows:**

In your Internet Explorer browser, you enter an URL like `http://server:port/path/wa/r/applicationName` where:

- *server* is the server name
- *port* is the port of the server
- *path* is a specific path, depending on your configuration (could by empty, could be "cgi-bin/fglccgi")
- *applicationName* is the name of the application you want to start

Then the Genero Application Server will send back an HTML page embedding GDC as an ActiveX, and it will start the application.

**When you run independently, it works as follows:**

In the shortcut system, you can specify either *server*, *port* and *applicationName*, or a complete URL. When you specify each piece of information, it is equivalent to the following URL: `http://server:port/ja/r/applicationName` ; then an internal protocol is send to the front end.

| | | |
|---|---|---|
| **wa/r** | What happens: | The Application Server will send an HTML page with GDC Ax embedded and start an application. |
| | When to use it: | This should be used in your **web browser** to start the GDC Ax and an application from the Application Server. |
| **ja/r** | What happens: | The Application Server will send an internal protocol understood by the GDC. |
| | When to use it: | This should be used in the **shortcut system** to specify a shortcut using the HTTP connection. |

---

## 6. GDC's behavior

### 6.1 GDC is blocked when a system dialog is open: is this normal ?

System dialogs depend on the Operating System. When the dialog is open, GDC waits for an answer from the Operating System. Everything that is sent by the Runtime System is stored and processed once the dialog is closed by the user. So, this is normal if, for instance, you have the "printer" dialog open (after a `START REPORT` with `EXPORT DBPRINT=FLGSERVER`); GDC seems to be blocked and does not start or carry on any other application. Close the dialog and it will carry on working as expected.

### 6.2 When I start lots of applications, GDC may be unstable and displays really weird things. What is happening ?

This happens only under Windows systems. Windows applications are limited in the number of USER and GDI objects. When this number reaches 10,000 for the whole system, it is no longer possible to create widgets or any system objects (font...). This results in a corrupted screen and unstable system.

You can monitor GDI and USER objects in the Windows Task Manager, "Processes" page (you may need to add GDI objects and USER objects columns using the View / Select columns menu).

This limit can be modified for your system by changing the registry. See http://msdn2.microsoft.com/en-us/library/ms725486(VS.85).aspx and http://msdn2.microsoft.com/en-us/library/ms724291(VS.85).aspx for more details.

Please keep in mind that this limit has been set by Microsoft and modifying the registry is a risky operation.

A better approach would be to reorganize your applications to have less windows open or to use smaller windows; even hidden and non visible items (like in a folder) consume user / gdi objects.

# Shortcut System

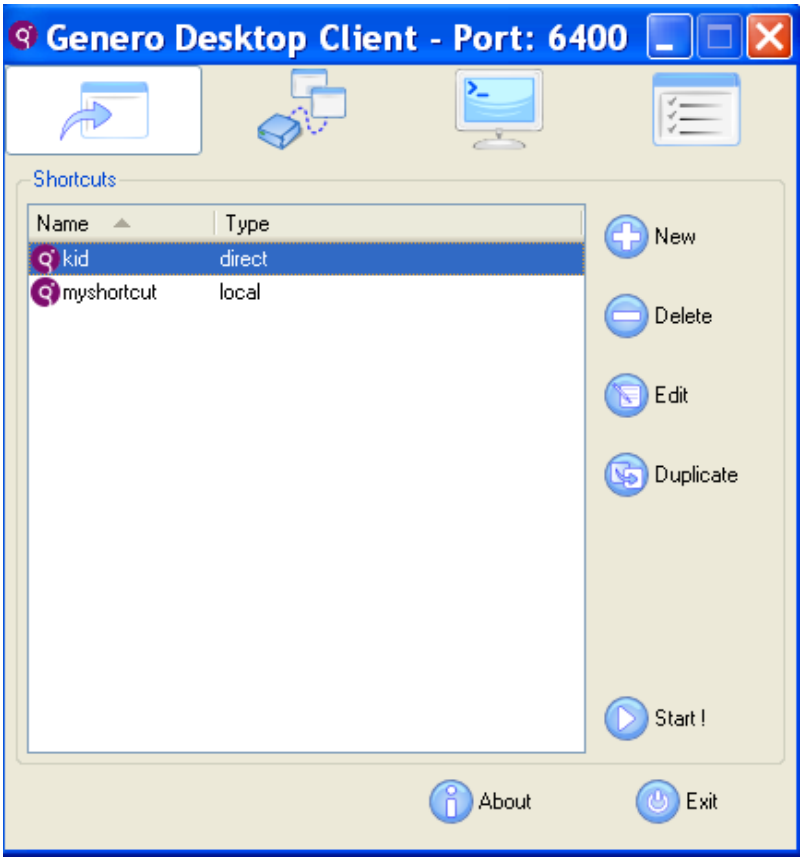
### Topics

- Shortcut System Overview
- Administration mode
- Shortcut Management
- Starting Shortcut
- Local Shortcuts

---

## Shortcut System Overview

Genero Desktop Client (GDC) is able to store the information needed to start an application. The information is stored as a **shortcut**. You add a shortcut for each application. Shortcuts are stored the same way internally on each platform.




---

## Administration Mode

By default, the Genero Desktop Client starts in *user mode*, where shortcuts and options cannot be modified. To create shortcuts or modify options, the Genero Desktop Client must be started in *admin mode* by using the "`--admin`" or "`-a`" command line option.

---

## Shortcut Management

Genero Desktop Client can interact with the Runtime System in different ways:

- The Runtime System is on a distant host and GDC will start it via telnet, rlogin or SSH (Direct Connection),
- The Runtime System is on the same host and GDC will start it as a local application (Local Connection),
- The Runtime System is on a distant host, and GDC will be connect to it via Genero Application Server (Connection via AS).

The Shortcut Management System allows you to add a new shortcut, to edit, duplicate or remove an existing one.

Editing or adding a shortcut displays a Wizard to help you to create (or edit) your shortcut.

---

## Starting a Shortcut (Application)

You can start a shortcut (start an application) by double-clicking the shortcut icon or by selecting the shortcut and pressing the "Start it" button.

---

## Local Shortcuts

By default, your shortcuts are saved into the <*GDCDIR*>/etc/config.xml file. You can also create **local** shortcuts that will be stored locally for each user. This feature can be useful if you share GDC on a network drive and don't want the user to modify the common shortcuts.

When config.xml file is read-only, any modification to a non-local shortcut will display a warning and create a local copy of the shortcut.

---

# Shortcut Wizard

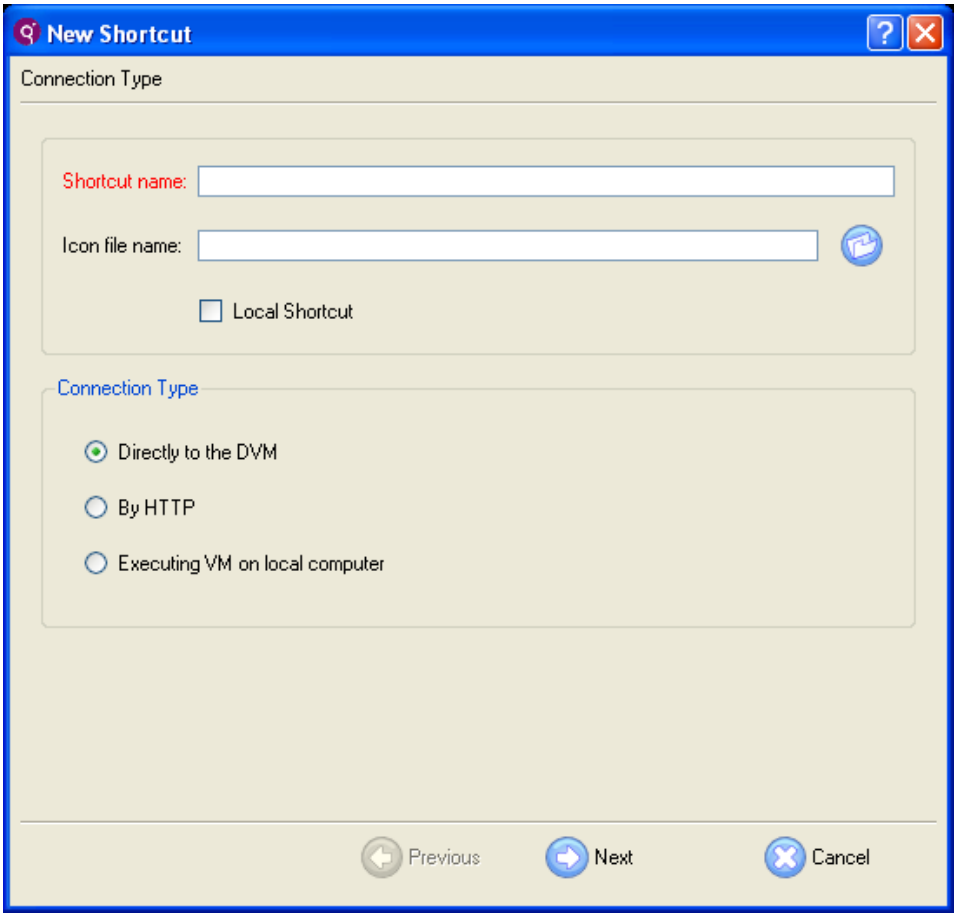
### Topics

- Wizard Start
- Direct Connection Shortcuts
- SSH Tunneling
- Local Connection Shortcuts
- Connections via Application Server
- Shortcuts and Environment Variables

---

## Wizard Start

When you start the Wizard, you will be asked for the name of the shortcut, and for an optional file name that will be used to display an icon associated with this shortcut.




You then must choose the Connection Type for your shortcut:

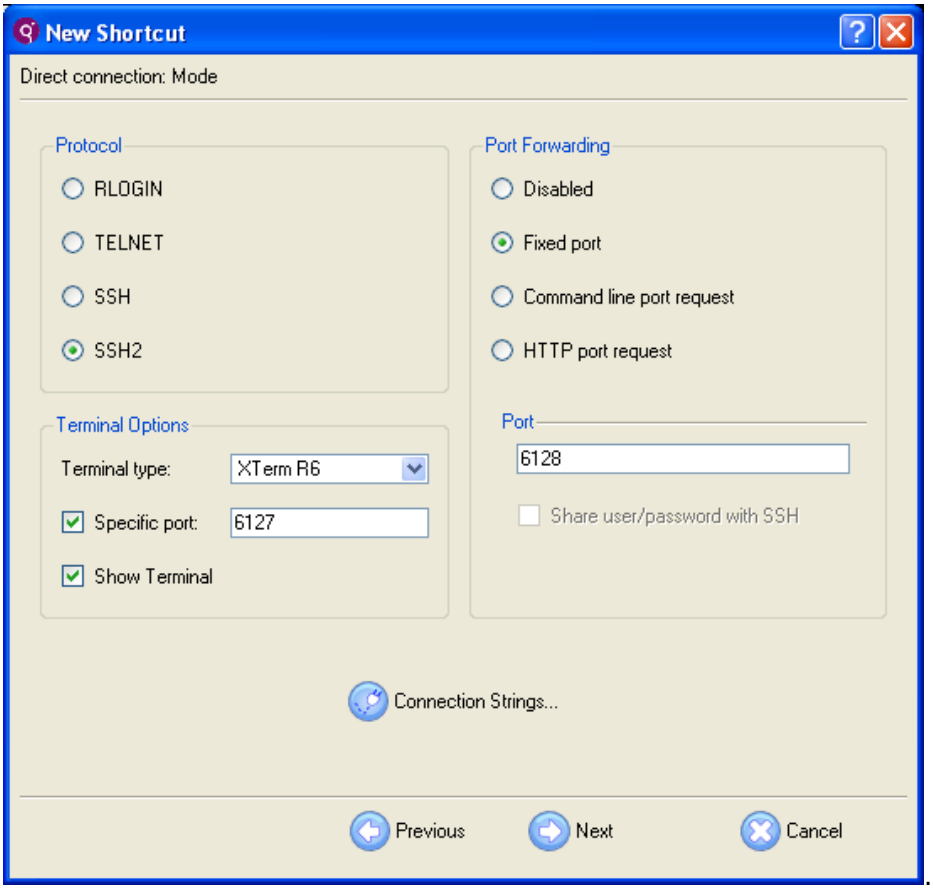
- **Directly to the DVM** - The Runtime System is on a different host, and GDC will start it via telnet, rlogin, or SSH (Direct Connection)
- **By HTTP** - The Runtime System is on a different host, and GDC will connect to it via the Genero Application Server (Connection via AS)
- **Executing VM on local computer** - The Runtime System is on the same host, and GDC will start it as a local application (Local Connection).

---

## Direct Connection Shortcuts

In this mode, the Runtime System is directly connected to the GDC using TCP/IP network. To start your program on the Runtime System host, GDC will connect to the host using either rlogin, telnet, SSH or SSH2. You can specify an alternative port, if your configuration needs it. Using SSH, tunneling can be established to secure your connection.

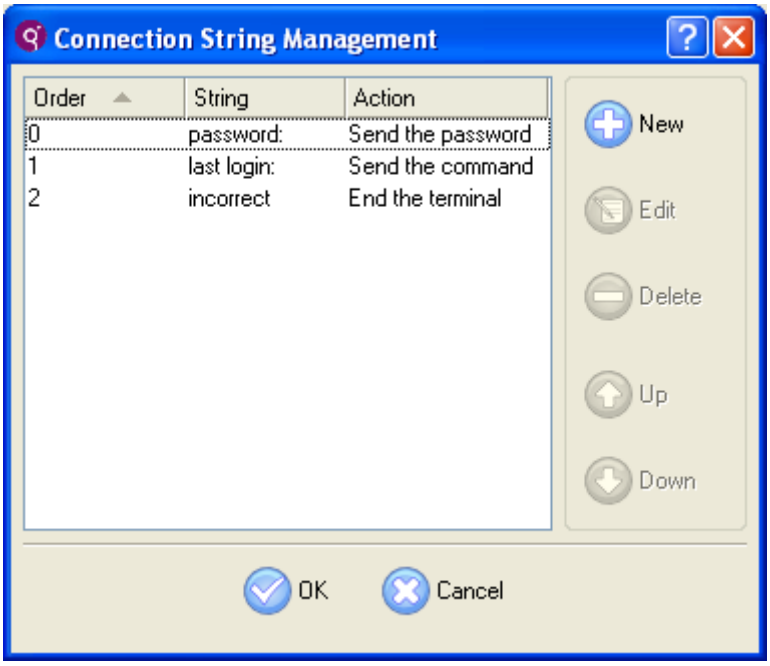
**Note:** This connection mode is only used when GDC connects to the host to start the application. Currently, the communication between the Runtime System and GDC does not use rlogin, telnet, SSH, or SSH2.



If ***Show Terminal Utility*** is checked, the window of FGLTTY, our Emulation Terminal Utility, will be visible. (Please refer to the Terminals Section). This could help you check whether your command line is valid..

**Warning!** fgltty must have a root sticky bit on Linux / Mac Os systems to be able to start rlogin. Please refer to the rlogin problem section.

Click the **Connections Strings** button to display a window where you can specify connection strings:




This table is used to tell GDC what to do when the Runtime System host displays a given string on the terminal. GDC can perform the following actions:

- Ask the user for a value, and send it back
- Display a message to the user
- Ask for a password
- Send the shortcut password
- Send the shortcut command
- Execute a local command and send the result
- Return a defined string
- Ignore the Runtime System string
- Send the login
- Get a free port number for Port Forwarding
- Show or hide the terminal
- Close the terminal

"Order" specifies in which order GDC tries to recognize the different strings. You can specify whether each string should be recognized only once or every time.

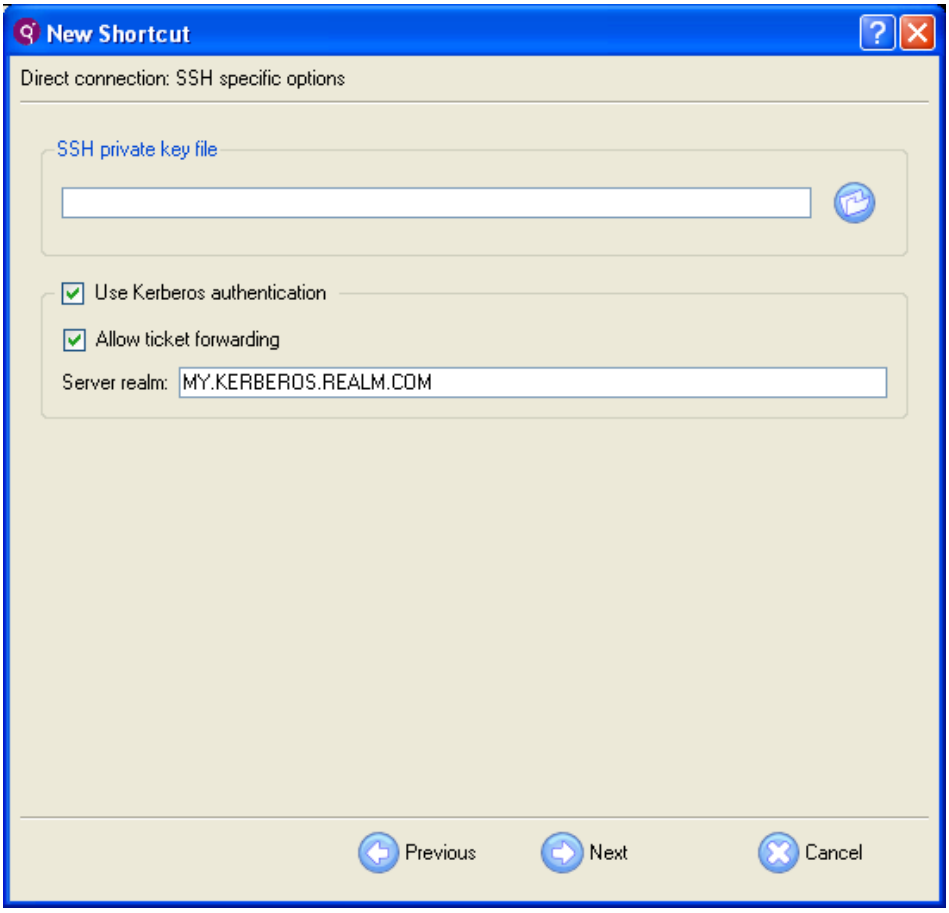
RLOGIN or TELNET systems have by default two connection strings: `Password query string` and `User logged in` .

Examples:

| Recognized string: | Description: | Action performed by GDC |
|---|---|---|
| `password:` | This is the string used by the telnet daemon to ask for the password. | `Sends the password` |
| `last login:` | This is the string used by the telnet daemon to tell the user he has logged in successfully. | `Sends the command` |
| `login:` | This is the string displayed by the telnet daemon when the login has failed | `Displays a message "Authentication has failed"` |

Please contact your System administrator if the default values are not appropriate.

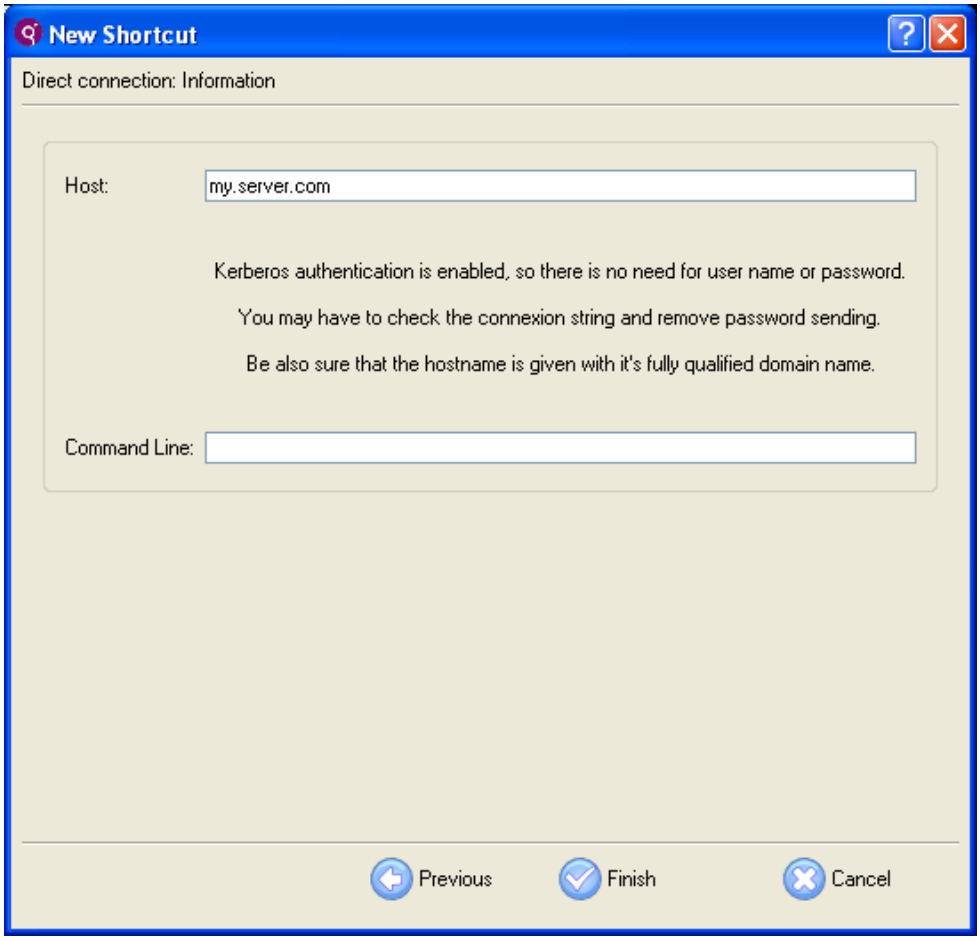
The next screen allows you to enter options for SSH:



- **SSH private key file:** If you use an SSH connection, you can also specify an ssh key file that contains the login information. The file format must use the PuTTY format and can be generated using PuTTY tools.
- **Use Kerberos authentication**: On Windows platforms ( all versions after Windows 2000 ) you can also use Kerberos authentication if your user and computer are registered on an ActiveDirectory that provides a Kerberos interface.
- **Allow Ticket Forwarding**: Ticket forwarding allows the SSH server to forward the Kerberos ticket that identifies the user to other processes.
- **Server realm**: The server realm identifies the Kerberos "domain". This field can be mandatory, depending on the ActiveDirectory / Kerberos server configuration. Ask your System administrator for further details.

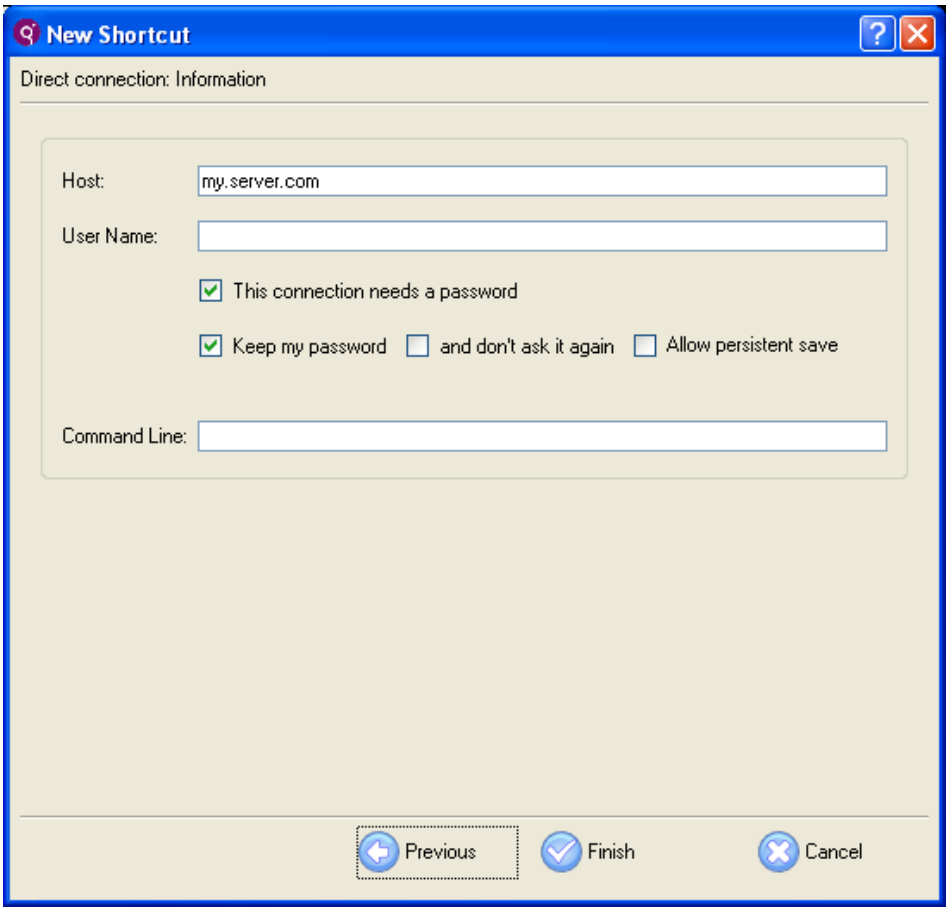
Next, you should enter login information.

If you use Kerberos authentication, you will see the following screen:




Since Kerberos uses the login information of the current user, on this screen you can only enter the hostname and the command line to be executed.

If you don't use Kerberos, you will see a different screen:




In this case, you have to provide the following information:

- the hostname where the Runtime System is hosted. This can be omitted if you use the -host command line.
- the username you are using to connect to the host. This can be omitted if you use the -user command line.
- the command line that will be executed to start the application on the Runtime System side.

If ***This connection needs a password*** is checked, GDC will ask you for a password. If your configuration allows you to connect without a password (for instance using rlogin combined with a .rhost file), uncheck this option. If a password is still requested, review your configuration.
**WARNING!** GDC will not modify your configuration to allow you to connect without a password. It is up to you or your administrator to manage this.

If ***Keep my password*** is checked, GDC keeps in memory the password you enter the first time you start a shortcut, and reuses it when you restart. The

password is kept in memory, and is lost if you stop GDC.
**WARNING!** GDC never stores your password in a file or elsewhere unless ***Allow persistent save*** is checked. The password is kept in memory while GDC is launched, and is forgotten once GDC is stopped.

If ***and don't ask it again*** is checked, GDC will only ask for the password the first time a shortcut is launched. After that, the password will be silently sent, without bothering the user.

If ***Allow persistent save*** is checked, GDC will store the password in settings once it is accepted. The password is also kept in memory, but it will not be lost if you stop GDC.
**WARNING!** GDC stores your password on disk in an encrypted form which is very difficult to read but not impossible. Someone with strong knowledge in cryptology can eventually break the password protection.

Within the command line, you can use the following tags:

| Tag | Replaced by |
|---|---|
| @FGL | FGLSERVER=<*IP Address*>:<*serv num*> |
| You can use one of the @FGL variants depending on your system: | |
| @FGLNT | set FGLSERVER=<*IP Address*>:<*serv num*>&&set FGLGUI=1 |
| @FGLCSH | setenv FGLSERVER "<*IP Address*>:<*serv num*>";setenv FGLGUI 1 |
| @FGLKSH | FGLSERVER="<*IP Address*>:<*serv num*>";export FGLSERVER;FGLGUI=1;export FGLGUI |
| @SRVNUM | <*GDC listening port - 6400 (The second part of FGLSERVER)*> |
| @PORT | <*GDC listening port*> |
| @USR | <*Client current user name*> |
| @USER | <*User name on the remote system*> |
| @IP | <*IP address of the client computer*> |
| @COMPUTER | <*Machine host name*> |
| @E_SRV | export FGLSERVER |
| @4GLSRVVER | <*GDC version*> |

These tags will automatically be replaced when the command is sent to the Runtime System host.

## SSH Tunneling

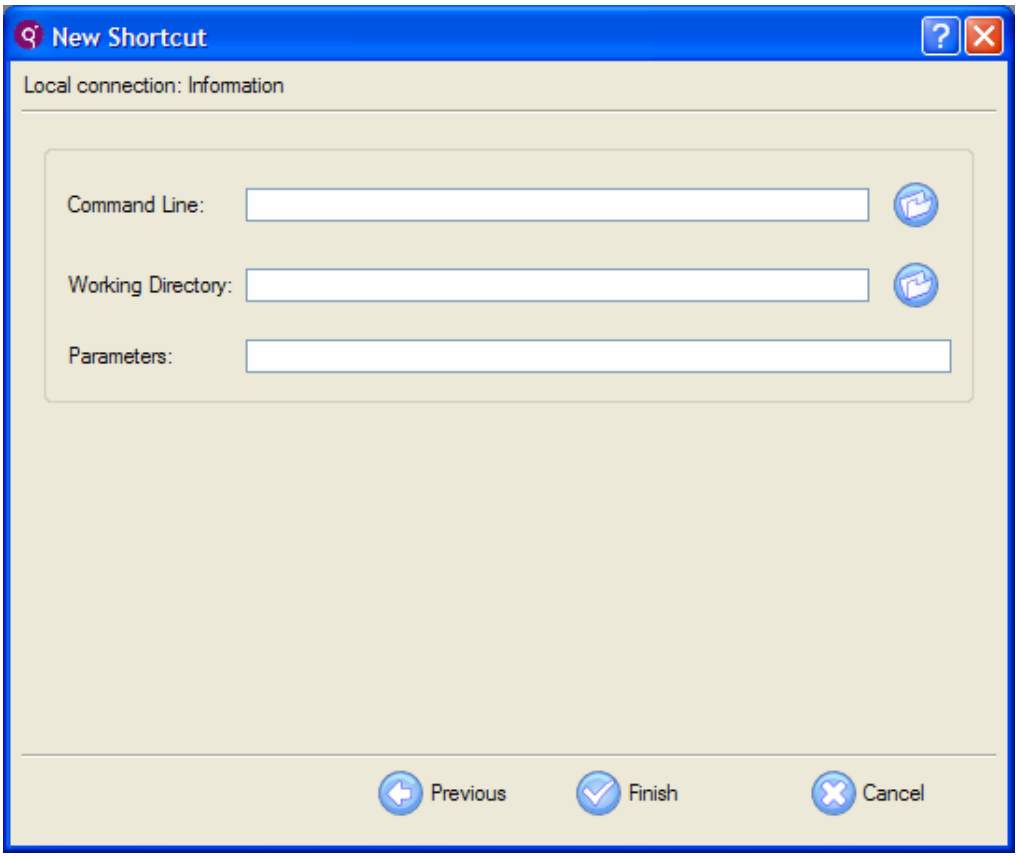
Please refer to the GDC and SSH section for more information about SSH tunneling.

---

## Local Connection Shortcuts

In this mode, the Runtime System is on the same computer as GDC. To start your program on the Runtime System, GDC will simply start an executable (giving it some parameters). This executable will typically be:

- fglrun started in the application directory, or
- a batch file that will perform all applications




You will have to provide the following information:

- the command line to select the executable
- the working directory
- the parameters needed by the executable

You can have, for example:

| Command Line | Working Directory | Parameters | Remarks |
|---|---|---|---|
| `fglrun` | `/home/fgl/demo/` | `stores.42m` | fglrun should be in the PATH |
| `c:\fourjs\fgl\bin\fglrun.exe` | `c:\genero\demo` | `stores.42m` | |
| `c:\demos\stores.bat` | | | stores.bat is a batch file that sets the environment and starts the program. |
| `C:\WINNT\system32\CMD.EXE /C "d:\fjs\fgl\envcomp.bat && fglrun D:\app\gift.42r"` | `D:\app` | | This will start envcomp.bat in FGLDIR to set the environment, then start the gift application. The Working directory is the directory where fglrun can find the required files. |

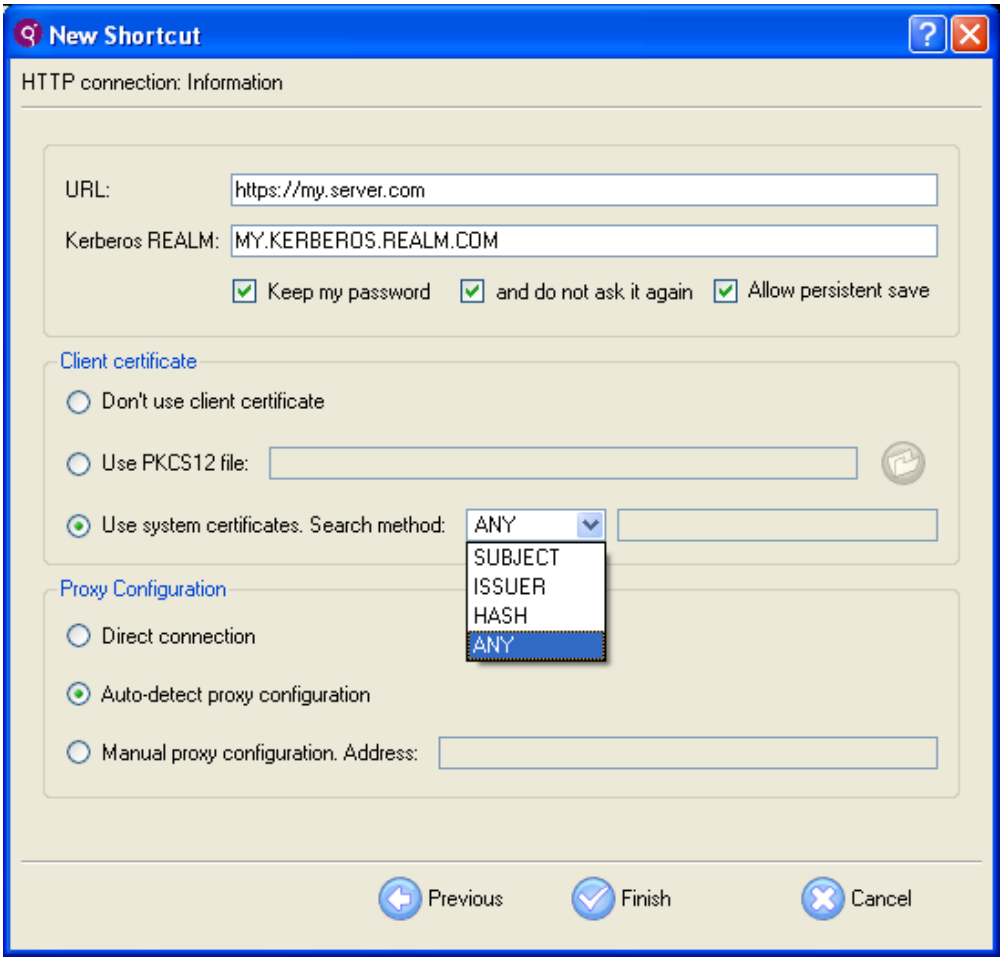
**WARNING !:** If you have the configuration shown below, you must be sure that all the environment variables are set **before** starting the application. The variables can be set in the "Environment Variables" system dialog.

| Command Line | Working Directory | Parameters | Remarks |
|---|---|---|---|
| `c:\fourjs\fgl\bin\fglrun.exe` | `c:\genero\demo` | `stores.42m` | |

---

## Connections via Genero Application Server

In this mode, the GDC will be connected to the Runtime System via the Genero Application Server, using the HTTP protocol.




The URL looks like: `http://myserver:6393/cgi-bin/fglccgi/ja/r/uidemo`

You can also specify the Kerberos Realm on Microsoft Windows platforms if the web server requires a Kerberos authentication. This field is not mandatory for a Kerberos authentication, but it can be useful when many Kerberos servers are on the same network. Keep in mind that Kerberos authentication is designed to work in a local area network, and the results are unpredictable across a wide internet connection.

If the URL begins with https (secure), you can specify a client certificate to authenticate the client to the HTTPs server. For the moment, except for Microsoft Windows systems where you can use a system certificate, only PKCS12 certificate are supported.. Since PKCS12 certificate are password protected, you will be prompted for a password when the certificate is installed. Please note that the password may be requested again, depending of the state of the three checkboxes "Keep my password", "and do not ask again", and "Allow persistent save".

On Microsoft Windows, there are four methods of selecting a system certificate:

- SUBJECT: use the first certificate in which the subject field contains the given string.
- ISSUER: use the first certificate in which the issuer field contains the given string.

- HASH: use a hexadecimal hash that identifies a certificate. (eg: A5 C8 3F 34 21 C5 FF 8B 0A 0B 24 57 DD B2 C8 9F 1C 7A 45 76)
- ANY: select the first one

If your connection uses a proxy, you can configure it also. See the FAQ about URLs and the Genero Application Server documentation for more information on configuring applications.

---

## Shortcuts and environment variables

In some fields, GDC will replace any `$xxx` (X11 / Mac OsX) or `%xxx%` (Windows) by the corresponding environment variables. The fields concerned are:

| Connection Type | Fields |
|---|---|
| direct | `host`, `username`, `commandline` |
| http | `url` |
| local | `command line`, `working directory`, `parameters` |

If you want GDC to simply send the text instead of replacing the environment variable, use the "\" character to escape the variable (e.g. \$HOSTNAME or \%HOSTNAME\%).

---

# Connections Panel

Summary:

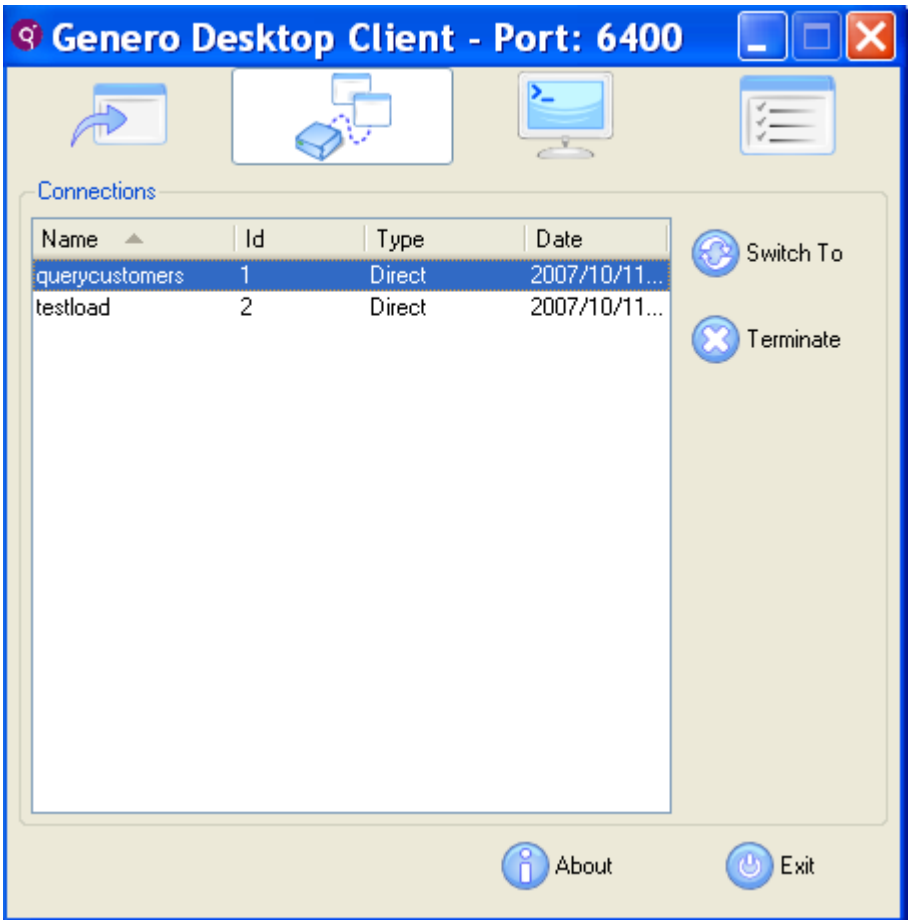
- Overview
- Switch to / Terminate Buttons

---

## Overview

The "Connections" Panel lists applications that can be handled by GDC. For each application, it displays the following information:

- **Name** - The name of the application. This refers to the text attribute of the `UserInterface` Node.
- **Id** - An internal identifier.
- **Type** - This refers to the way the application is connected: directly connected to the Runtime System (direct) or using HTTP protocol via Genero Application Server (http).
- **Date** - An indication when the application was started.



## Switch to / Terminate Buttons

If you click on the "Switch to" button, the selected application will be raised to the top, and the focus will be set on the current window. This will allow you to find your application easily if a lot of programs are launched.

If you want to stop any application, you can select it and click on the "Terminate" button. This will send the information to the Runtime System and the application will be stopped by GDC.

---

# Terminals Panel

### Topics

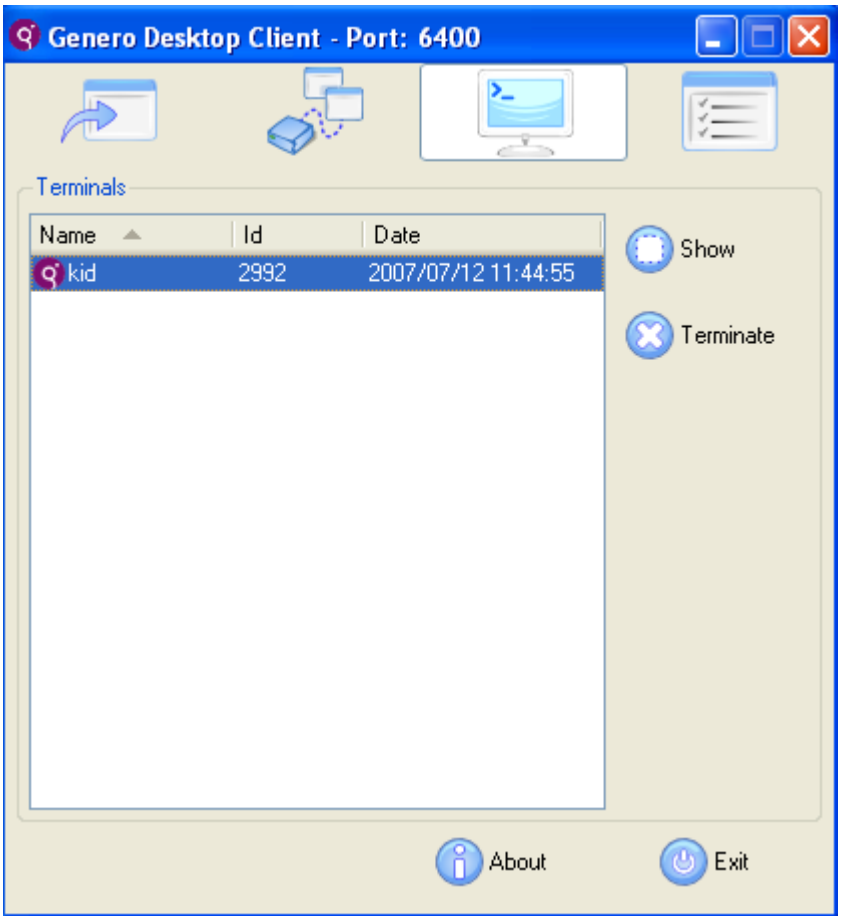


---

## Overview

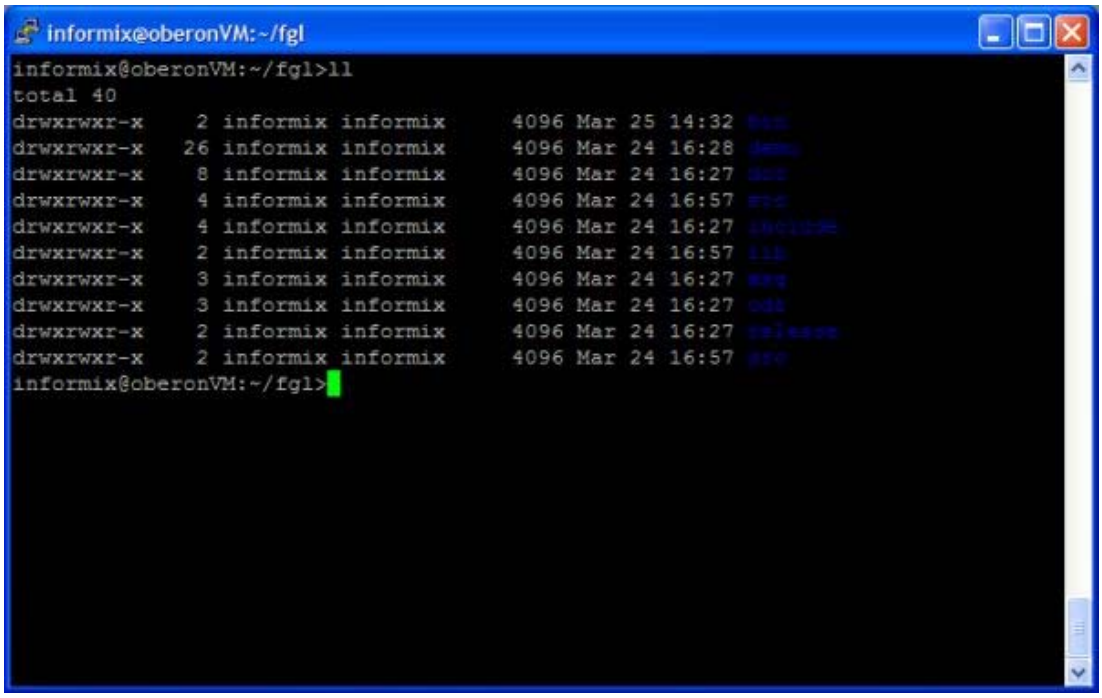
Shortcuts use a terminal emulation utility (called fgltty) to connect to the system hosting the Runtime System. Each line of the list in the Terminals panel refers to an active instance of the utility.

Terminals are automatically started by the Shortcut System.

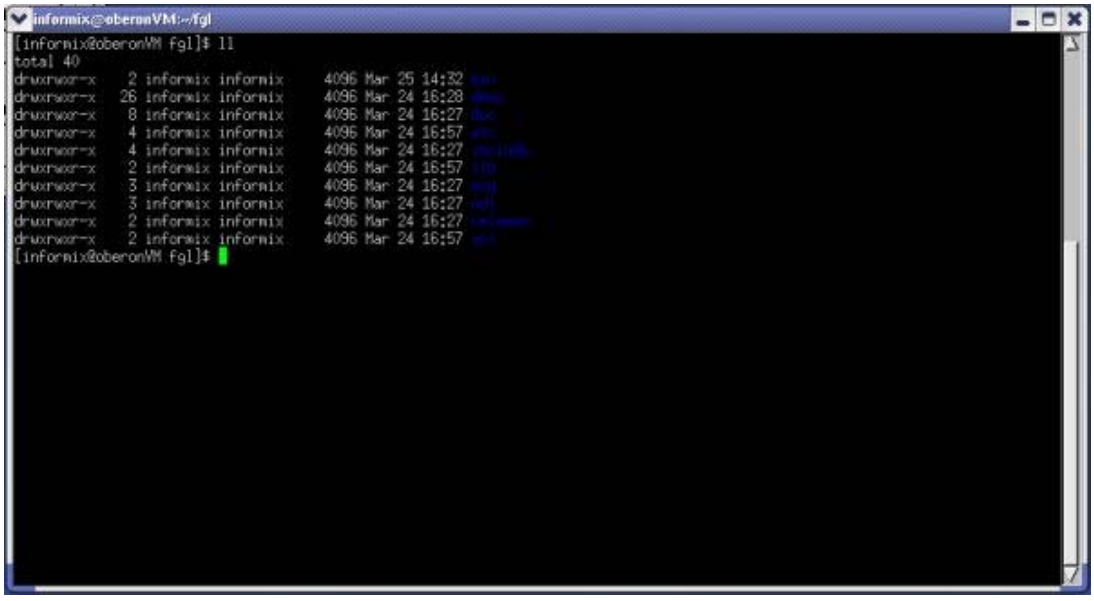



The terminal utility provided is called fgltty.

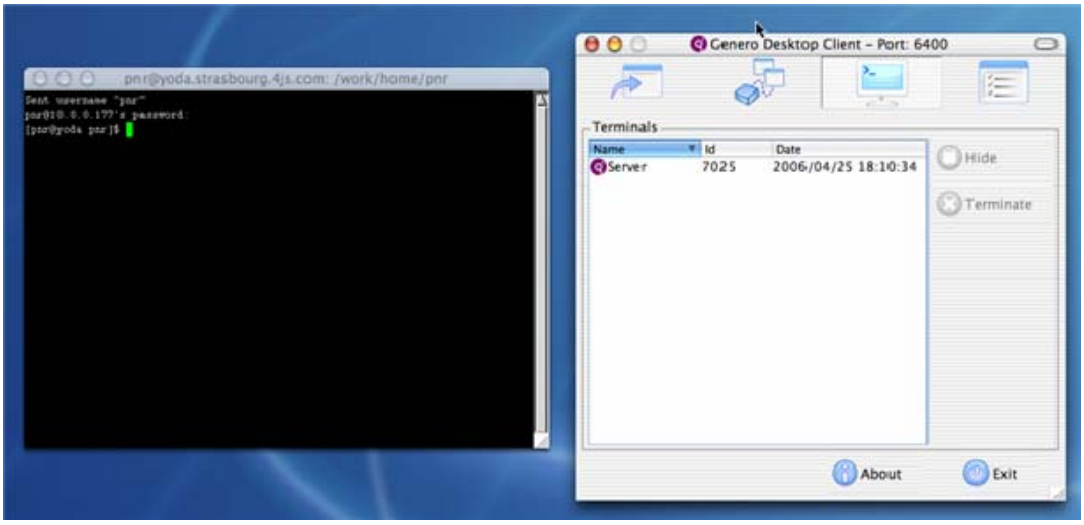
**Windows version:**




**X11 version:**



**OS X version:**




---

## RLogin Problem

The rlogin protocol specifies that each connection from an rlogin client to an rlogin server must originate on a privileged port, that is, a port in the range of 512 to 1023 inclusive. On a UNIX system, the client **must have root privileges** to gain access to these ports.

The system rlogin command is owned by root, and has a sticky bit that allows normal users to start rlogin as root:

```
>ll /usr/bin/rlogin
-rwsr-xr-x 1 root root 15376 Jun 24 2002 /usr/bin/rlogin
```

If you need to use rlogin with GDC Linux or Mac Os systems, you must enter the Administrator password during installation.

---

## Show / Hide

This button allows you to show or hide the selected Terminal. When you create a shortcut using the Shortcut Wizard, you can specify whether the Terminal Utility is shown. With this button you can show a hidden terminal, or hide a visible one.

Typically, this is used to check why your application has not started. Showing the Terminal Utility will display what has happened.

## Terminate

This button allows you to terminate the selected Terminal Utility.

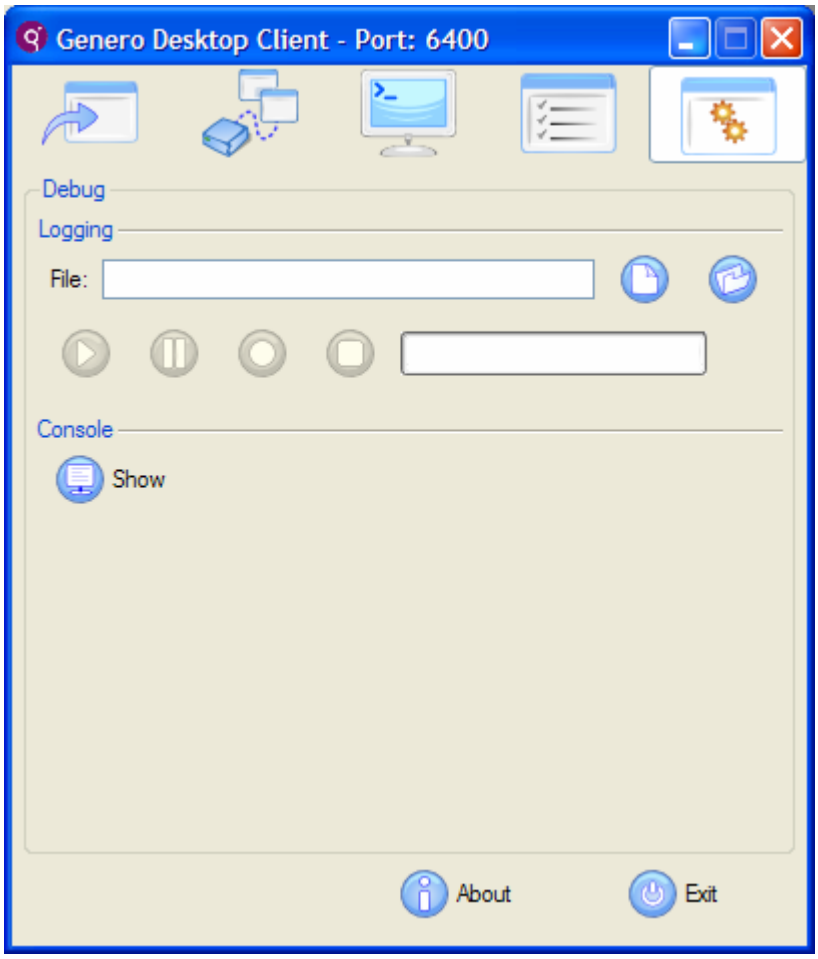
# Debug Panel and Logging System

### Topics



---

## Overview




The "Debug" Panel shows the GDC debug facilities: the logging system and the debug console.

The Debug Panel is only available when GDS is started in debug mode.

## Logging

When GDC is started in debug mode, the logging system is available. This system will help you to:

- debug your applications
- create a demo

What is logged? The complete communication between the front end and Runtime System, so the Runtime System is not needed when you replay your demo.

**Warning!** As only the communication is recorded, the "local-only" actions such as moving columns are not saved and replayed. Only the sent value of a field is saved; all user interactions (copy / paste, cursor...) are not saved.

### Recording Demo

To record a demo, first select a log file to store the scenario. Then, click on the record button to start recording.

**Warning!** Only applications you launched AFTER starting recording are stored.

### Replay Demo

To replay a demo, first select the log file where the scenario is stored. Then, click on the play button to start playing the demo. You can pause the replay by clicking on the pause button. The progress bar will indicate the progress of the demo.

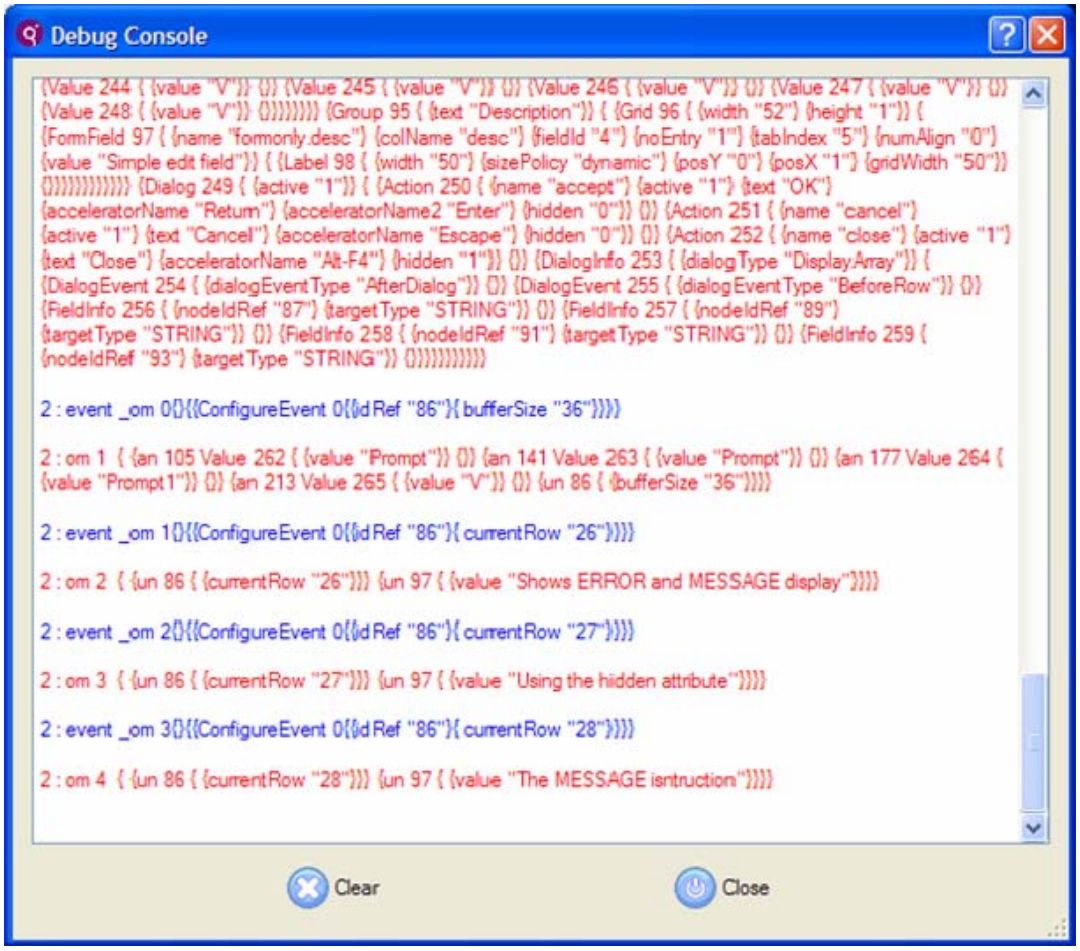
**Warning!** No user interaction is possible when replaying a demo, so you may have to stop recording the demo before the end of the application. Then, if you want to kill this application, you must use the Connections panel.

---

## Debug Console

If you click on the "Console" button, a Window called Debug Console will appear.

**Warning!** The debug console is only available in debug mode.




In this window some debug information will be displayed:

- in blue - what is sent by GDC to the Runtime System
- in red - what is received by GDC from the Runtime System
- in green - some comments or other information

This debug console could help you to see the communication between the GDC and the Runtime System.

# Stored Settings

**Topics**

- Overview
- List:
  - Window Settings
  - Toolbar Settings
  - Table Settings
  - TextEdit Settings

---

## Overview

If you change some graphical aspects of your application, GDC will store them and reload them when the same application is stored again.

Under Windows Systems : Settings are stored under \HKEY_CURRENT_USER\Software\Four J's Development Tools\GDC. key in the registry.
Under Linux and MacOS an internal system is used to store data.

This page lists all the parameters that are stored.

Stored settings can be disabled or cleared in the Options panel.

---

## List

| **Window Settings** | |
|---|---|
| **Global Settings** | These settings are stored under .**\Application\<*applicationName*>\Windows\<*windowName*>** <*applicationName*> is taken from the "text"-Attribute of the `UserInterface` Node |
| | The initial position of the Window is stored with the "posX" and "posY" Key; the initial state (maximized or normal) under the "state" key. |
| | **posX** - x position of the upper left corner of the top-level window |
| | **posY** - y position of the upper left corner of the top-level window |
| | **state** - "normal" or "maximized" |
| Remarks: | The window position and initial state of the settings are only taken into account if the "position" style attribute of the window is "default" (not "center" and not |

| | |
|---|---|
| | "field").<br>If an application is popping up for the first time and no settings are available, the window is positioned automatically by the Window-Manager/GDC. |
| **Start Menu settings** | The **width** of the **StartMenu** (when the style is "tree"). |
| **Toolbar Settings** | |
| **Global Settings** | These settings are stored under<br>.**\Application\<*applicationName*>\Windows\<*windowName*>**<br>**\toolbars\<toolbarId>**<br><*toolbarId*> is the toolbar id for the window; the id starts at 0 and is increased when you add another window. |
| | **area** - the area where the toolbar is docked (left, right, top, bottom) |
| | **areaSettings** - coordinates of the toolbar |
| | **floating** - the toolbar can be floating ('1') or docked ('0') |
| | **orientation** - "vertical" or "horizontal" |
| | **styleAttribute** - styleAttribute "toolbarPosition" of the toolbar. |
| | **usesTextLabel** - specifies whether the toolbar shows the text. |
| **Table Settings** | |
| | The first time the table is sent, the GDC stores the initial settings prefixed by DVM_ : **DVM_numColumns**, **DVM_pageSize**, **DVM_sortColumn**, **DVM_sortType**.<br>Each time the table is sent again, GDC compares the initial settings with the stored settings. If something has changed, then the 4GL program has been changed, and the user-stored settings for this table are reset. |
| **Global Settings** | These settings are stored under **.\Forms\Table Columns**<br>**<*FormName*>.<*ArrayName*>**<br><*FormName*> is the "name" attribute of the "Form"-Node in the UI-Tree<br><*ArrayName*> is the "tabName" attribute of the "Table"-Node.<br>This allows the settings to be re-used in multiple applications. |
| | **numColumns** - the number of columns the table had when this setting was written; if a table with a different number of columns is created under the same settings identifier, these settings will be ignored. |
| | **pageSize** - the number of rows the Table had when the Table was closed; used to restore exactly this number of rows if the table is shown again. |
| | **pageSizeInitial** - the number of rows the Table had when the Table was created; if a table with the same identifier is created with a different number of rows the "pageSize" settings key is ignored. |
| | **pixelWidth** - the pixelWidth the table had when it was closed the last time; will be restored the next time it is shown. |
| | **sortColumn** - the sort column the table had when it was closed; will be restored if the table is shown again. A Reset of this saved column (to -1) is only possible by clicking on the Right Mouse Click-Context Menu at the Table header and |

| | |
|---|---|
| | choosing "Reset To Defaults". |
| | **sortType** - the sort Type "asc" or "desc" the table had when it was closed and a sortColumn was saved; will be restored if the table is shown again. |
| **Column Settings** | Individual column settings are stored under **.\Forms\Table Columns\<*FormName*>.<ArrayName>\<*columnNumber*>** |
| | **pixelWidth** - the pixelWidth this column had when the table was closed; will be restored the next time the table is shown. |
| | **realColumn** - the actual visible number (order) of the column; if the user had moved columns by drag and drop, this number is different from <columnNumber> and will be restored the next time the table is shown. |
| | **text** - "text" attribute of the "TableColumn" Node, used for debugging. |
| | **visible** - if a user used the context menu of the table header to switch off/on a column, the result of this operation is saved under this key and will be restored the next time the table is shown. |
| **TextEdit Settings** | |
| **Global Settings** | These settings are stored under **.\Forms\TextEdits\<*FormName*>.<*TextEdit*>** <*FormName*> is the "name" attribute of the "Form"-Node in the UI-Tree <*TextEditName*> is the "colName" attribute of the "Textedit"-Node. This allows the settings to be re-used in multiple applications. |
| Global | **height** - the height this textedit had when the form was closed, will be restored the next time the form is shown. |
| Global | **width** - the width this texedit had when the form was closed, will be restored the next time the form is shown. |

# Command Line

### Topics

- Command List
- Warnings
- Examples

---

## Command List

GDC handles the following options :

| | | |
|---|---|---|
| **Information** | | |
| `-h` | | Makes GDC display the About Box. |
| **Network, System** | | |
| `-p` | `new_port` | GDC will listen on the *`new_port`* port (if available). |
| `--port` | | |
| `-n` | | Starts a new instance of GDC. |
| `--new` | | |
| `-q` | | If the expected port (either 6400, or port specified by `--port)` is not available, GDC will stop (exit with -1). |
| `-D` | | Starts GDC in debug Mode (debug Tree and debug Console are active) |
| `-A` | `security level` | Sets GDC's security level regarding the Runtime System's connection. |
| `--Authentication` | | |
| `-R` | | Automatically starts the built-in RCP deamon. (Windows System only) |
| `-RCPd` | | |
| **Start Application** | | |
| `-S` | `shortcut_name` | If GDC has not been launched, GDC will start minimized; then, the shortcut named `shortcut_name` will be started. |
| `--Start` | | |
| `-s` | | If GDC has not been launched, GDC will start minimized, using the information given by -U, -H, -T, -P and -C to |
| `--startDirect` | | |

| | | |
|---|---|---|
| | | connect to a DVM. |
| -U<br>--User | user name used | The specified user name will be used when a Direct Connection starts.<br>This option can be used if you share GDC; then each user can create a link to the bin and differentiate the shortcut that will be launched. |
| -H<br>--Host | host name | The specified host name will be used when a Direct Connection starts with a defined shortcut (with -S), or starts directly (with -s). |
| -P<br>--Password | password | The specified password will be used when a Direct Connection starts with a defined shortcut (with -S), or starts directly (with -s). |
| -K<br>--KeepPassword | | The password specified with -P option will be kept in memory and no longer requested. |
| -C<br>--Cmd | command_line | The specified command_line will be used when a Direct Connection starts with a defined shortcut (with -S) or starts directly (with -s). |
| -T<br>--Type | connexion_type | Defines which protocol should be used when an application starts with -s. Values can be: TELNET, RLOGIN, SSH, SSH2. Default is SSH. |
| -w<br>--ShowTerminal | | Defines whether the terminal window is visible (when --startDirect option is used). The terminal is hidden by default. |
| -f<br>--ShowFirstLogin | | If a password is provided with --Password, GDC won't display a login box when starting a shortcut. If you explicitly want the login box to be shown, with password and user pre-entered, use the -f |

| | | |
|---|---|---|
| | | option. |
| **Start GDC** | | |
| `-a` | | Starts GDC in admin mode. |
| `--admin` | | |
| `-M` | | Starts GDC minimized. |
| `--Minimized` | | |
| `-X` | | Closes GDC if there is no longer an application or terminal running. |
| `--close` | | |

Note for Mac OS X users: command line can be used in either of the following ways:

- Starting the terminal application (Applications, Utilities), and then `./Applications/gdc.app/Contents/MacOS/gdc <command line>`. OSX expects the path to be absolute and not relative.
- Using the following Apple Script : `do shell script "./Applications/gdc.app/Contents/MacOS/gdc <command line>"`

---

## Warnings

- The `-S` and `-s` options must be used separately; `-S` is used to start an existing shortcut, and `-s` to start an application using the command line.
- When using `-s`, you must specify at least the host and the command line. The username and password will be prompted if needed.
- Even if you're using the `-q` option, GDC will first check whether another instance is already running. If you really want your GDC instance to stop if the port is not available, use `-n` and -`q` together. Using `-q` alone will stop GDC if the port is not free and not being used by another GDC.

---

## Examples

- `gdc -p 6350`

  Starts GDC on port 6350.

- `gdc -S demo`

Starts GDC, and the shortcut named "demo"

- `gdc -S demo -U smith`

  Starts GDC, and the shortcut named "demo" using "smith" as the user name.

- `gdc -s -T SSH2 -U smith -H server -P whatisthematrix -C "cd demo ; fglrun demo" -X`

  Starts GDC, then connects to "server" as the user "smith" with the password "whatisthematrix". Once connected, performs the specified command line "cd demo ; fglrun demo" and closes the GDC when all the applications or terminals are over.

# Screenshots

### Topics

- Overview
- Screen Shots

---

## Overview

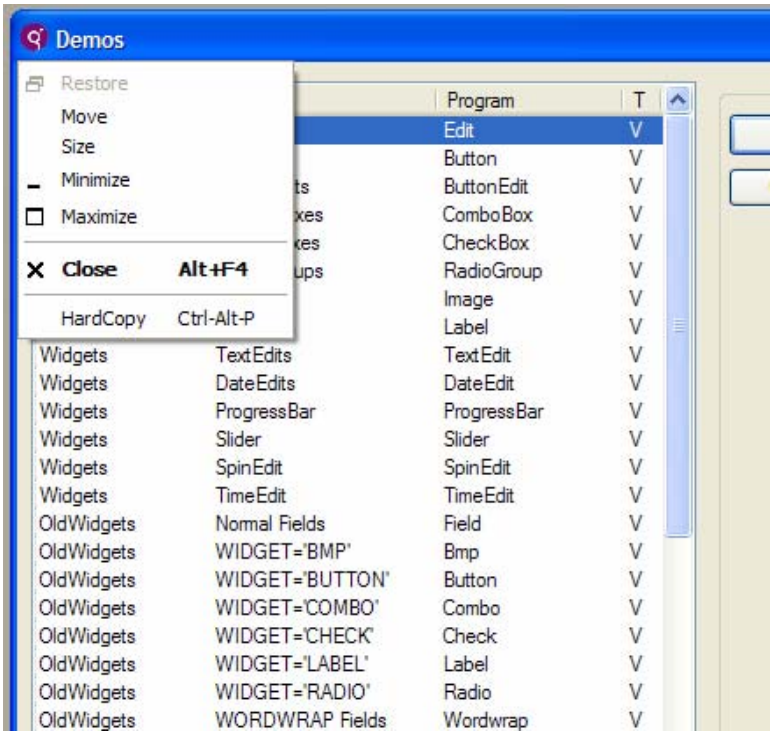
GDC provides a feature to send the current window to any installed printer. This will allow you to print a screenshot directly, without any other tool.

---

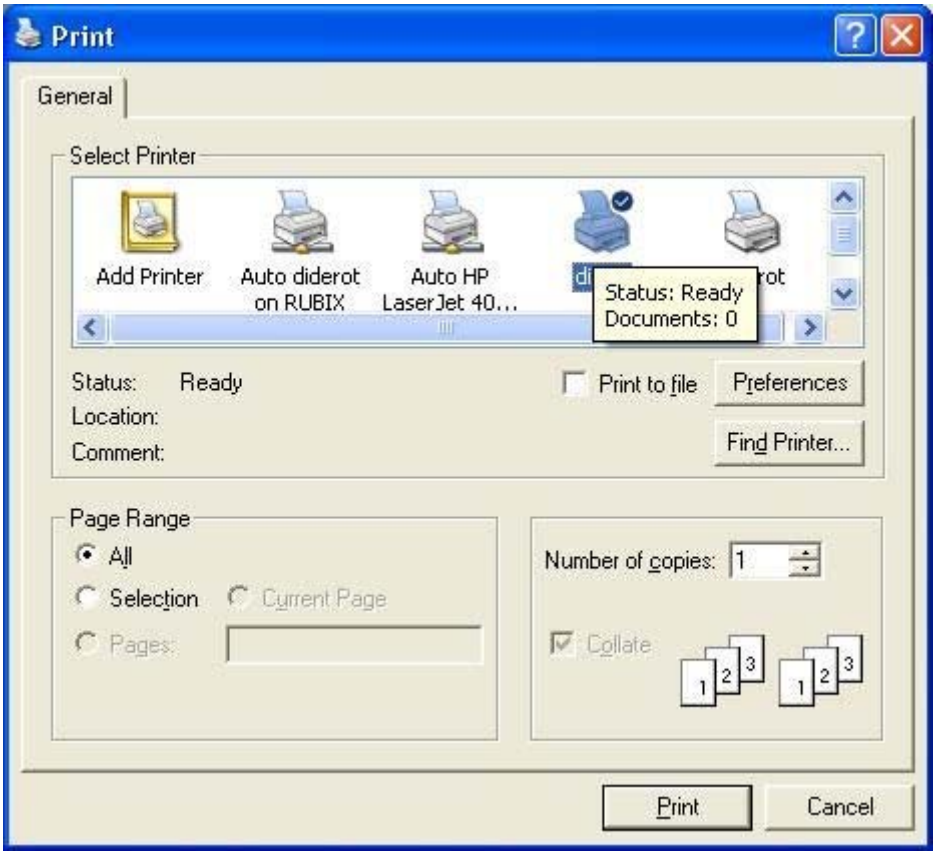
## Screen Shots

To call this feature, you can:

- press Control-Alt-P
- press Alt-Print_Screen (under Linux systems only, under Windows this combination will be used by the system to put the current screenshot into the clipboard)
- "Hardcopy" option in the System Menu (Windows systems only !)



This will open the classic "Print dialog" which will allow you to select your printer:




There you will be able to select the right printer, configure it, and then print the current window.

# Local Actions

### Topics



---

## Overview

Some features of the GDC are defined as "local", including "completely local" features like `copy`, `cut` or `paste`. Some others depend on the DVM but concern local behavior, like the navigation in a table.

To allow you to customize these features with accelerator, images, comment, etc., the features are integrated as local actions. They follow the same rules as Runtime System actions, but they are created by the Front End instead of the Runtime System.

You will be able to create actionViews for these actions, as you can for any other action.

**Example:**

```
01 BUTTON btn1 = nextfield;
```

When this button is pressed the focus will go to the next field.

**Example:**

```
01 <ActionDefault name="nextfield" accelerator="return">
```

The "return" key will be the accelerator to go to the next field.

---

## List of Local Actions

| Edition | | Shortcut Hotkeys |
|---|---|---|
| `editcopy` | copies the selected text into the clipboard | `CTRL-C` |
| `editcut` | cuts the selected text into the clipboard | `CTRL-X` |
| `editpaste` | pastes the content of the clipboard into the current field | `CTRL-V` |

| Navigation in fields | | Shortcut Hotkeys |
|---|---|---|
| nextfield | goes to the next field | TAB |
| prevfield | goes to the previous field | SHIFT-TAB |
| **Navigation in Tables** | | **Shortcut Hotkeys** |
| firstrow | goes to the first row | HOME |
| prevpage | goes back one page (with TABLE, it follows Internet Explorer behavior) | PRIOR |
| prevrow | goes to the previous row | KEY-UP |
| nextrow | goes to the next row | KEY-DOWN |
| nextpage | goes forward one page (with TABLE, it follows Internet Explorer behavior) | NEXT |
| lastrow | goes to the last row. | END |
| **Interrupt** | | **Shortcut Hotkeys** |
| interrupt | sends interrupt to the Runtime System | |

---

## Interrupt Local Action

The Interrupt action is enabled when the Runtime System is running without sending information to the front end. It allows the front end to send an interruption request, using a special communication system, named Out Of Band (OOB). When the Runtime System receives OOB, it sets INT_FLAG to 1 (Please refer to 'The Dynamic User Interface' section in the Genero Business Development Language documentation).

---

# Localization

Localization Support: list of encodings supported by GDC

| Encoding List | Description |
|---|---|
| Latin1 | |
| Big5 | Chinese |
| Big5-HKSCS | Chinese |
| eucJP | Japanese |
| eucKR | Korean |
| GB2312 | Chinese |
| GBK | Chinese |
| GB18030 | Chinese |
| JIS7 | Japanese |
| Shift-JIS | Japanese |
| TSCII | Tamil |
| utf8 | Unicode, 8-bit |
| utf16 | Unicode |
| KOI8-R | Russian |
| KOI8-U | Ukrainian |
| ISO8859-1 | Western |
| ISO8859-2 | Central European |
| ISO8859-3 | Central European |
| ISO8859-4 | Baltic |
| ISO8859-5 | Cyrillic |
| ISO8859-6 | Arabic |
| ISO8859-7 | Greek |
| ISO8859-8 | Hebrew, visually ordered |
| ISO8859-8-i | Hebrew, logically ordered |
| ISO8859-9 | Turkish |
| ISO8859-10 | |
| ISO8859-13 | |
| ISO8859-14 | |
| ISO8859-15 | Western |
| IBM 850 | |
| IBM 866 | |
| CP874 | |
| CP1250 | Central European |

| | |
|---|---|
| CP1251 | Cyrillic |
| CP1252 | Western |
| CP1253 | Greek |
| CP1254 | Turkish |
| CP1255 | Hebrew |
| CP1256 | Arabic |
| CP1257 | Baltic |
| CP1258 | |
| Apple | Roman |
| TIS-620 | Thai |

# Active X Overview

### Topics

- Overview
- Installation
- Uninstallation

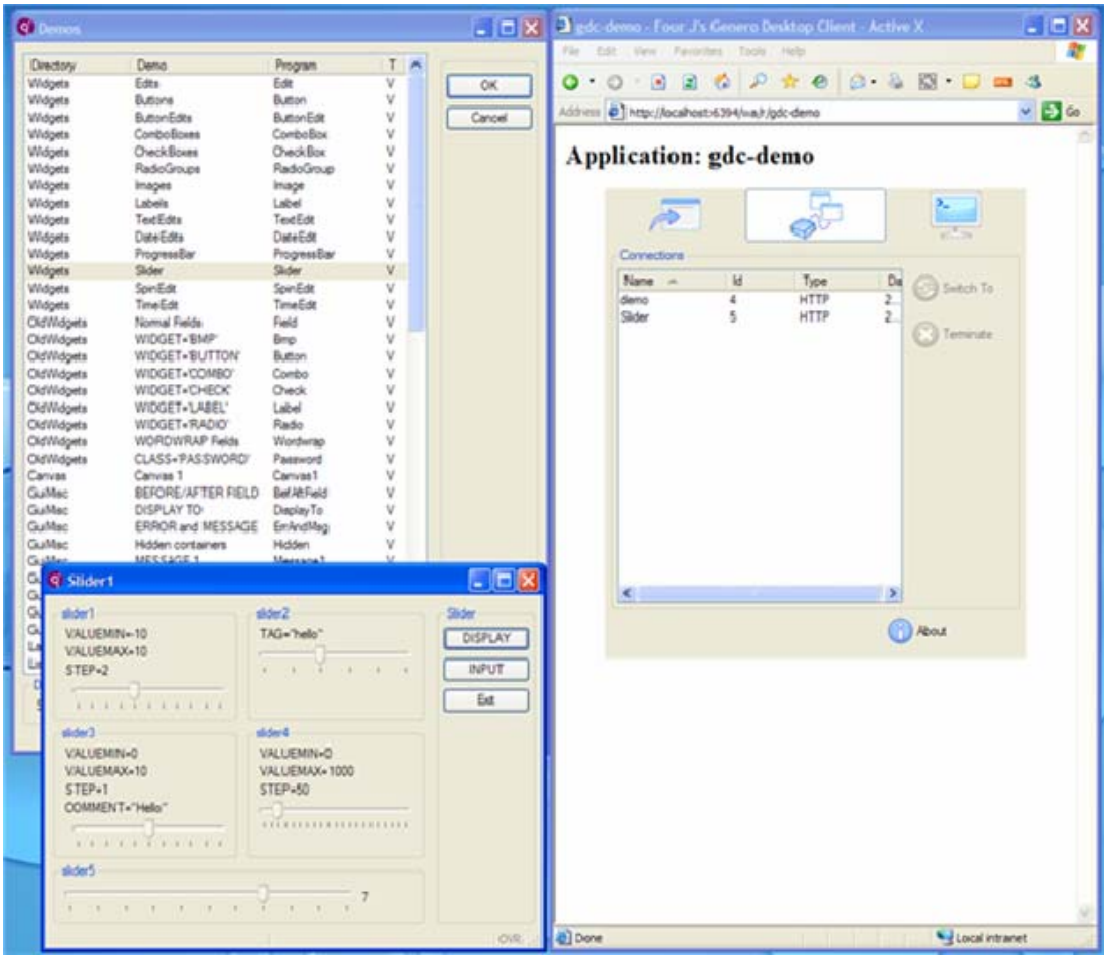
See also : Active X and GAS

---

### Overview

Genero Desktop Client is also an ActiveX. Packed into its .cab file, it can be put into a web site and then used directly with Internet Explorer.

Once a browser connects to the given URL, Genero Desktop Client will be displayed within a web page and can also be used exactly in the same way as the "classic" version.



The first time you access the Client in Active X mode, it installs itself and creates a shortcut in Windows Start Menu. Then Genero Desktop Client can be executed from the web page or directly with the shortcut.

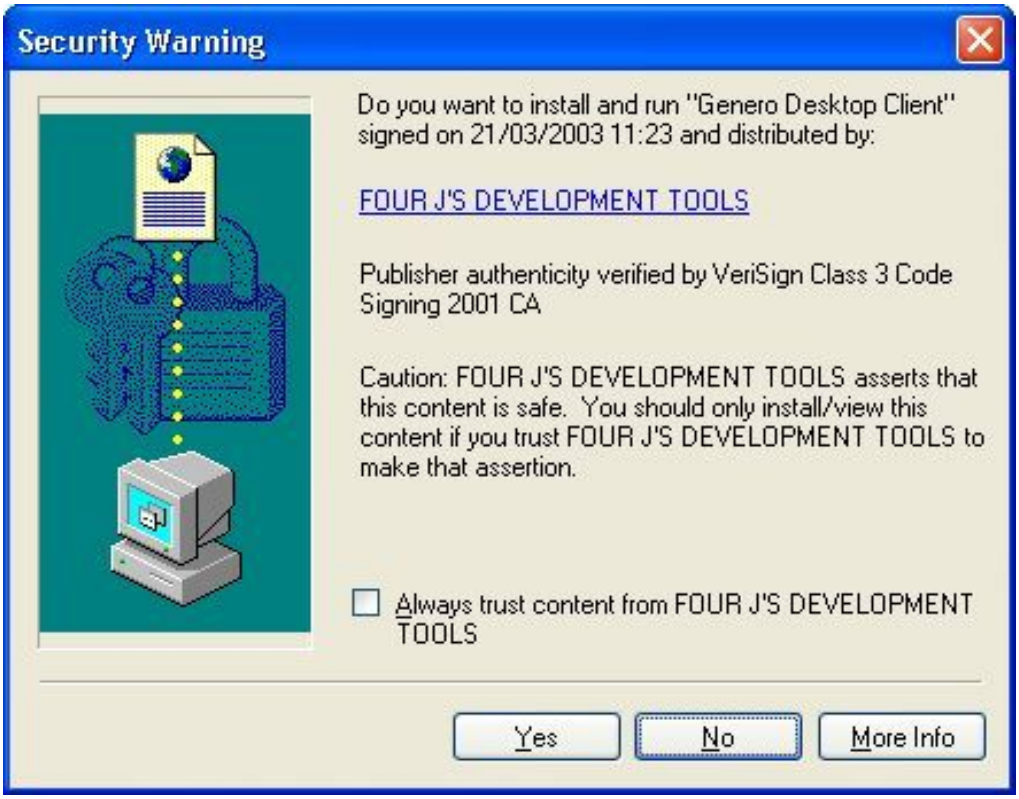
---

## Installation

**Note:** if you are running Windows XP Service Pack 2, the installation is described in the topic GDC and Windows XP Service Pack 2.

To install the GDC, connect to the Application Server where GDC-AX is installed.

Next, you get a certificate prompt. To install the GDC  press "Yes".




After a few minutes GDC will be installed on your computer in the "program files\Four J's Development Tools\Genero Desktop\" directory.

You will be able to launch the GDC through the Windows start menu (classic GDC) or via Genero Application Server (GDC ActiveX)
However it is launched, it can be used in the same way as the normal version.

If you get a new version of the Active X, this new version will automatically update your old version.

---

## Uninstallation

It is possible to uninstall the GDC, just as you would uninstall any Windows application, using the control panel.

---

# Active X and Application Server

### Topics

- Overview
- Installation
- After installation
- Installation Problems
- Template

See also : GDC Active X.

---

## Overview

GDC can be associated with Genero Application Server. With this system, you will be able to start a 4GL application simply with an Internet Explorer Browser and an URL.

---

## Installation

**Warning!** The Genero Application Server must be installed first. GDC will be installed into Genero Application Server.

### Windows Systems

To start the GAS installation:

1. Close all applications.
2. Execute the installation program.
3. Follow the directions that appear.

### X11 Systems

To start the GAS installation:

1. Close all applications.
2. Execute the installation shell using the following command:
```
sh installation-script.sh -i
```
3. Follow the directions that appear.

When prompted, select the directory where GAS is installed.

## After installation

See the Genero Application Server documentation for instructions on configuring GAS and creating applications for GDC.

A demo application is available at the following URLs:

*Mode*: `direct`
*URL*: `http://<server>:<port>/wa/r/gdc-demo`
*Where:* `<server>` is the IP of the server
`<port>` the port used by the gas

*Example:* `http://myServer:6394/wa/r/gdc-demo`

*Mode*: `cgi`
*URL*: `http://<server>[:<port>][<path>]cgi-bin/fglccgi/wa/r/gdc-demo`
*Where:* `<server>` is the IP of the server
`<port>` the port of the web server (default 80)
`<path>` is the path where the cgi is installed

*Example:* `http://myServer/cgi-bin/fglccgi/wa/r/gdc-demo`

**Warning!** You must use an URL with /wa/r. In the Shortcut System, you must use an URL with ja/r. Please refer to the FAQ for more information.

## Installation Problems

If the message: "`[Object not available! Did you forget to build and register the server?]`" displays when you start an application, the installation was not completely successful; the new binary has been installed, but it was not correctly registered in the Windows Registry. Versions 1.21.1d and greater implement a system to avoid this problem.

To solve this problem, use the **Control Panel / Add/Remove Programs** utility to uninstall the Active X, and start an application again.

If the problem still occurs, the only solution is to remove the Active X entries in the registry, by hand:

- `HKEY_CLASSES_ROOT\CLSID\{2311DF65-9D1A-4DDA-94AA-90568D989633}`
- `HKEY_CLASSES_ROOT\Interface\{A31C5317-4015-4080-BB1B-A6E948E99673}`

- HKEY_CLASSES_ROOT\Interface\{A90A2B09-D05D-426C-A471-D9FCA74FADA5}
- HKEY_CLASSES_ROOT\TypeLib\{0F2D0DFC-EB4B-421A-93BF-DD20351DB07B}
- HKEY_CLASSES_ROOT\AppID\{96503F06-661C-4A33-B543-C03F43B89587}
- HKEY_CLASSES_ROOT\gdc.MonitorView
- HKEY_CLASSES_ROOT\gdc.MonitorView1

---

## Template

When you install GDC Ax, a basic template is installed. This template can be modified to fit your needs; see the Genero Application Server documentation for information.

This section discusses parameters specific to GDC Active X:

### The OBJECT tag

The OBJECT tag loads the GDC.

```
<OBJECT
    Id="DesktopClient"
    Name="gdc"
    CLASSID="clsid:2311DF65-9D1A-4dda-94AA-90568D989633"
    CODEBASE="gdc.cab#version=1,21,1,7"
    height=440
    width=395>
[Object not available! Did you forget to build and register the
server?]
</OBJECT>
```

### Notes:

1. `CLASSID` is related to the identifiers of GDC Active X; `clisid` is a unique identifier.
2. `CODEBASE` indicates where `gdc.cab` is; `#version` is used to define which version is used.

**Warnings:**

1. The Four J's versioning system uses a letter as the last version number (e.g. 1.21.1g), but Active X systems can only use numbers. The letter is then transformed into a number (g -> 7).
2. Microsoft has released an update for Internet Explorer that modifies the way Active X is handled. The template must be changed accordingly.
   See IE ActiveX update (KB9 12945), Ms Technet about KB 912945, and MSDN about Active X activation

### The object functions

| Function name | Definition | Description |
|---|---|---|
| `setSrvUrl:` | The url used to start the html page ; usually : "`location.href`". | Indicates to GDC Ax the URL used. This is needed to correctly initialize the communication between the GDC and the Genero Application Server (GAS). |
| `setPictureUrl:` | Default remote path for pictures ; usually "`"$(pictures.uri)"`" | Indicates to GDC Ax where to search for images stored on the server side.<br><br>**Warning**! This requires GDC Version 1.21.1g or greater and GAS Version 1.21.1a or greater |
| `setAppName:` | Application name as defined in the GAS configuration file ; usually "`"$(application.id)"`" | Indicates to GDC Ax that it has to start the given application.<br><br>**Warning**! Without this parameter, the application will never start. |

### Example

#### HTML template:

```
<HTML>
 <HEAD>
  <TITLE>
   $(application.id) - Four J's Genero Desktop Client - Active X
  </TITLE>
  <META http-equiv="expires" content="1">
  <META http-equiv="pragma" content="no-cache">
 </HEAD>
 <BODY BGCOLOR="#FFFFFF" onload="startIt();"
onbeforeunload="preventClose();">
  <H2> Application: $(application.id) </H2>
  <CENTER>
   <DIV id="divgdc"/>
  </CENTER>
 </BODY>
 <SCRIPT src="$(connector.uri)/fjs/activex/fglgdcdefault.js"></SCRIPT>
 <SCRIPT language="javascript">

function startIt()
{
 // first activate ActiveX.
 var gdc=loadGDC("divgdc", "GeneroDesktopClient", "2,0,1,1");

 // this is to ensure that any popup window will appear in front of
```

```
the browser
 gdc.setFocus();

 // the serverUrl must be set BEFORE starting the application
 if ("$(connector.uri)" != ""){
   gdc.setSrvUrl(location.protocol + "//" + location.host +
"$(connector.uri)/wa/r/$(application.id)?$(application.querystring)");
 } else {
   gdc.setSrvUrl(location.href);
 }
 gdc.setPictureUrl("$(pictures.uri)");
 gdc.setAppName("$(application.id)");
 return false;
}

function preventClose()
{
 event.returnValue = "Genero Desktop Client";
}

 </SCRIPT>
</HTML>
```

**fglgdcdefault.js file:**

```
function loadGDC(gdcdiv, id, minversion)
{
var d=document.getElementById(gdcdiv);
d.innerHTML='<OBJECT NAME=gdc Id=' + id + '
CLASSID=clsid:2311DF65-9D1A-4dda-94AA-90568D989633
height=440 width=395 CODEBASE=gdc.cab#version=' +
minversion + '>'
+ '[Object not available! Did you forget to build and
register the server?]</OBJECT>';
return document.getElementById("GeneroDesktopClient");
}
```

The function **preventClose** will be called when Internet Explorer is about to close, and will display a short message. If the user presses 'cancel', the page won't be closed.

# Security

### Topics

- Overview
- Security Level 1
- Security Level 2 and 3

---

## Overview

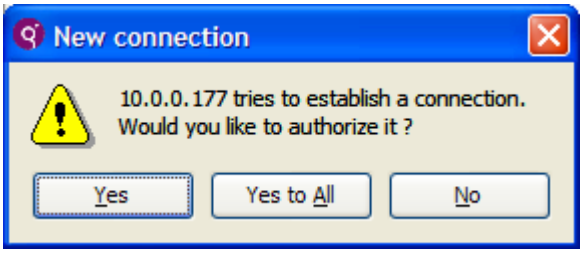
In previous versions, the Genero Desktop Client accepted all connections that arrived on the listening port, without any verification.
In Genero 2.0, the security level has been raised to "level 2".
You may change the security level with the command line "-A" option, or through the Security Option panel.

---

## Security Level 1

Command Line : `gdc -A 1`

When the runtime system starts a connection, the Genero Desktop Client user is warned by a message in a pop-up (or dialog) window.




- Select '`Yes`' and the Genero Desktop Client accepts this connection and only this connection. Any other connection causes the pop-up window to display the warning message again.
- Select '`Always`' and the Genero Desktop Client accepts this connection and any other connection from the same host. This setting is saved when the Genero Desktop Client closes and reapplied when the Genero Desktop Client is restarted.
- Select '`No`' and the Genero Desktop Client rejects this connection. As a result, the application will not run on the front end.

"Always" authorized hosts are saved into $GDCDIR/etc/hosts.xml. You can modify this file if needed, or remove this file if you want to reset the authorized list.

---

## Security Level 2 and 3

Command Line : `gdc -A 2` or `gdc -A 3`

**Warning! This only works when using a direct shortcut to start an application.**

Security level 2 and 3 use the following mechanism:

1. When started, the Genero Desktop Client generates a random key.
2. When you start a shortcut, @FGL (and @FGLxxx) sets this key into the `_FGLFEID` environment variable.
3. When `fglrun` starts, it reads the environment variable and sends the key back to the Genero Desktop Client.
4. The Genero Desktop Client compares the internal key and the key sent back by the runtime system. It only accepts a connection with the identical key. When a wrong key connection is encountered, if security level 2 has been set, a popup displays and asks whether to allow the wrong key connection to connect. If security level 3 has been set, the wrong key connection is simply rejected.

**Warning! If the runtime system you are using does not handle this feature, you won't be able to run an application at this security level.**

---

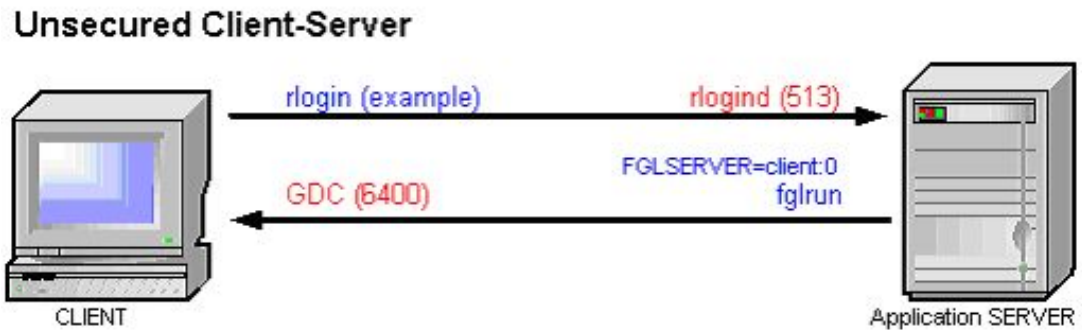
# GDC and SSH

### Topics



---

## Overview




SSH stands for "Secure SHell". It was designed to replace the 'r' commands like rlogin and rsh, because they offer no real security. SSH encrypts all data end-to-end, offers data compression, and prevents snooping and connection hijacking. One additional feature it offers is *port forwarding*.

Port forwarding allows an application on one computer to connect to a local port and have its data tunnelled through an SSH session to the other computer. This does not require you to open any ports on your firewall router, other than the port you already have open for SSH - a distinct advantage. If you have firewalls, this is an advantage because Genero needs to establish a connection from the client to the server to start the user application, and another connection originating from the user application to the client to display the graphical user interface. When Genero establishes a connection from the server to the client, it can use the existing connection to tunnel the graphical connection.

**Warning!** **Any environment that uses firewalls or connections over the Internet should use SSH or SSH2 for those connections**. You should never send unencrypted data such as account numbers, social security numbers, and passwords through the Internet. Some companies might even consider using secure shell connections for

internal use when handling sensitive data such as accounting and payroll information. At any point along the way, someone could be monitoring the data - for network diagnostics or possibly with malicious intent. Whatever the reason, encryption is simple and offers peace of mind.

SSH is comprised of two main components, the server component "sshd" and the client component "ssh". Genero provides its own client component (built-in).

---

## Prerequisites

**Things you should know about your system:**

In order to determine how to proceed, you will need the following information about your environment:

- Is there a server-side firewall between the server and the client?
- Is there a client-side firewall between the server and the client?
- Is encryption needed for all your data?
- Are you using a VPN (Virtual Private Network) or NAT (Network Address Translation)?
- Will you need protection from inactive sessions timing out?
- Do you have more than one server to access from outside the firewall?
- Do you have more than one client accessing servers outside the firewall?

We recommend reading about SSH and how to configure it. We will not cover this topic in this document.

**How do I make sure data is encrypted?**

To ensure that your data is encrypted, select SSH or SSH2 Both offer data compression and port forwarding; the difference is SSH2 has different implementation code and a more advanced encryption algorithm than SSH.

If you are using the shortcut buttons in the Genero Desktop Client, two connections are established between the client and the server. The first connection is established from the client to the server, in order to log in and start the application. The second connection is made from the server's application to the client, in order to display the graphical data. Use the chart below to determine which settings you will need.

| Type of connection | Command encrypted | GUI encrypted |
|---|---|---|
| rlogin | | |
| telnet | | |
| ssh | ✔ | |
| ssh port forwarding | ✔ | ✔ |
| ssh2 | ✔ | |
| ssh2 port forwarding | ✔ | ✔ |

**What connection method should I use?**

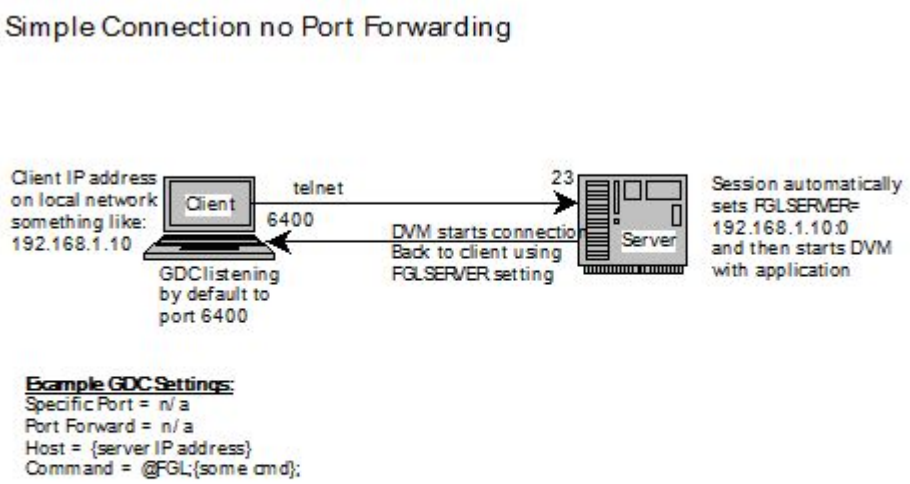
Knowledge of your configuration will be necessary to make Genero work properly. Refer to the "Things you should know about your system" section to find out what information will help. Use the matrix below to determine which connection methods will support what you are trying to do. Currently the SSH or SSH2 with Port Forwarding provides the most flexible connectivity.

| | telnet | rlogin | SSH | SSH + Port Forwarding | SSH2 | SSH2 + Port Forwarding |
|---|---|---|---|---|---|---|
| Firewall or NAT on Server Side | 1 | 1 | 1 | 1 | 1 | 1 |
| Firewall or NAT on Client Side | 2 | 2 | 2 | ✔ | 2 | ✔ |
| Firewall or NAT on Both Sides | 1,2 | 1,2 | 1,2 | 1 | 1,2 | 1 |
| Private Network | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| VPN (Same as Private Network) | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Encryption of all Data | | | | ✔ | | ✔ |
| Password/login Encrypted | | | ✔ | ✔ | ✔ | ✔ |
| Keep Alive | | | | ✔, 3 | | ✔, 3 |

1 - Requires configuring the server side firewall router to open or forward the port used by sshd.
2 - Requires configuring the client side firewall router to open or forward the port(s) used by the GDC.
3 - May require changes to firewall connection timers if firewalls are involved.
- Indicates full functionality with no changes.

# GDC and SSH: Simple Setup

The simple setup assumes that you are on a corporate LAN with no firewalls. All methods of connections are possible here (telnet, rlogin, ssh, ssh2, with/without port forwarding) without any special set up. Using SSH or SSH2 will work fine and will offer encryption. The GUI connection will be made on the default port 6400. FGLSERVER will be set to '<client IP> :0' and it will expect to be able to access that IP and port directly.




If you don't want any encryption or compression, select rlogin or telnet as your method of connection.

What if you want to connect to a port other than 6400 for the GUI? Specify the option "-p <port> " on the command line for GDC,  and GDC will listen on that port for the GUI connections. The FGLSERVER will have its information adjusted accordingly. For example, execute "gdc -p 7400". When you look at the value of FGLSERVER, it will contain "<client IP> :1000". It would contain "<client IP> :0" if the default of port 6400 was used (the number displayed after the colon is the port number that you specified minus 6400, the default number.)

If you do port forwarding while using "-p 7400" on the GDC command line, the offset number after the colon will still be your Port Forward value minus 6400. This is because fglrun doesn't care what port you are listening on the client side, only what port needs to be connected on the server side. The tunnel takes care of connecting to the correct port on the client side. Using @FGL keeps everything automatic. If you have a need for multiple GDC's running at the same time, see Port Forwarding and Firewalls.

# Port Forwarding and Firewalls

### Topics

- Port Forwarding
- Client-side Firewalls
- Server-side Firewalls

---

## Port Forwarding

Port Forwarding is used in situations where you want all data encrypted, no session timeouts, or simple firewall setup.

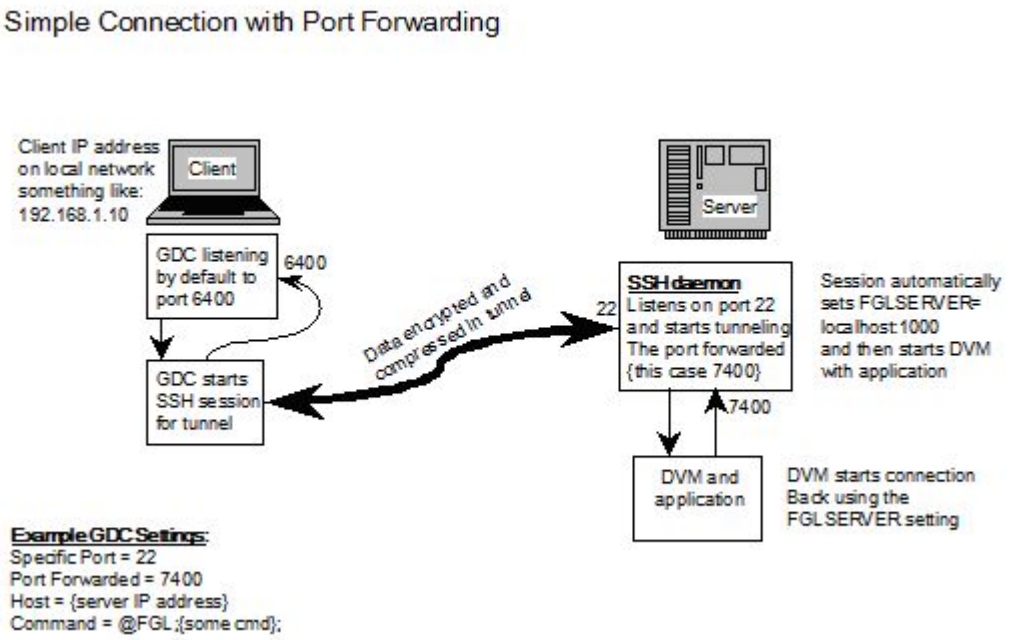



Figure 1

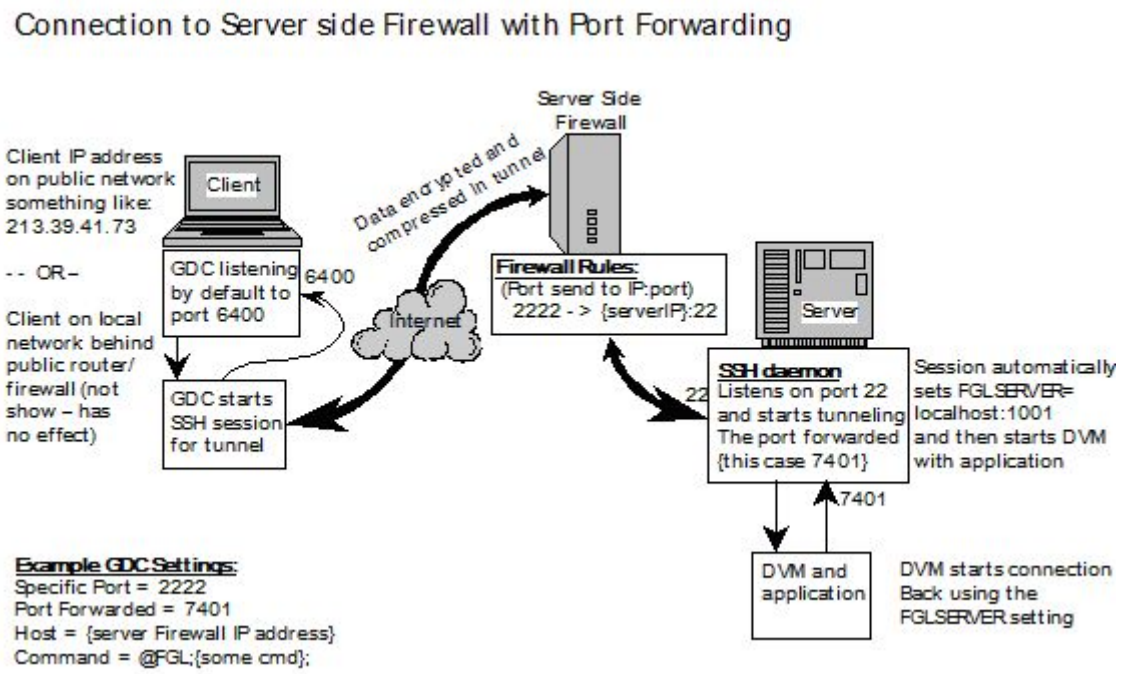



Figure 2

Figure 1 shows a simple configuration that does not involve a firewall. Sshd, the portion running on the server, will accept a connection from the GDC (client) and start your application. It will also set up a listener for a port that the application will connect to for the GUI. This port is then tunnelled through the existing connection to the client, where the client will display the application. Note that both sides still use ports to accomplish this.

You must have ssh installed and set up on the server. If you are expecting to access your Genero application from somewhere on the Internet, you will most likely have a firewall router and must open a port on your router to allow connections to the sshd. See figure 2 for an illustration of this.

Sshd is by default listening on port 22. You can set a port on the firewall to forward to sshd. Whatever port number you use must be set in the GDC using the "Specific Port" field:

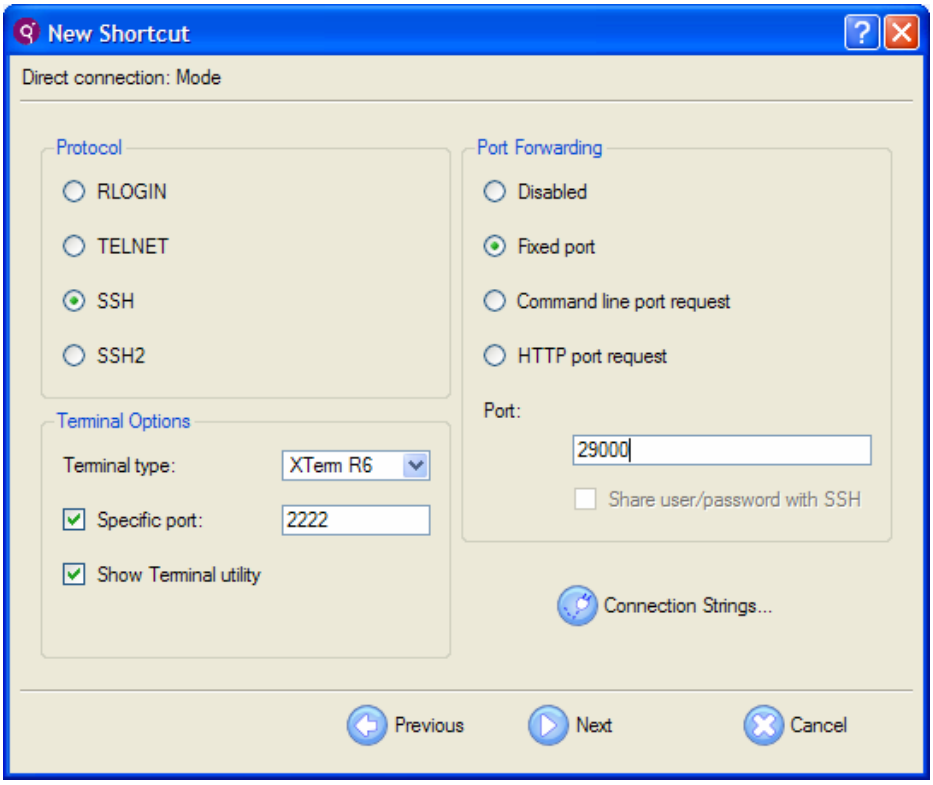



Figure 3

In figure 2 we have set our firewall router to forward port 2222 to our server sshd. There is no reason you couldn't just use port 22 and pass it straight through to your server. If you have more than one server you need to access from outside your firewall, you must use different port numbers and map each server with a different port number. Most routers will allow the destination port to be different from the origination port. For example, a rule could be entered into your firewall router to forward port 2222 to a server on port 22;  set another rule to direct 2223 to a different server on port 22, and so on. More details on this are in the Firewall Server Side section.

In figure 3 we have also set Port Forwarding to 29000. This will cause the sshd running on the server to listen to port 29000 for connections from the application. The FGLSERVER environmental variable will be set to 'localhost:22600'. It is localhost because it will be tunnelled and sshd is running on the same machine. The 22600 is an offset for the port. To clarify, Genero GDC listens on 6400 by default and any number after the colon in FGLSERVER is added to this number. So 22600+6400 works out to be the port we specified on the client side configuration, 29000.

To use *Automatic Port Forwarding*, you can specify a command line that will execute on the server and return a free port number. As this application is really depending on the system where the Runtime System is installed, we can't provide a version for each system. This program must be used in combination with the GDC connection strings system.

Another way to achieve automatic port forwarding is to have a service running on an HTTP server. This can be a CGI. The program must return lines containing information for the coming SSH connection. One line is always like the following:
<attribute name>=<attribute value>
For the moment, the attributes managed are "host" and "port", which can indicate the host IP to connect to and the port the sshd will listen to on the server side. By default, the host IP is the same as the HTTP server machine.

Click "Next" for the configuration.

The IP address is that of the server machine unless the firewall on the server side is doing NAT (Network Address Translation). If it is doing NAT, the IP address should be set to the address of the firewall router. Put @FGL on the line labeled "Command Line", so Genero can set the FGLSERVER variable for you when it logs into the server. FGLSERVER will have the port number corresponding to the "Port Forwarding" value you put in the previous screen. Several commands can be placed on the command line and executed in succession. In Unix you use a semi-colon (;) and in Windows you use two ampersands (&&) to separate the commands.

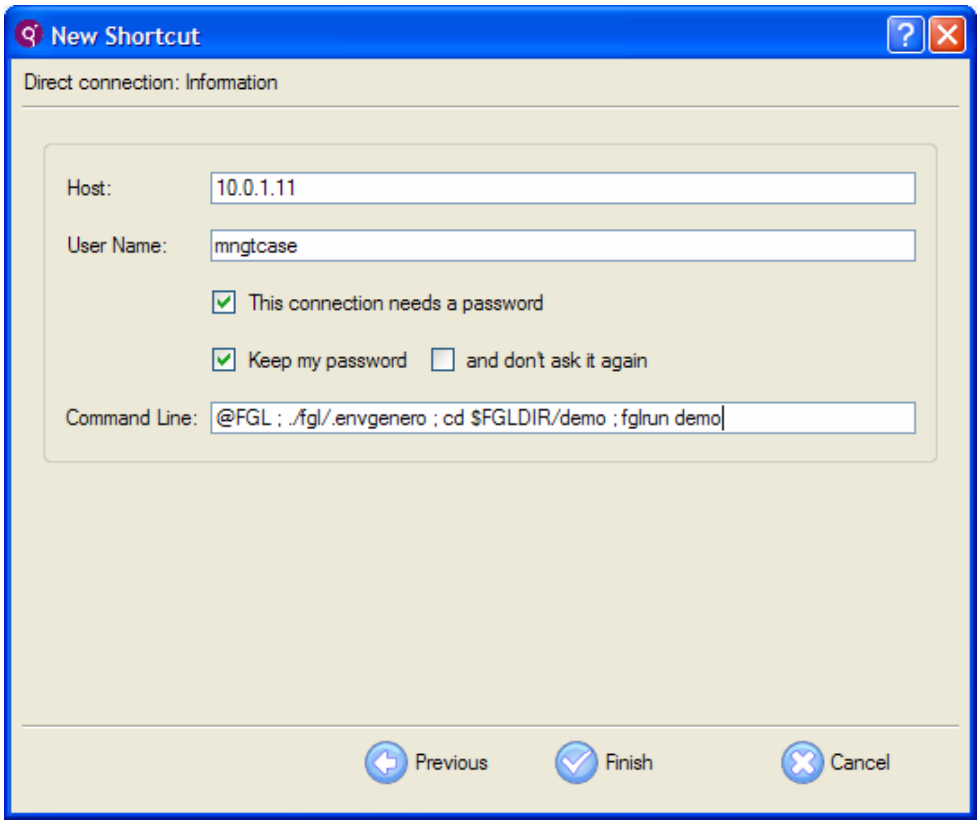



Figure 4

## Firewalls: Client Side

This section details how to configure the client side.

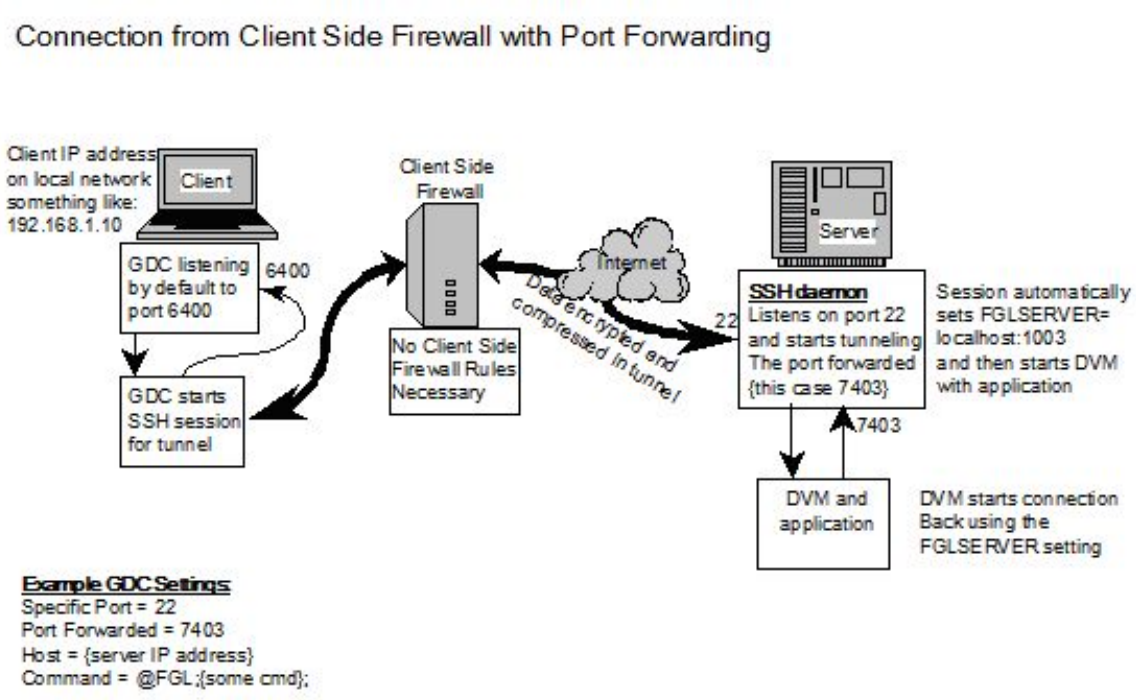



Figure 5.1

If you have a client side firewall, you cannot connect directly to your clients from outside the firewall. There are two solutions to this problem:

- First, you can set up port forwarding while using SSH or SSH2 (Fig. 5.1). This is by far the easiest and most secure method to connect without the help of a VPN.
- The second method requires adding rules to the router to allow connections (Fig 5.2). The set up of the router will be covered here; port forwarding is covered in a separate section.

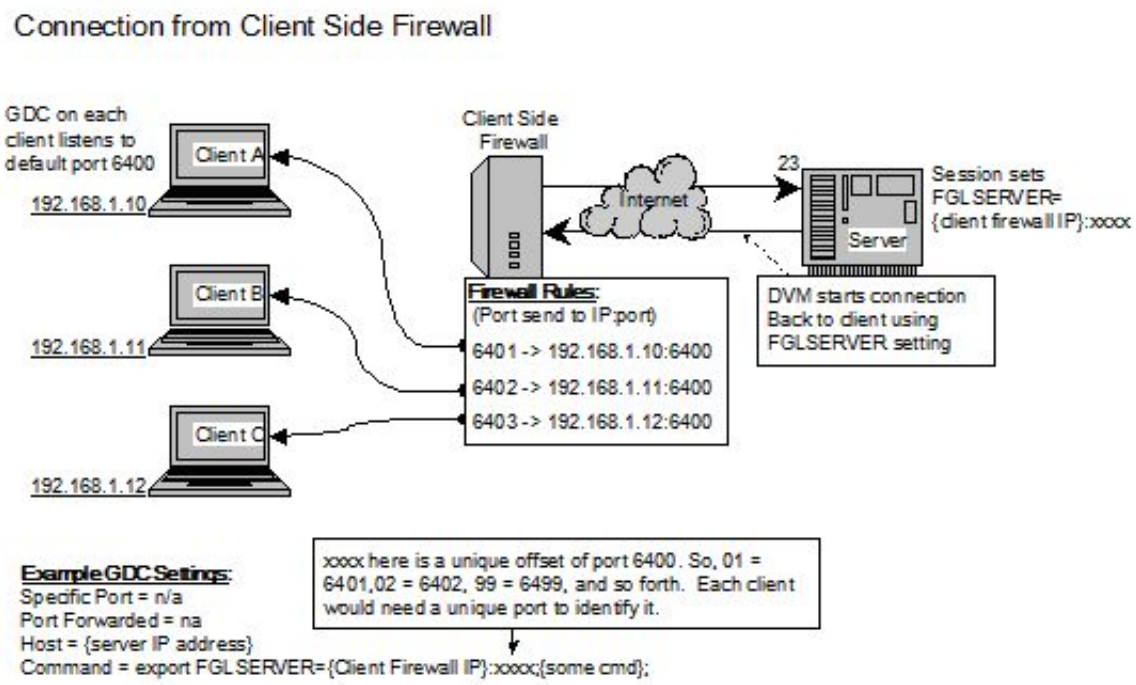



Figure 5.2

The router will need rules added to take a connection coming in on a specific port and direct it to one of your clients. The way Genero is normally configured, all clients would use port 6400. If you only have one client, you can add a rule to the router to forward 6400 to the client on port 6400. If you have more than one client, you will need to allocate other ports on the router to forward to the other clients.

Note: In the examples shown the internal addresses are not public IP addresses. If you have public IP addresses on each client, you can open port 6400 for each of the clients.

Example rule:

```
Incoming 6400 -> 192.168.1.10:6400
```

If you have more than one client, you can map them as follows:

```
Incoming 6401 -> 192.168.1.10:6400
Incoming 6402 -> 192.168.1.11:6400
Incoming 6403 -> 192.168.1.12:6400
```

Another option if your firewall won't allow you to change the destination port number:

```
Incoming 6401 -> 192.168.1.10:6401
Incoming 6402 -> 192.168.1.11:6402
Incoming 6403 -> 192.168.1.12:6403
```

This last example requires that you start the GDC with the -p option, causing it to listen on a different port from the default port.

```
>gdc -p 6401
>gdc -p 6402
```

If you are setting up multiple clients in this manner, you may want to avoid starting the first client on 6400; any mis-configured new clients will pop up on that user's console unexpectedly.

On the command line of the GDC shortcut setup, assign FGLSERVER to be the IP of the firewall router with the corresponding port of the router. This must be hard-coded, since there is no way for the client computer or Genero to know how the connection is established.

For example, if the client firewall router's IP address to the Internet is 213.39.41.73, and port 10000 is mapped to the client 192.168.0.53 port 6400, then the entry in the router would be:

```
Incoming 213.39.41.73:10000 -> 192.168.0.53:6400
```

The command line in the GDC would look like this:

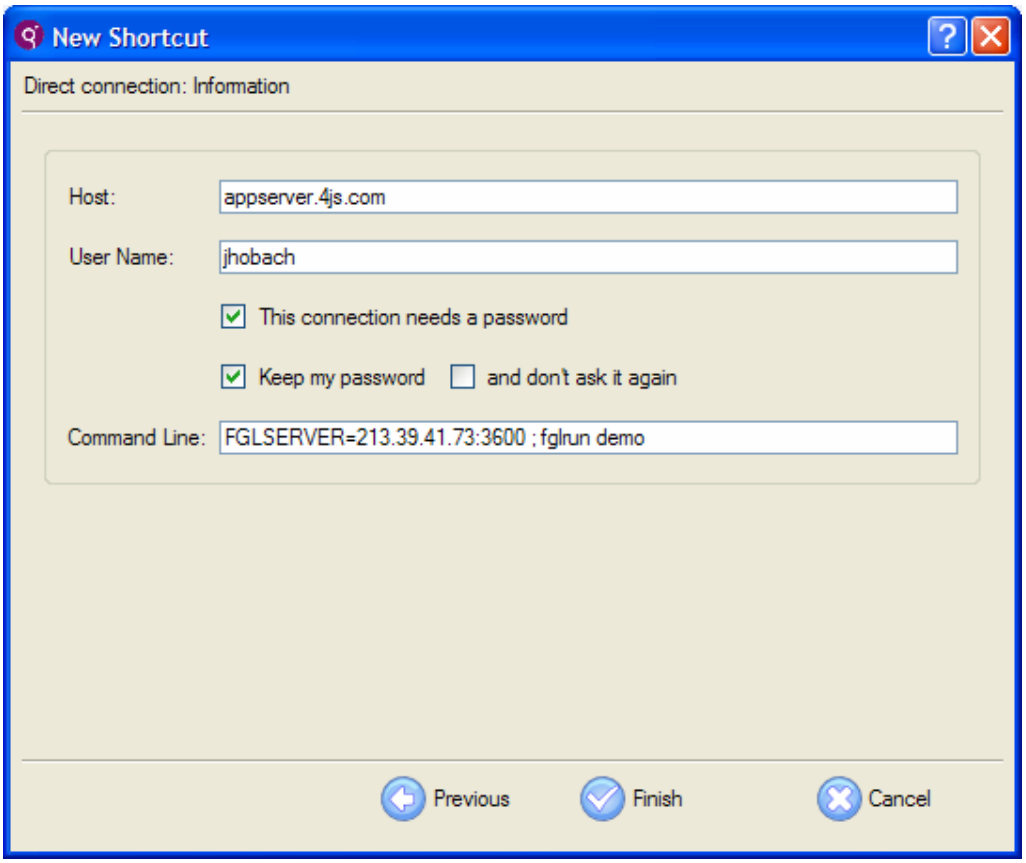



Figure 5.3

The FGLSERVER variable is normally set using @FGL, but that would set FGLSERVER to the IP of the local client machine and the port specified when the GDC was started with -p.  If the IP addresses used behind the firewall are public, this would be OK. If the addresses are not public, however, we must use the IP address of the router, and let the router translate and forward it. If the router is translating the port, then we must use the port that the router is expecting.

In our example the port that the router is looking for is 10000. The FGLSERVER port value must be set to 10000 minus 6400, resulting in 3600. This is because FGLSERVER=<ip> :0 tells Genero to connect on port 6400. The number after the colon is added to 6400.

---

## Firewalls: Server Side

Having a server side firewall is the typical configuration on many systems. There is only one method for doing this, whether you use telnet or ssh: map a port to be forwarded to the server in the firewall router. It is not advised that you use rlogin from the Internet for security reasons; that is usually why you have a firewall.

Decide which method of connectivity will be allowed, and determine what port you will use to forward to this service. If there is only one server involved, you can use port 22 for ssh or 23 for telnet and forward them straight through to the server. But if there are several servers involved and they do not have public IP addresses, you will need to pick different ports on the firewall router and let the router forward those ports to the different internal servers.

See Figure 6.1 for an example of how to do this for a telnet connection. Notice that the returning GUI path doesn't require any special handling unless there is a client side firewall. For details on this see the Client Firewall section.

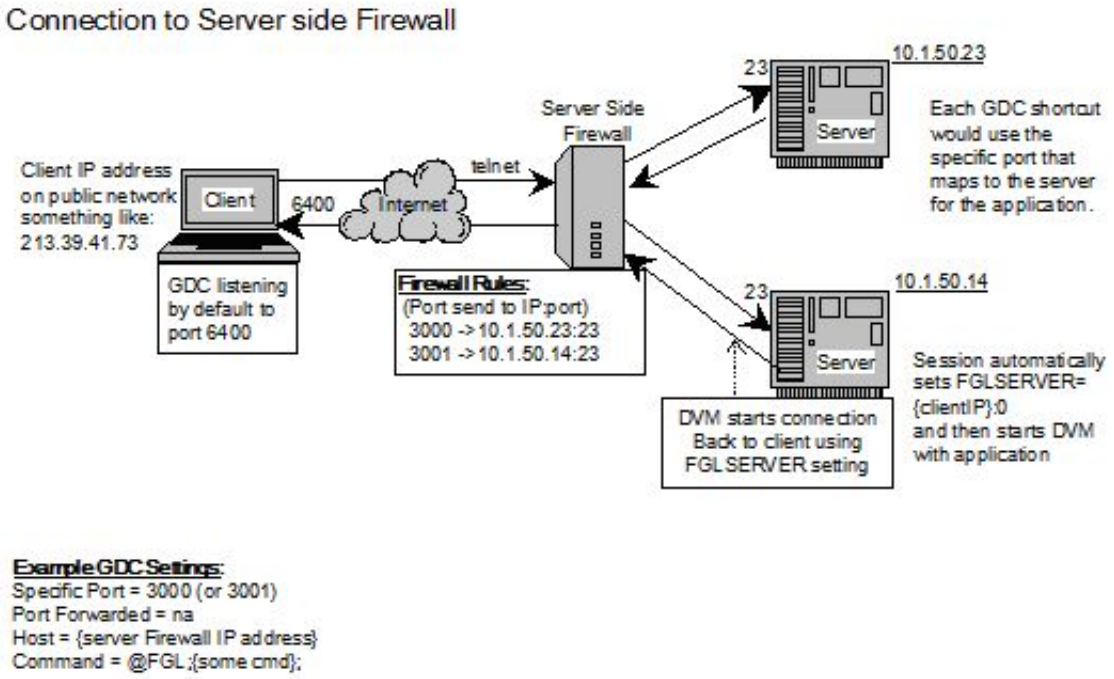



Figure 6.1

See Figure 6.2 for an example of how to do this using ssh with port forwarding.

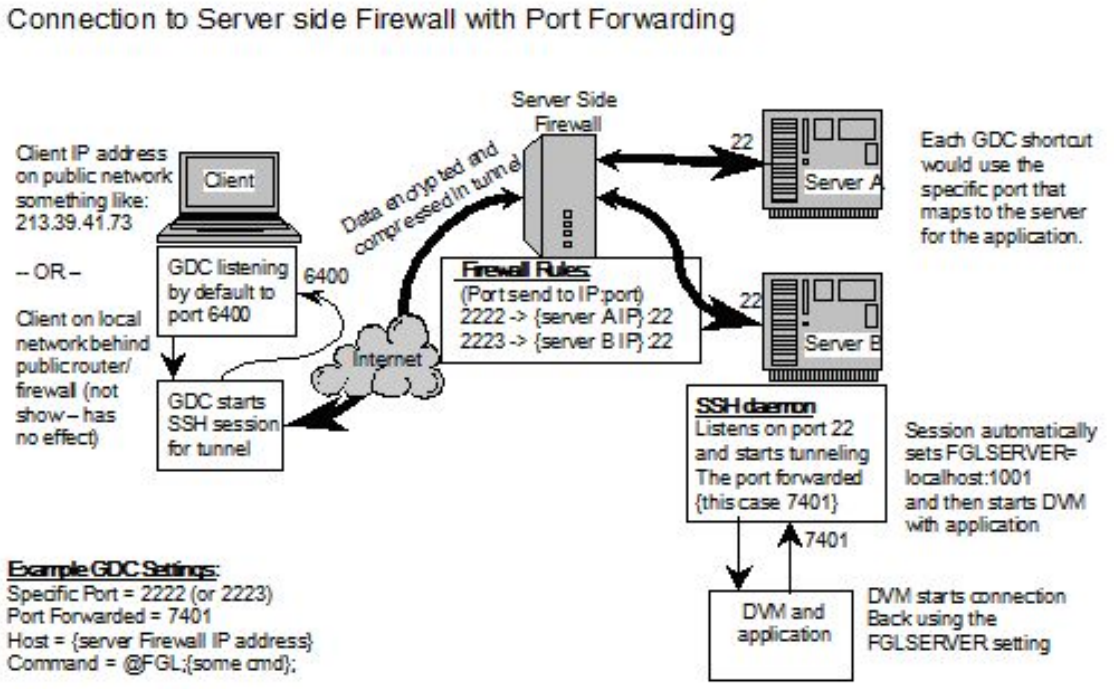



Figure 6.2

The client GDC would connect to the server firewall router on port 3000 to access server 1, and port 3001 for server 2. We chose these ports arbitrarily; almost any port could be used. Numbers below 1024 are reserved for well-known services, so choose numbers above 1024.

Using port forwarding will work without modification because the GUI interface is tunnelled through the initial connection, and the port it tells the server application to use is a local port to the server. Of course, the same methods as above must be used if there is more than one server. Using telnet or non-port forwarded ssh will work also, because connections for the GUI originating from behind the server firewall will be allowed out without special mapping. If there is a client side firewall as well, see client side firewall configuration.

**Example:**

We have two servers that will be accessed via clients somewhere on the Internet. They will use ssh2 with port forwarding to simplify client set up and keep things secure. The firewall on the server side has an IP address of 192.168.50.2 (only valid for this example). We have mapped the two servers:

```
213.39.41.73:3000 -> 10.1.50.23:22
213.39.41.73:3001 -> 10.1.50.14:22
```

The GDC client will need to be configured as well:

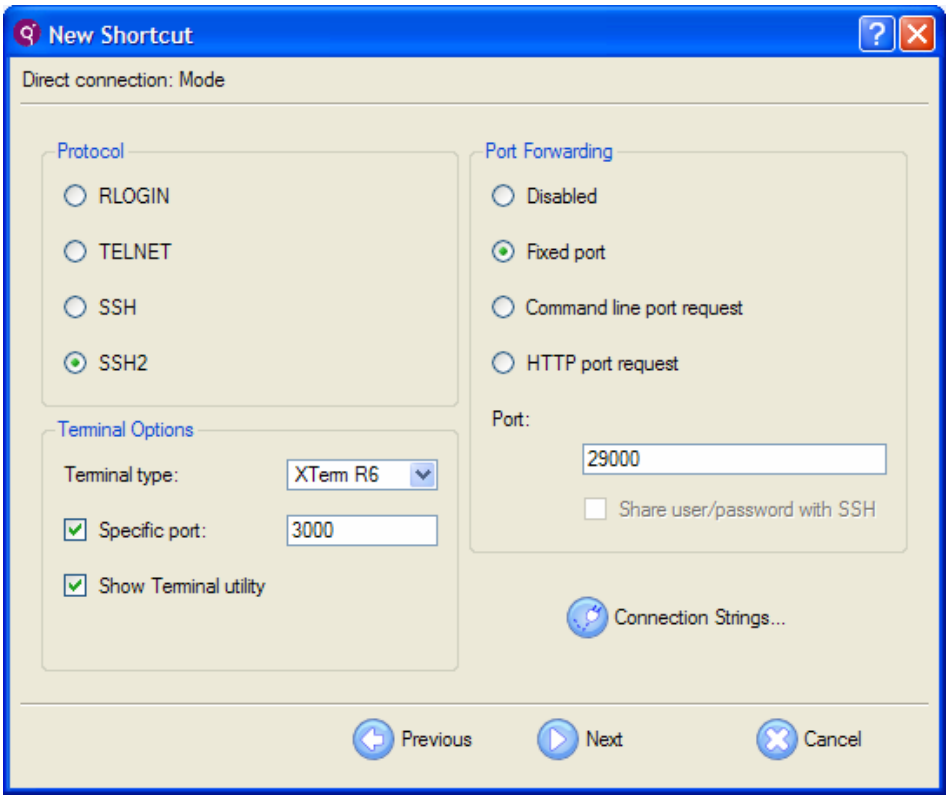



Figure 6.3 Showing configuration for access to Server 1

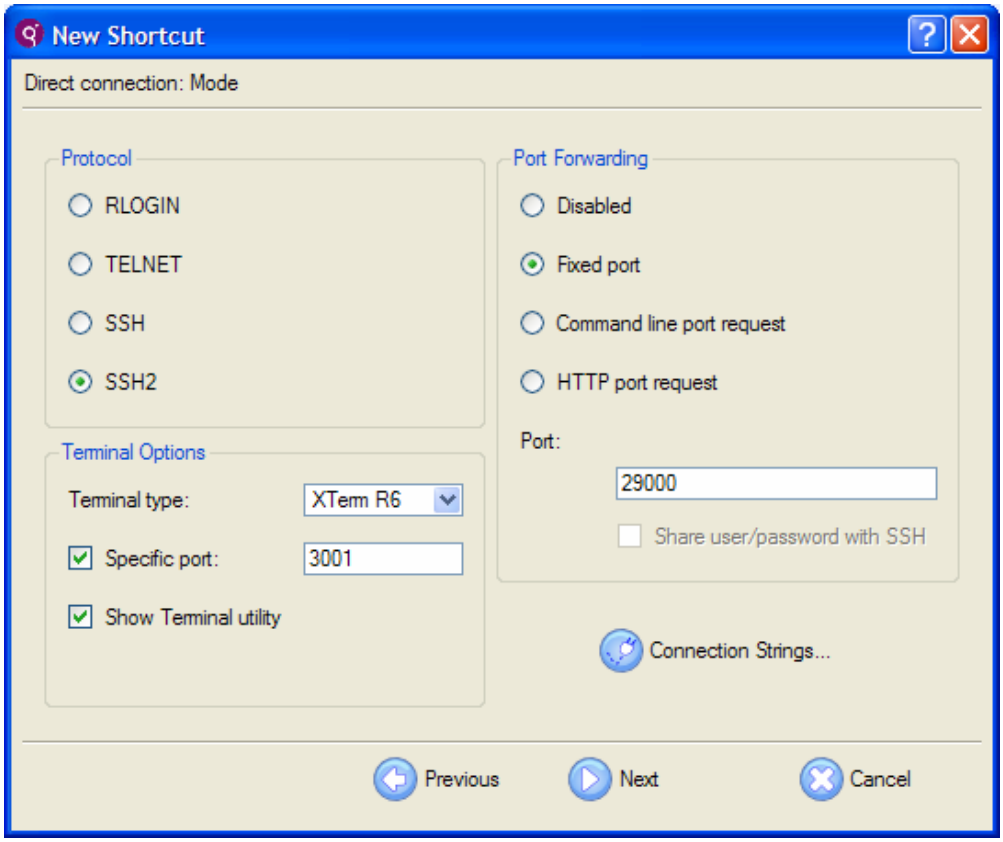



Figure 6.4 Showing configuration for access to Server 2

Figures 6.3 and 6.4 show how to access each server by specifying the appropriate port for each, one with 3000, the other 3001. This will allow the firewall router on the server side to direct each to the appropriate server. The IP address used would be the IP of the router.

Keep in mind that if you have two users accessing the same server, you must manually select a different port forward number to keep them unique. See Possible Configuration Problems.

# Implementing a Secure Server with Genero Desktop Client

In an enterprise deployment, it is typical for the Genero Desktop Client to be configured to launch in the default user mode with all application shortcuts pre-defined.

When the "-a" or "--admin" option is specified, however, the Genero Desktop Client launches in admin mode, and the user is able to modify existing shortcuts or create new shortcuts of their own. Therefore, when in admin mode, a Genero Desktop Client user with sufficient knowledge can modify the string passed to the server (Unix or Linux) and effectively execute any command. While this is expected behavior -- if they can log in to the server, they can enter commands -- this ability can present a problem in some environments.

The following paragraphs explain how to implement a secure server preventing Genero Desktop Client users from executing arbitrary commands, by preventing client access to the (Unix or Linux) command line or shell while still allowing Genero applications to be started. This is accomplished by not giving them access to the shell, yet allowing the Genero Desktop Client to pass values to the system to indicate which application to start.

**Warning!** This is intended to be the framework for a larger implementation and should be reviewed by your system administrator for any security concerns.

### Topics



---

## Prerequisites

To implement a secure server, the following prerequisites must be met:

- Genero Desktop Client, version 1.32.1f or greater
- Unix or Linux platform
- SSH configured on the server
- Familiarity with Bourne or Korn shell programming
- Access to root for implementation

## Solution Overview

When a user logs in, the system determines which shell to give them, based on a value in the /etc/passwd file. We will replace this shell with a shell script that will parse the values passed to it and set the environment accordingly. The application that is started will be from a list of valid applications; no other options will be accepted (thus controlling what a user can do).

### Passing Values to the Script

The Genero Desktop Client must pass specific information to the script:

- The *application name* must be passed if more than one application exists. You can add additional logic to the script to control which users have access to specific applications.
- The *port* accepting connections for the Genero Desktop Client is important so that the application can connect back to the Genero Desktop Client to display information.
- The *two security values* prevent anyone from spoofing the connection. The DVM must make a socket connection to the Genero Desktop Client for the application screens and user interaction. The @FEID and @FEID2 contain a value that must match on both the client and server. The Genero Desktop Client compares the @FEID value it has internally and the one it received from the DVM attempting to connect. If they do not match, it assumes an application it did not start is trying to connect and rejects the connection. Likewise, @FEID2 contains a value that the DVM must receive from the Genero Desktop Client in order to validate that the Genero Desktop Client is the one that started it. These security values are enabled by specifying '-A 3' as a command-line argument when starting the Genero Desktop Client.

### Auto Port Forwarding

With version 1.30, the automatic assignment of the port to use for port forwarding was added to the feature set of the Genero Desktop Client. Port Forwarding is the term used for tunnelling with ssh. It allows applications to connect back to the client via a port that is open on the server, tunnelled through the ssh secure client connection, then connects to the Genero Desktop Client on the client. The port is specified by the client, but it is usually not known whether this port is in use on the server prior to initiating the connection. In an enterprise this could be a problem, because every forwarded port must be unique between users.

The solution is to ask the server system for a port number to use. Because there is no way to reserve the port, we must get the number and open it quickly. Once we have the port opened for our session, we will have it until we log off and the connection is closed. We use a small "C" program that uses network system calls to allow the server to assign a port number. This port number is produced by the operating system by incrementing some internal OS counter and issuing numbers from a pool. If the port it would assign is

in use, it will automatically increment the value until it finds an unused port. The next number it assigns to us, or to any other network request, will be managed the same way. This process insures to a large degree that the number we get will not be reassigned or used for some time, certainly long enough for our purposes.

### Process Summary

- Log in.
- Get a port number from the system.
- Close the connection.
- Establish another connection and provide that port number for the tunnel.
- Log in (again).
- Start the application.

In normal situations the terminal activity of this process is hidden. The user simply sees their application appear.

---

## The Shell Script

The shell script accepts the information on the command line and parses it, assigning values as needed to start the application. The application name is matched in a case statement, preventing direct execution of what the user sends.

The script provided later in this section is intended to be an example, and we expect you to tailor it according to your needs. Save it in a location where it can be executed but not changed by your users. Edit the /etc/passwd file to make a user call the script instead of a shell. Here is an example of the user “user1” running the script named "gdcstart".

```
user1:x:569:569::/home/user1:/home/user1/gdcstart
```

The script LOGIN_SCRIPT is designed to recognize the difference between being started from sshd or from telnetd. You could modify it to handle either condition differently. For example, you may want it to start an application in text mode when accessed via telnet, or in GUI mode when accessed via ssh.

---

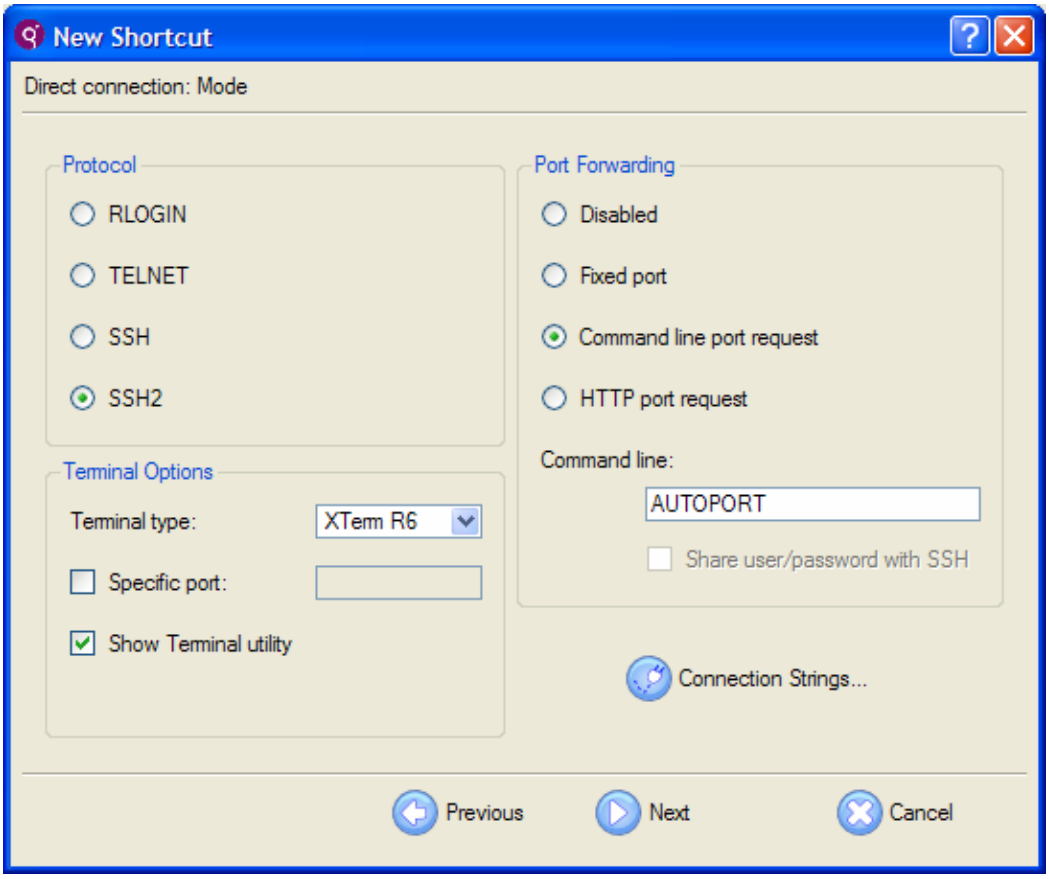
## Setup SSH Login

An advantage of using ssh and port forwarding is that the GUI information is encrypted during transmission. However, the unused port must be assigned on the server for the tunnel -- a difficult task if you are the system administrator. To solve this, we ask the

server to tell us what port to use. This section shows how to implement this solution while maintaining system security.

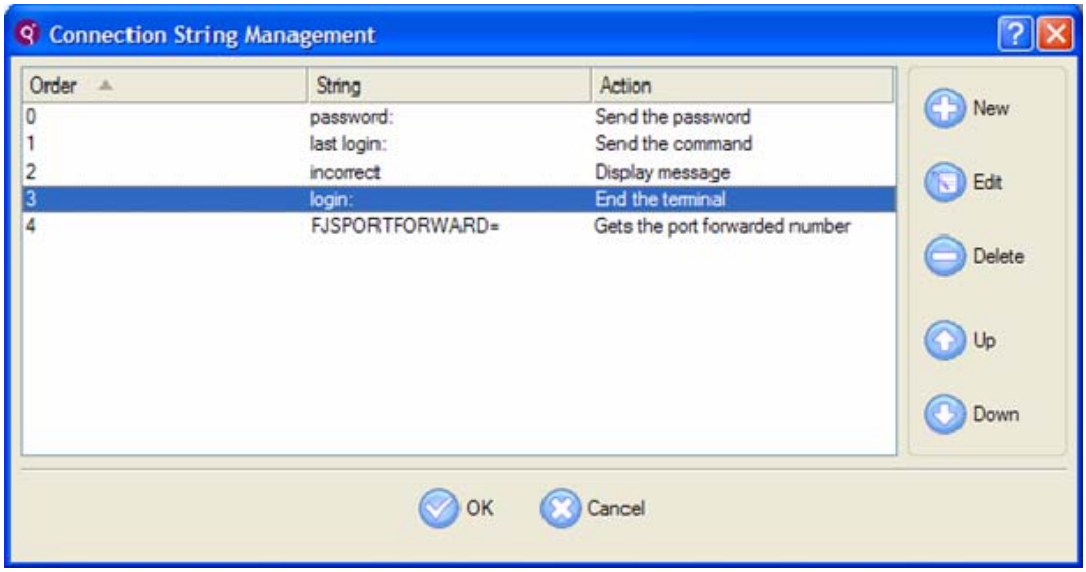
As stated previously, we use a shell script to start the requested application instead of giving the user a shell; the login script is used for that purpose. In order for the script to work properly, the information in the Command Line field of the Genero Desktop Client shortcut must be altered accordingly to launch the application. The automatic assignment of the port forward number must also be set up.

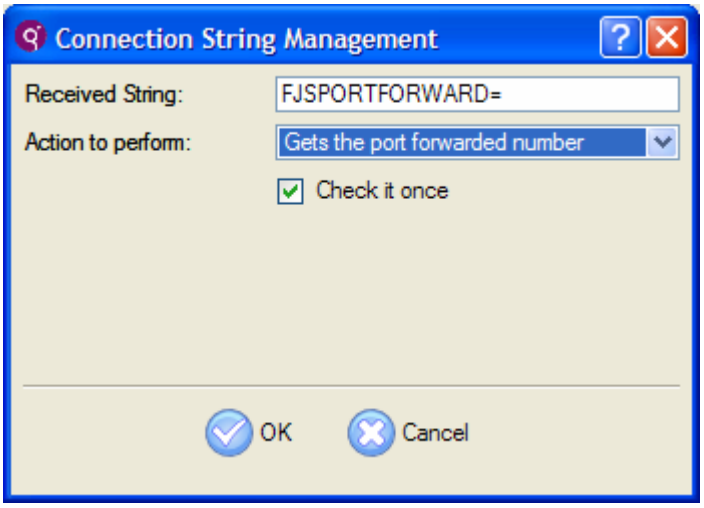
This is the Genero Desktop Client shortcut entry for using ssh. In the Automatic field, we have specified AUTOPORT. This corresponds to an option near the end in the login script.



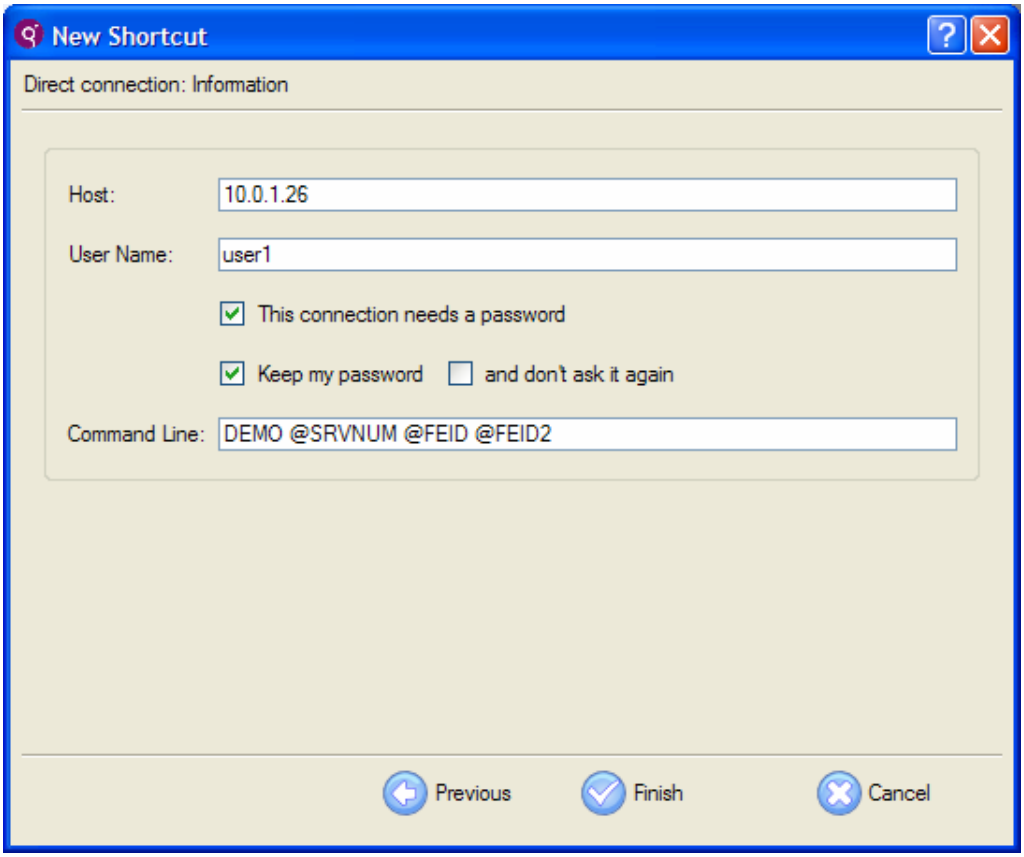

When the login script receives “AUTOPORT”, it executes a program called “autoportfind". The –e option will make it output a string like “FJSPORTFORWARD=nnnn” where nnnn is the port number provided by the operating system. The string matching rule we use looks for FJSPORTFORWARD= and retains the number following the ‘=’. This session is then closed and a new session is started using that number as the port to forward. It should not matter where in the sequence this rule is added.

You will also need to make an addition in “Manage Connection Strings”:




Normally, the Command Line is passed to the shell that is started when a user logs in. Since we are are using our shell script, the Command Line is where we specify the application to run, and pass the port number and the security fields. In our example we want to run the demo application. The command “DEMO” can be changed to your own application name, and an entry in the login script can then be added to start your application.



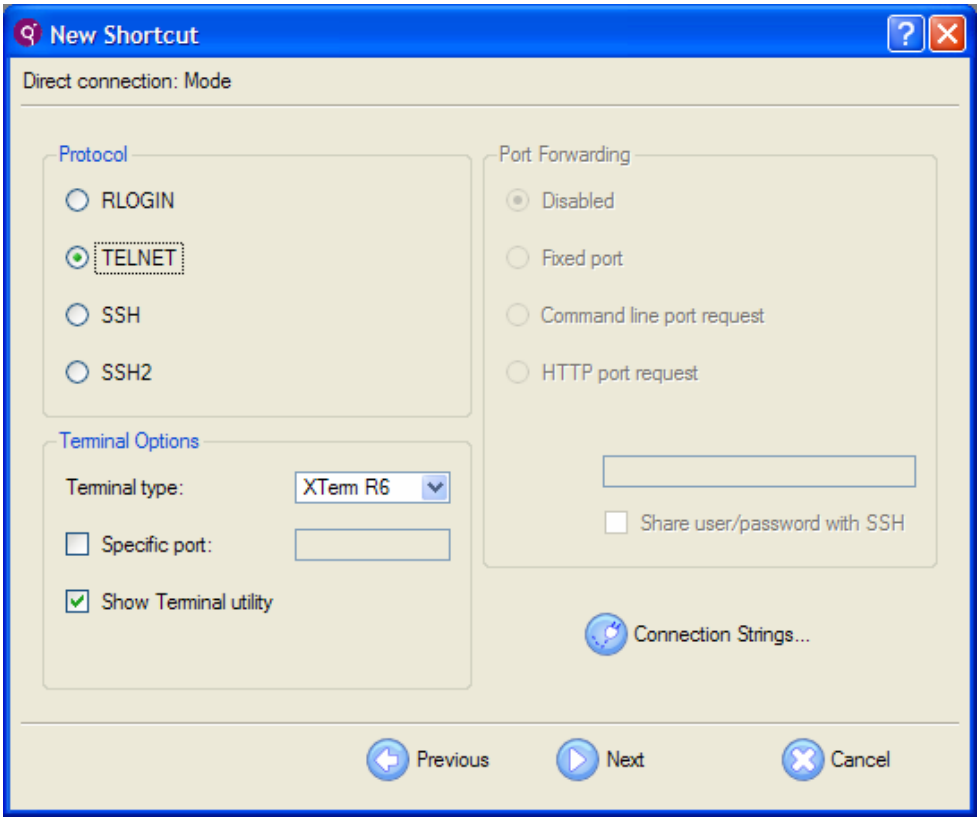

When the shortcut is run it will log in using AUTOPORT first. This will match a case statement in the script, and return a string “FJSPORTFORWARD=nnnn” where nnnn is a port number. Genero Desktop Client will then close the connection, and log in again using that port for the port to forward (tunnel) and pass it on the command line of the server @SRVNUM. This is what the login script uses to set the environment for the execution of the command DEMO. When using Port Forwarding, the server (127.0.0.1) is always the target for FGLSERVER, (and therefore only the port number is needed).

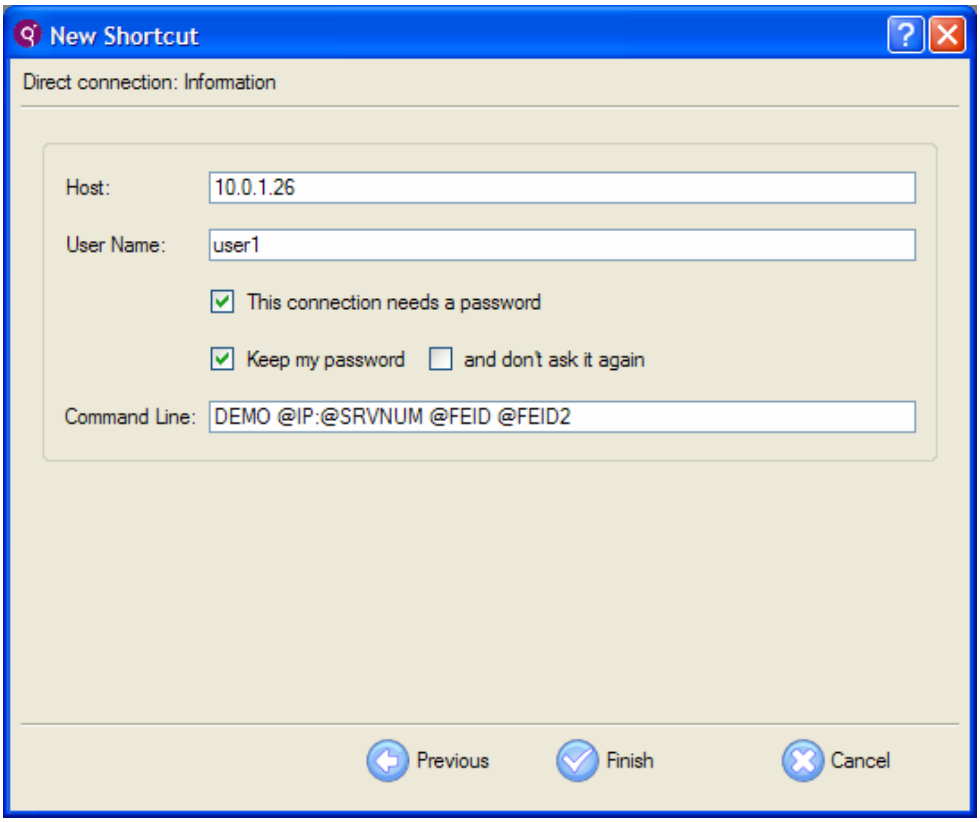
---

## Setup Telnet/Rlogin

Telnet and rlogin don’t offer port forwarding, so the setup is a bit simpler. But they also don’t give the flexibility needed when going through firewalls, and offer no encryption or privacy like ssh.

You simply need to pass the required arguments via the command line, and the login script sets the environment and launch the application.


New Shortcut
Direct connection: Mode
Protocol
RLOGIN
TELNET
SSH
SSH2
Port Forwarding
Disabled
Fixed port
Command line port request
HTTP port request
Share user/password with SSH
Terminal Options
Terminal type:
XTerm R6
Specific port:
Show Terminal utility
Connection Strings...
Previous
Next
Cancel


New Shortcut
Direct connection: Information
Host:
10.0.1.26
User Name:
user1
This connection needs a password
Keep my password
and don't ask it again
Command Line:
DEMO @IP:@SRVNUM @FEID @FEID2
Previous
Finish
Cancel

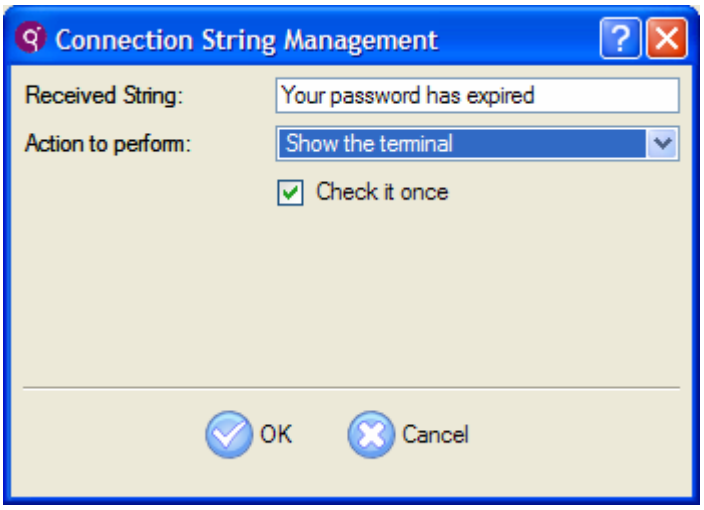
With ssh and tunnelling, the IP address is not needed because the tunnel is listening on the same server that will run the application. But with Telnet and rlogin, we must pass the client machine's IP and port using @IP and @SRVNUM. The security values are passed as well, so the environment is complete. For the Genero Desktop Client to make use of the security values, you must start it with the option "–A 3" on the command line of the Genero Desktop Client. Put your application name in place of DEMO, and make an entry in the login script accordingly.

---

## Checking for Expired Password and Changing Password

### Handling Expired Passwords

To handle expired passwords, edit the shortcut and add a filter under "Manage Connection Strings". The entry should look like the following:

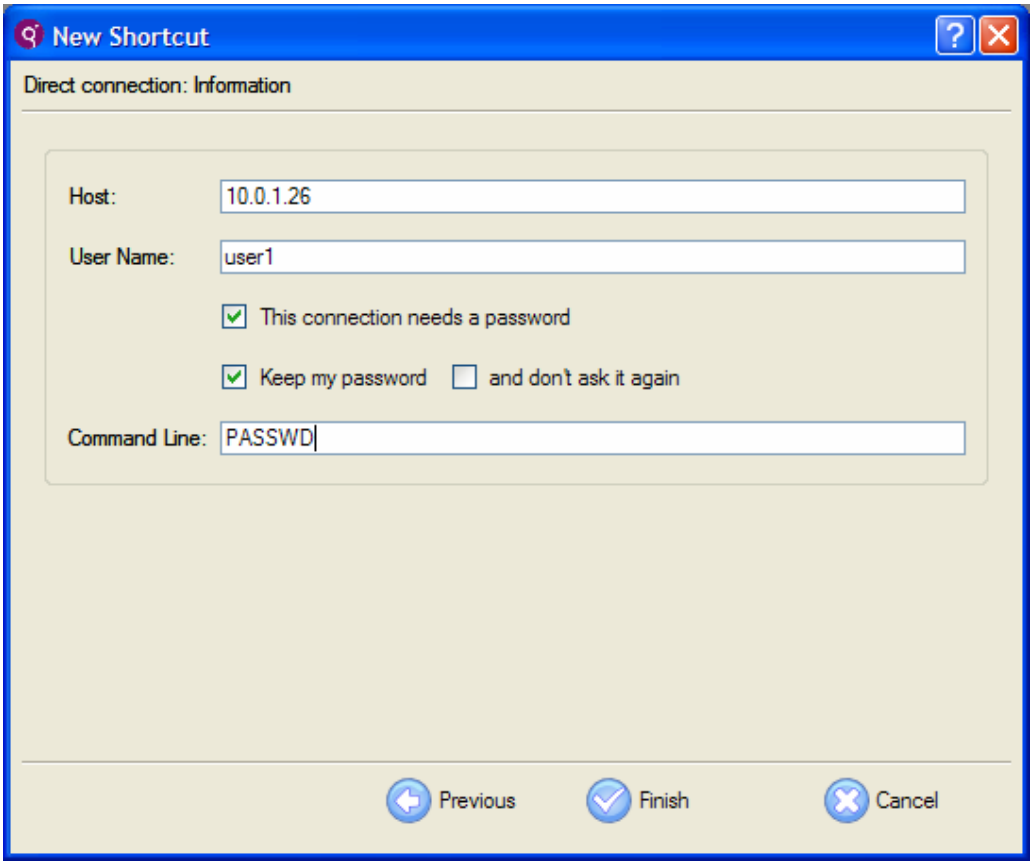



This rule looks for "Your password has expired" and open a text dialog window. Internally, the terminal window prompts for a new password from the server, as the existing password has expired. "Show the terminal" causes the Genero Desktop Client to display the server window, allowing the user to see the message and type in the correct passwords to complete the process. The window then closes and the user can click the shortcut once more and use the new password to start the application.

**Warning!** The string entered in the Received String field must match the string displayed by the system. It is case-sensitive, where "Password has expired" does not match "password has expired". The string for an expired password may be different than the example shown above, based on your system. You should verify the string for an expired password that is returned by your system prior to implementing this solution.

### Changing Passwords

Users may want to change their passwords prior to expiration. To allow for this functionality, provide a shortcut in the Genero Desktop Client that issues the password command. The sample login script uses a case statement that checks for PASSWD. The specifics of the shortcut are as follows:




---

## AUTOPORTFIND Source Code Example

This is the source code used to produce the port number for tunnelling with ssh. It should compile with little or no modification and does not need to be run as root.

```
Autoportfind.c/*
      Copyright(C) 2004, Four-J's Development Tools, all rights
reserved.
      Written by John A. Hobach, Dallas Texas, May 5th, 2004      The
purpose of the application is to return a port number that
      will not be used for awhile.  This port number can then be used
      by the Genero client for port forwarding.
      The operating system assigns ports in a round
```

```
      robin fashion so the port assigned is unlikely to be used again
      very soon.  This will give the GDC time to start ssh and use
      that port.  The OS will automatically skip ports in use.
      Revised 08/25/2004 Ver 2.1 to use bind() to get a port number
      assigned.  It is assigned a port automatically from the
      operating system and we immediatly get it and return it.
Revised 10/25/2005 Ver 2.2 to support returning a port number
      within a given range.  This is accomplished by requesting ports
      from the OS until it is within the range specified.

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>

#define USE_SOCKETS
#include "util.h"

char *progname;
static char *ver="autoportfind - Version 2.2, 2005-10-20, Four-J's
Development Tools, Inc";
static char *help=
"autoportfind [OPTION]\n"
"\n"
"Generate a port number for use with port forwarding.\n"
"\n"
"    -e, --env\n"
"           Send FJSPORTFORWARD=<port> to stdout.\n"
"\n"
"    -r     Cycle through port assignments to determine which ports\n"
"           the OS assigns to ports when originating connections.\n"
"    -u n   Upper limit. Request port numbers until one is returned\n"
"           below 'n'.\n"
"    -l n   Lower limit. Request port numbers until one is returned\n"
"           above 'n'.\n"
"    -h     Display this help message.\n"
"    -v     Display the version number.\n"
;

main(int argc, char **argv) {
      int sockfd, connected_socket, retval;
      int size, x, outofrange;
      int range_flag=0, env_flag=0;
      unsigned int      port, startport, highest,
                  lowest, cycle, direction,
                  llimit=0, ulimit=~0;
      int reuse_addr=1;
      char **arg;
      struct sockaddr_in serv_addr;

      progname=argv[0];
      arg=argv;
      while (--argc) {
```

```
            ++arg;
            if (!strcmp(*arg,"-r") || !strcmp(*arg,"--range")) {
                  range_flag=1;
            } else if (!strcmp(*arg,"-e") || !strcmp(*arg,"--env")) {
                  env_flag=1;
            } else if (!strcmp(*arg,"-u")) {
                  ++arg;
                  if (argc == 1 || *arg[0] == '-') {
                        fprintf(stderr,"%s: Value missing for -
u\n",progname);
                        exit(1);
                  }
                  --argc;
                  ulimit=atol(*arg);
            } else if (!strcmp(*arg,"-l")) {
                  ++arg;
                  if (argc == 1 || *arg[0] == '-') {
                        fprintf(stderr,"%s: Value missing for -
l\n",progname);
                        exit(1);
                  }
                  --argc;
                  llimit=atol(*arg);\
            } else if (!strcmp(*arg,"-v")) {
                  printf("%s\n",ver);
                  exit(0);
            } else if (!strcmp(*arg,"-h") || !strcmp(*arg,"--help")) {
                  printf("%s",help);
                  exit(0);
            } else {
                  fprintf(stderr,"%s:Unknown argument '%s'\n",
                    progname, *arg);
                  exit(1);
            }
      }

      lowest=~0;
      highest=0;
      startport=0;
      cycle=0;
      direction=1;

       do {
            outofrange=0;
            memset((char*) &serv_addr,0,sizeof(serv_addr));
            serv_addr.sin_family=AF_INET;
            serv_addr.sin_port=0;         /* allow system to assign */
            serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);


            sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
            if (sockfd < 0) {
                  perror("socket");
                  close(sockfd);
                  exit(1);
            }
```

```
            if (bind(sockfd, (struct sockaddr *) &serv_addr,
              sizeof(serv_addr)) < 0) {
                  perror("bind");
                  close(sockfd);
                  exit(1);
            }

            size=sizeof(serv_addr);
            if (getsockname(sockfd, (struct sockaddr *) &serv_addr,
              &size) == -1) {
                  perror("getsockname");
                  exit(errno);
            }

            if (range_flag) {
                  port=ntohs(serv_addr.sin_port);

                  if (!startport) startport=port;
                  if (port > highest) highest=port;
                  if (port < lowest) lowest=port;

                  if (direction==0 && port <= startport) {
                        cycle++;
                        direction=1;
                  } else if (direction==1 && port >= startport) {
                        cycle++;
                        direction=0;
                  }

            } else {
                  port=ntohs(serv_addr.sin_port);

                  if (port > llimit && port < ulimit) {
                        if (env_flag) printf("FJSPORTFORWARD=");
                        printf("%d\n",ntohs(serv_addr.sin_port));
                  } else
                        outofrange=1;
            }

            close(sockfd);

      } while ((range_flag && cycle < 3) || outofrange);

      if (range_flag)
            printf("Lowest port: %lu\nHighest port:
%lu\n",lowest,highest);

      exit(0);
}

---
Util.h
#ifndef UTIL_H
#define UTIL_H

#ifndef MAX
#  define MAX(a,b) a>b?a:b
```

```
#endif


#ifdef USE_SOCKETS
#  ifdef _WIN32
#    include <winsock.h>
#  else
#    include <sys/types.h>
#    include <sys/socket.h>
#    include <netinet/in.h>   /* struct sockaddr_in, ... */
#    include <netinet/tcp.h>  /* TCP_NODELAY, ... */
#    include <arpa/inet.h>    /* inet_addr, inet_ntoa, inet_aton */
#    include <netdb.h>        /* gethostbyname */
#  endif
#endif

#ifdef _WIN32
#  define SOCKLEN_T int
#endif

#ifdef __osf__
#  define SOCKLEN_T int
#endif

#ifdef _AIX
#  ifdef USE_SOCKETS
#   include <sys/ioctl.h>
#   include <sys/time.h>
#   include <sys/select.h>
#  endif
# define SOCKLEN_T socklen_t
#endif

#if defined  (M_I386)
/*  SCO */
#  ifdef USE_SOCKETS
#    include <sys/ioctl.h>
#    include <sys/time.h>
#    include <sys/select.h>
#  endif
#  define SOCKLEN_T int
#endif

#ifdef linux
#  define SOCKLEN_T socklen_t
#endif

#ifdef sun
#  if defined USE_SOCKETS
#    undef USE_SYS_SOCKIO
#    define USE_SYS_SOCKIO
#  endif
#  define SOCKLEN_T  int
#endif

#ifdef __hpux
#  define SOCKLEN_T int
```

```
#endif

#ifndef SOCKLEN_T
#  define SOCKLEN_T size_t
#endif

#ifndef MSG_DONTWAIT
#  define MSG_DONTWAIT 0
#endif

#endif
```

---

## Login Script

This is an example of the login script that is executed when users log in. It is intended to be an example, and we expect you to tailor it according to your needs. The login script is invoked via the /etc/passwd file.

```
#! /bin/sh
# Invoked directly by login mechanism such as telnetd, rlogind, or
sshd.
# This file is specified in the /etc/passwd file as being the shell.
This
# gives us the control we need for users that should never be allowed a
# shell prompt.
#
# Arguments passed are <COMMAND> <PORT> <FEID> <FEID2>
#
# <COMMAND> string must match the case statements below.

br

br
br
# set your env vars here
export FGLDIR=/fjs/f4gl/genero-training
export FGLRUN=fglrun
export FGLGUI=1
export LD_LIBRARY_PATH=/fjs/f4gl/genero-
1.20.1d/bin:/db/ifx/csdk270/lib:/db/ifx/csdk270/lib/c++:/db/ifx/csdk270
/lib/cli:/db/ifx/csdk270/lib/client:/db/ifx/csdk270/lib/dmi:/db/ifx/csd
k270/lib/esql::/db/ifx/ids921/lib
export PATH=${PATH}:$FGLDIR/bin
br
br
# The command line arguments passed from the GDC will be here. If there
# aren't any then we abort.


br
```

```
br

br

br

br
br
br

br

br

br

br

br

br
br
br
br
br
if [[ "$SSH_TTY" == "" && "$SSH_CONNECTION" == "" ]]

then
  # coming in from telnet or rlogin

br
br
 echo -n "$ " # fake shell prompt for GDC
 read APPLICATION FGLSERVER _FGLFEID _FGLFEID2


br
br
br
 if [[ "$APPLICATION" == "" ]]
 then
 echo "exiting due to bad arguments"
 sleep 5 # give time to view error because window will close
 exit 0
 fi

 export FGLSERVER
 export _FGLFEID
 export _FGLFEID2

 else

 # coming in from ssh and sshd

 if [[ "$1" == "" || "$1" != "-c" ]]
 then
```

```
 echo "exiting due to bad arguments"
 sleep 5 # give time to view error because window will close
  exit 0
 fi
 let pos=0
 for arg in $@
 do
 if [ $pos -eq 1 ]; then APPLICATION=$arg; fi
 if [ $pos -eq 2 ]; then export FGLSERVER="127.0.0.1:${arg}"; fi
 if [ $pos -eq 3 ]; then export _FGLFEID=$arg; fi
 if [ $pos -eq 4 ]; then export _FGLFEID2=$arg; fi

 #echo "$pos:'$arg'"
 let pos=pos+1
 done
fi

#echo "FGLSERVER=$FGLSERVER"

# Add case statements according to 1st value passed from the GDC
command line.
# Never execute the value passed directly as this would be a security
hole
# allowing the client to dictate what gets run.

case "$APPLICATION" in

 DEMO) cd $FGLDIR/demo
 $FGLDIR/bin/$FGLRUN demo
 ;;

# SHELL) /bin/sh # don't leave this in for production
# ;;

 AUTOPORT) /home/fjspf/autoportfind -e
 exit 0
 ;;

 PASSWD) /usr/bin/passwd
 exit 0
 ;;

 *) echo "Unknown application '$APPLICATION'"
 sleep 5 # allow time to read message
 ;;

esac
```

# Possible Configuration Problems

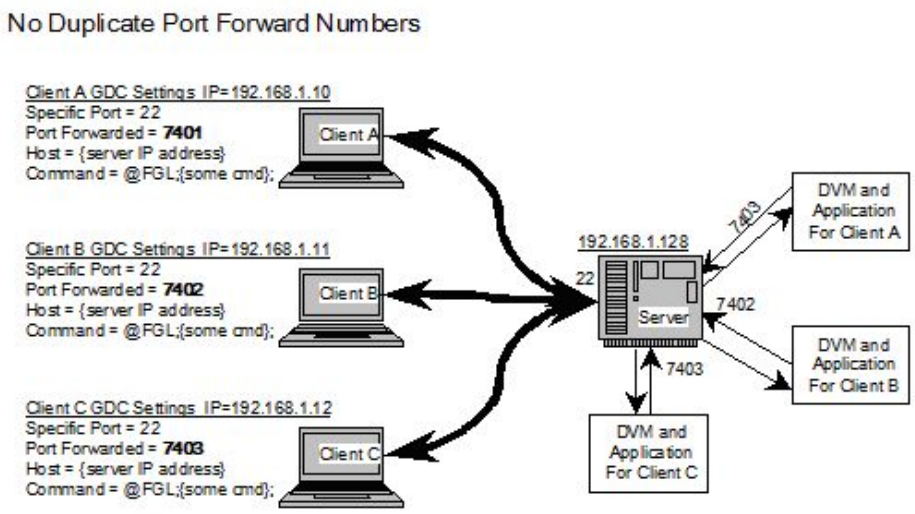
### Topics

- Duplicate port forward number
- Wireless Systems
- Need to change the port that GDC listens on
- Sessions Expiring

---

## Duplicate port forward number

If you have more than one client using the same Port Forward port number, the application display could go to the wrong client. This is an expected result, so precautions need to be taken to prevent this. Make sure that each client using port forwarding to a particular server uses a different port forward port number.




The sshd sets up a tunnel and listens on the port that is specified in the port forward field. This allows the Genero application (or DVM) to connect to that port and have the GUI data sent to the client. Only one listener can be listening on any port at a time. Each client needs to use a unique port number to avoid any problems.

**Note:** An automatic configuration solution is being worked on but has not been released at the time of this writing.

---

## Wireless Systems

The latest technology to use 802.11(a,b or g). This is great at avoiding the wire mess, but there is a new risk. Under Windows, if you are using a plugged in or built in wireless card, the interface goes offline if the signal is lost for even a second.. When this happens, it is treated similar to unplugging your network cable. The Windows drivers report to the network stack that the interface is now offline, and everything associated with that interface is removed. If an application has an open channel, it is signaled that it has closed. As a result, you lose all your connections and must wait for your signal to return in order to log in again.

A possible workaround is to use an external wireless device that doesn't take the connection down when the signal is lost. This works because it doesn't look like the cable was unplugged when it loses signal, so Windows doesn't know there is a problem. When the signal returns, everything works just at before.

---

## Need to change the port that GDC listens on

Why would you want to change the port that GDC listens on?

You may need to run several versions of the GDC on the same machine. Since each one must have its own listening port, Genero allows you to specify the port. If you run more than one and don't specify the port, Genero opens the next available port. For example, the first instance would open 6400, the next instance would open 6401.

```
>gdc                <- The port assigned would be 6400
>gdc -n             <- The port assigned would be 6401
>gdc -n -p 7400     <- The port assigned would be 7400
>gdc -n -p 7400     <- The port assigned would be 7401
```

Another reason to change ports might be that you can't use the ssh functionality. What if you haven't installed the SSH package yet, but you have more than one client behind the same firewall router? You can add rules to the router to send 6400 to the first client, 6410 to the second client, and so on. Each client would be started with the corresponding -p <port>, and the router would make sure each client gets the connections intended for it.

---

## Sessions Expiring

If you have sessions expire or applications that disappear, check for routers that expire sessions. Most likely, there is a firewall router in the path. If you are using a firewall

router, check for session expiration timers for the ports used to get through the firewall. The expiration duration (aka KeepAlive) should be set greater than the interval set in your operating system. This is set to 2 hours as a default on most computers. The operating systems can be tuned to have shorter values, but it is usually easier to adjust the router; use a value of 2 hours and 10 minutes.

---

# GDC and Windows XP Service Pack 2

### Topics

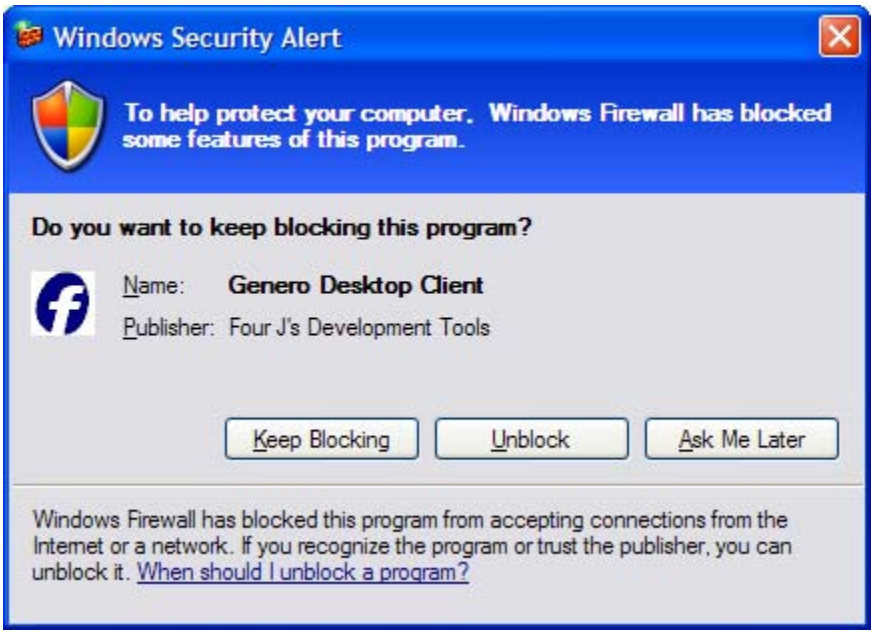
- Firewall configuration
- Active X

---

## Firewall configuration

Microsoft has added several security systems in Windows XP Service Pack 2 (SP2). The firewall included in Windows XP has been improved and is now enabled by default. From the network point of view, GDC is a server: it listens on a defined port (6400 by default) for Runtime System connections.

When GDC starts, the firewall detects it listens on port 6400 and warns the user:

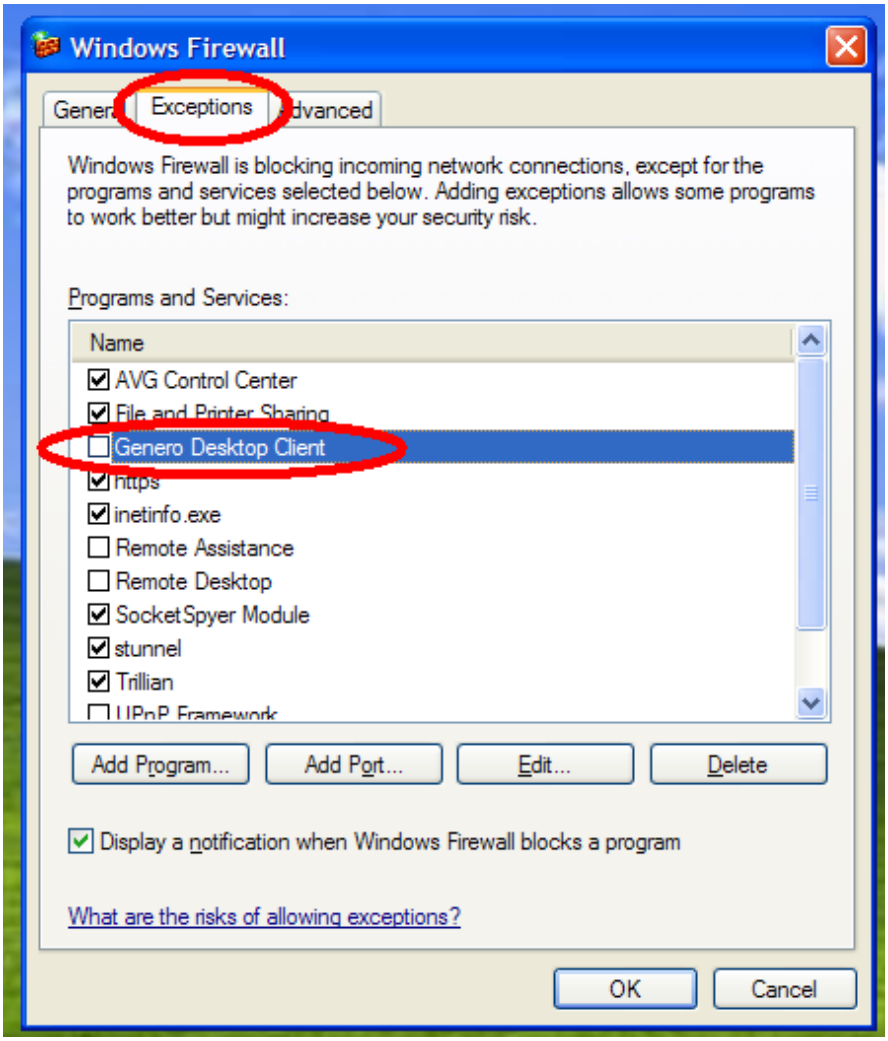



Press "Unblock" to allow the GDC running correcly.

**Warning!** pressing "Keep Blocking" or "Ask Me Later" will keep GDC from working. Connections from the Runtime System will be blocked by the firewall.

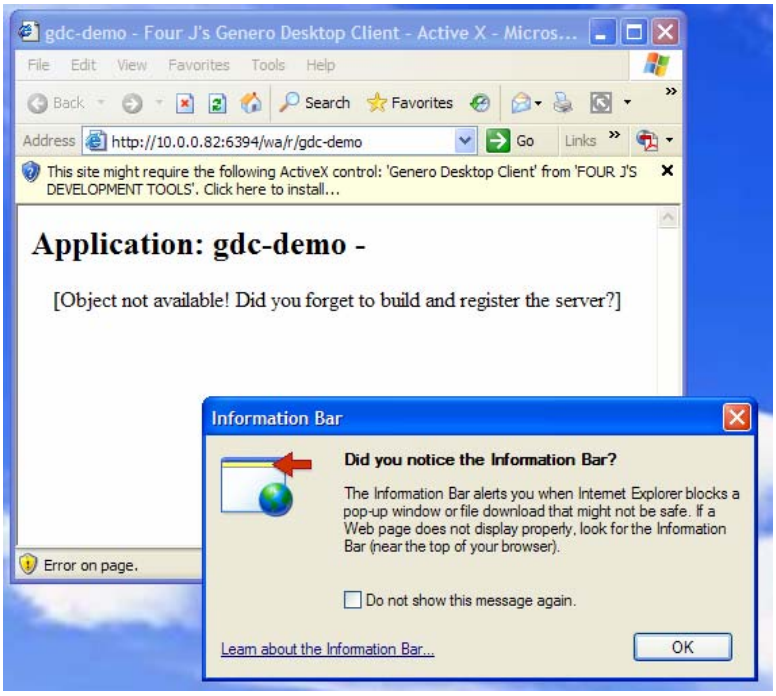
If "Keep Blocking" has been pressed by mistake, this parameter can be changed in the Firewall settings (Control Panel. In the "Exceptions" tab, Genero Desktop Client must be present and checked:


Control Panel
File Edit View Favorites Tools Help
Address Control Panel
Go
System
Taskbar and Start Menu
User Accounts
Windows Firewall
Wireless Link
Wireless Network Setup Wizard
Configure the Windows Firewall

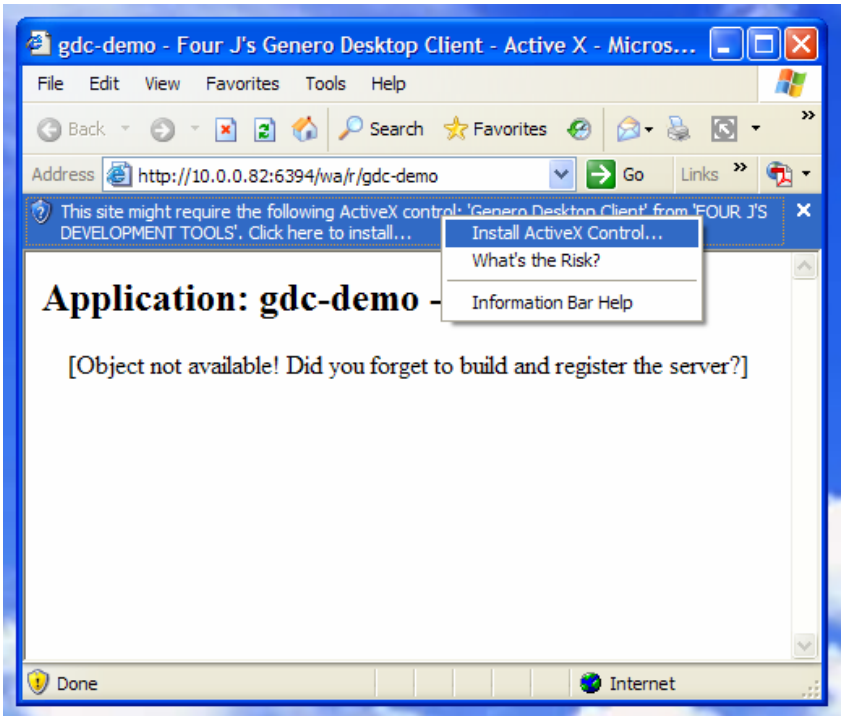

Windows Firewall
Exceptions
Windows Firewall is blocking incoming network connections, except for the programs and services selected below. Adding exceptions allows some programs to work better but might increase your security risk.
Programs and Services:
Name
AVG Control Center
File and Printer Sharing
Genero Desktop Client
https
inetinfo.exe
Remote Assistance
Remote Desktop
SocketSpyer Module
stunnel
Trillian
Add Program...
Add Port...
Edit...
Delete
Display a notification when Windows Firewall blocks a program
What are the risks of allowing exceptions?
OK
Cancel

## Active X

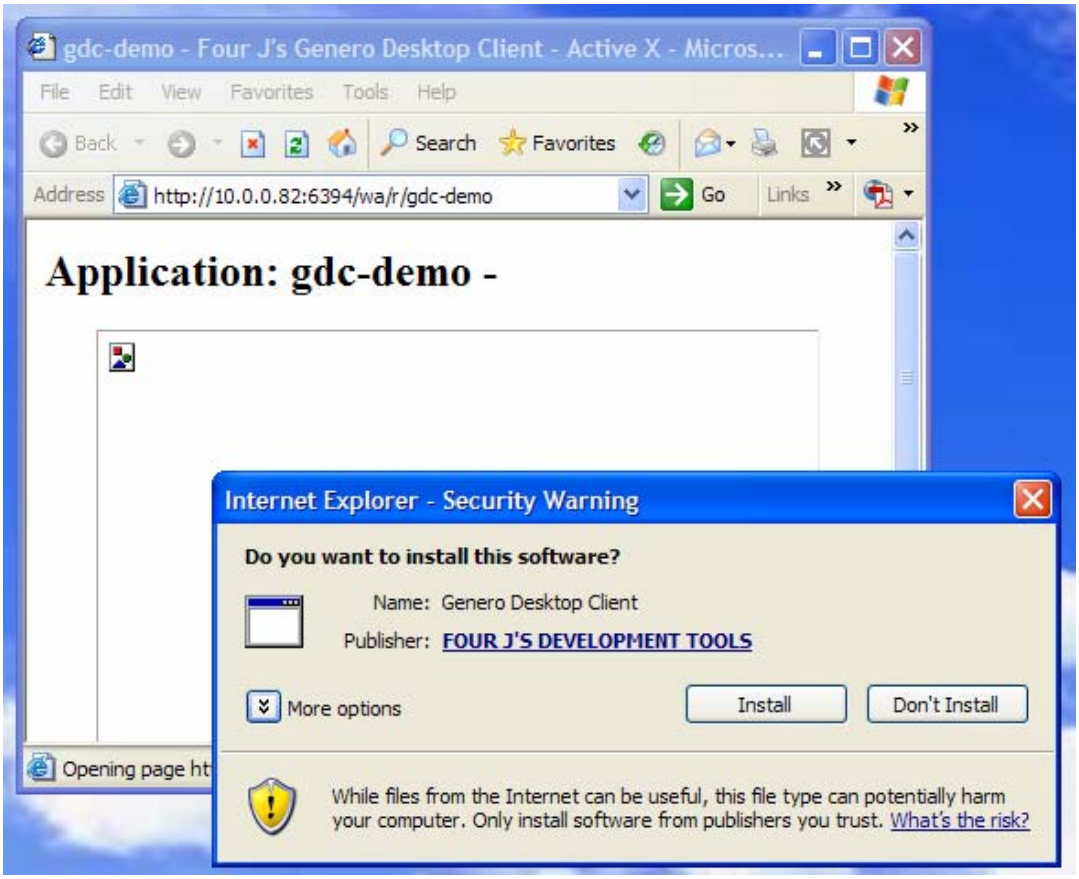
Depending on your configuration, the installation of the Active X may be blocked by Windows Security:




To install the ActiveX, you have to click on the "Information Bar" and choose "Install ActiveX Control.."



The page will be reloaded and then you will be prompted for installing GDC Active X




Then press "Install". The ActiveX will be installed and started. As ActiveX is also listening on port 6400 in the same way as the non activeX version, the warning described in the firewall configuration will show up.

# GDC and Windows Vista

### Topics



---

## User Account Control

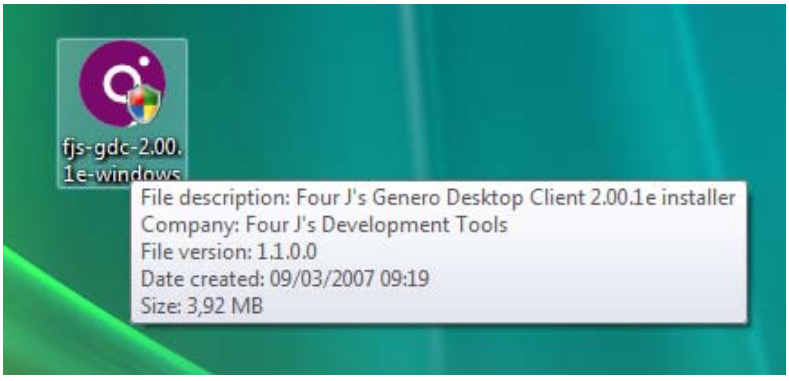
One of the new features of Window Vista is the User Account Control (UAC). UAC prevents any software from silently hurting your system by prompting the user before any administrative actions such as:

- installing a new program
- modification of the registry

It requires a user with Standard User rights (users not in the *Administrator* group) to provide an Administrator login and password when running a program that performs system-level tasks. Administrator Users will only have to confirm their actions. More details can be found at the Microsoft Web site.

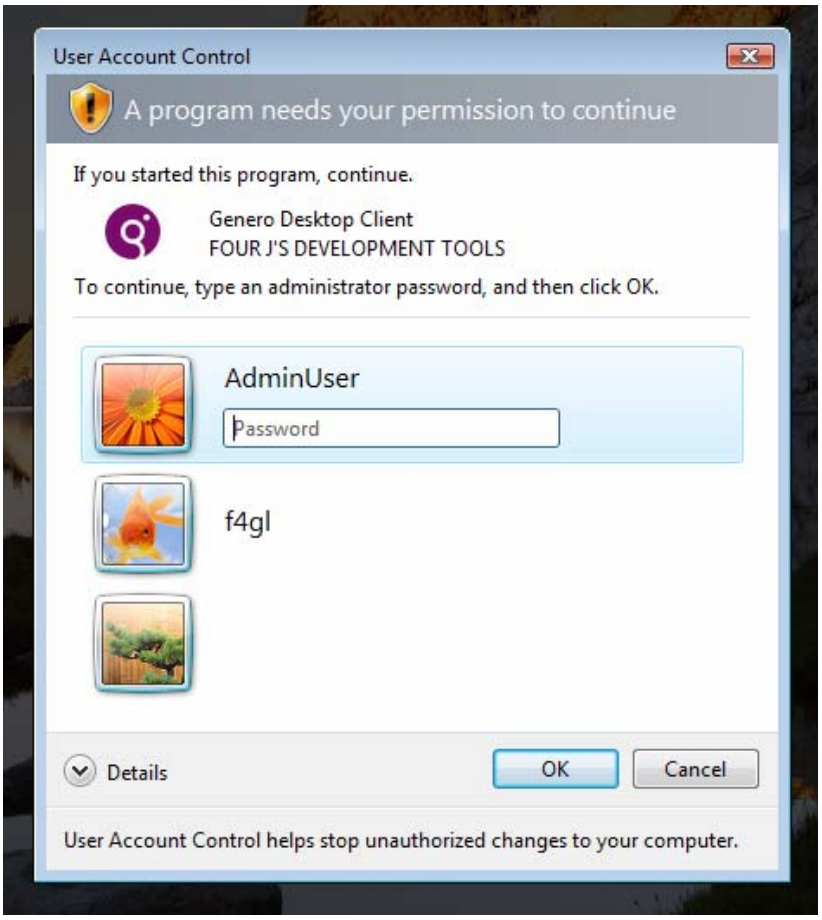
The User Account Control feature affects the Genero Desktop Client installation, as well as how GDC is run.

## Installation

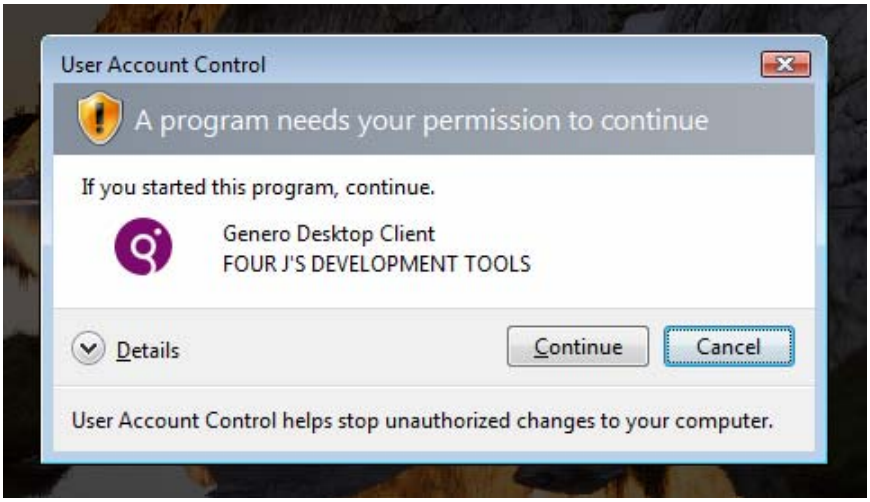



When the installation program starts, you'll be prompted to validate the installation.

**Standard User prompt:**



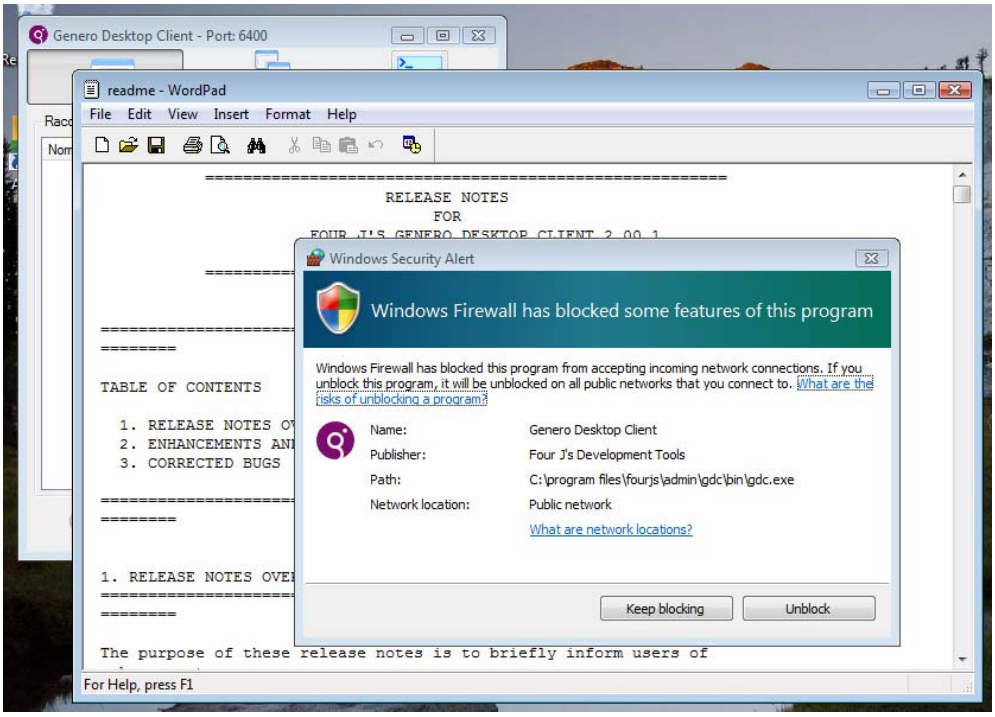

**Administrator User prompt:**



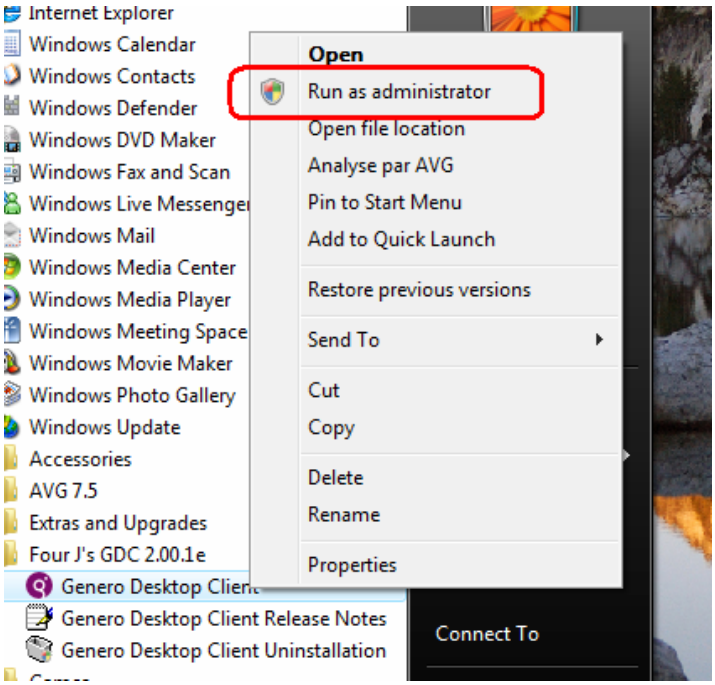

The installation then continues as in Windows XP.

## Running GDC

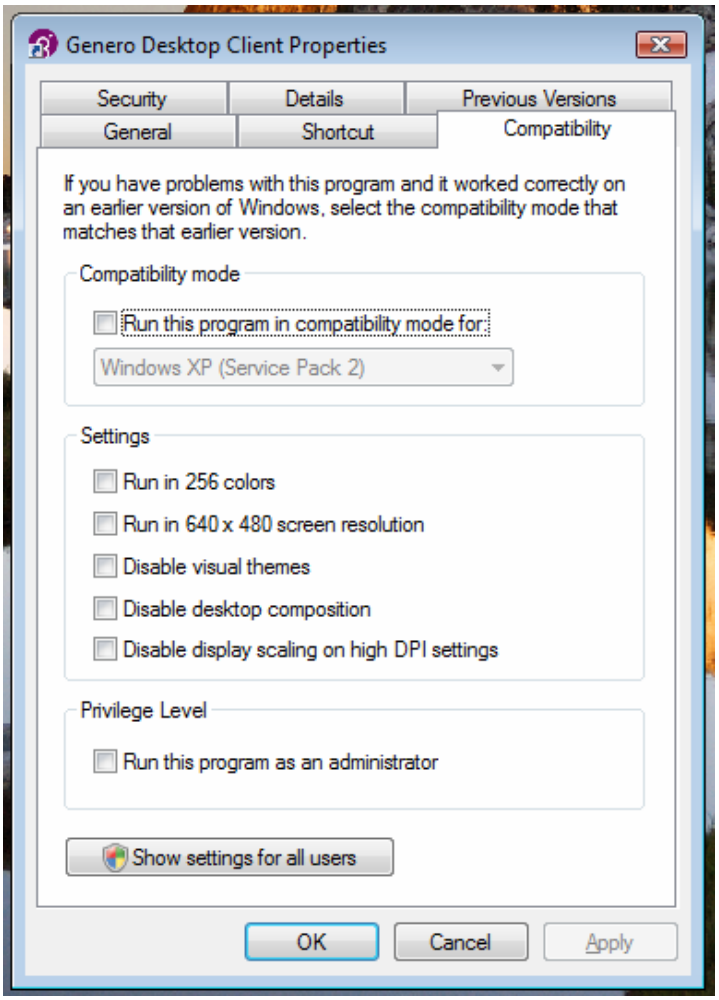
Once GDC is installed, the Windows Firewall will prompt the user to unblock the program, as in Windows XP SP2.




Although most of the features of Genero Desktop Client will work out of the box on Windows Vista, some features will only work if you start GDC "as administrator":



Even an Adminstrator User has to run the program "as administrator". However, Administrator users can create a shortcut and specify in the **Compatibility** tab that this program is always run as an administrator:




**Note:** Using the  -a GDC command line option to run GDC in "admin mode", a mode where you can manage GDC shortcuts, is not the same as asking Windows Vista to start GDC "as administrator", a special mode where the program can perform system-level tasks.

Some of the GDC features that are affected by the UAC are:

### Configuration

The folder %Program Files% is now protected with Windows Vista. So when starting GDC in normal mode, GDC will react as if the config file is read-only, and will only allows changes via the registry (local shortcuts). This may change in future versions of GDC.

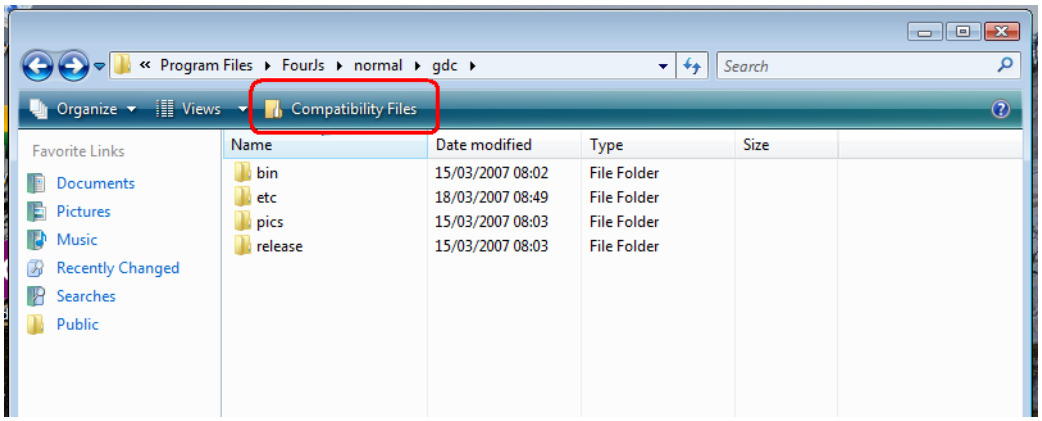
## File Transfer

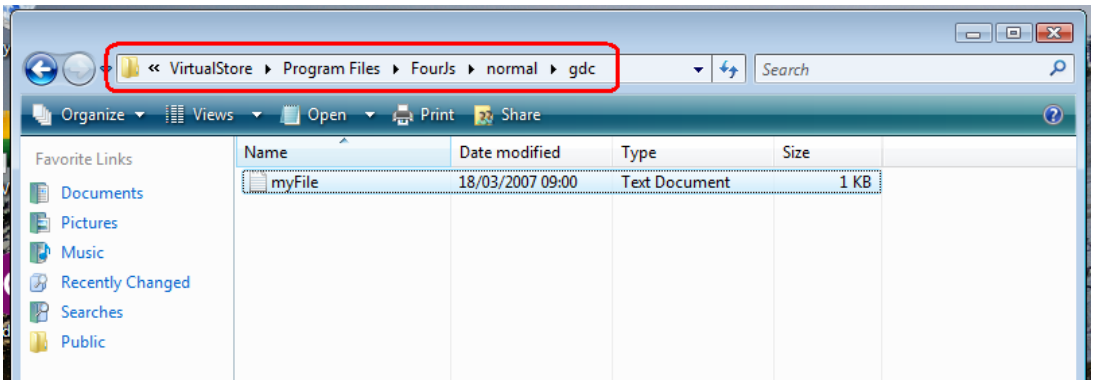
- The ProgramFiles folder

  As the folder %Program Files% is protected, Vista has introduced the concept of Virtual Store: instead of using the %Program Files% folder, programs will access a special directory. This means that if you use FGL_PUTFILE without a destination, the file will not be transferred to %Program Files%\Fourjs\GDC\ but to %VirtualStore%\Program Files\Fourjs\GDC.

  Example:

```
CALL FGL_PUTFILE("myFile.txt","myFile.txt")
```




- Uploading files

  Vista will prevent the upload of some files, such as executables or DLLs.

Example:

```
MAIN

  DISPLAY "upload text file..."
  CALL FGL_PUTFILE("myFile.txt","myFile.txt")
  DISPLAY "... done"
  DISPLAY "upload DLL..."
  CALL FGL_PUTFILE("myFile.dll","myFile.dll")
  DISPLAY "... done"

END MAIN
```

```
$fglrun ft
upload text file...
... done
upload DLL...
Program stopped at 'ft.4gl', line number 7.
FORMS statement error number -8066.
Could not write destination file for file transfer.
```

Running GDC as administrator will allow GDC to upload such files to the system.

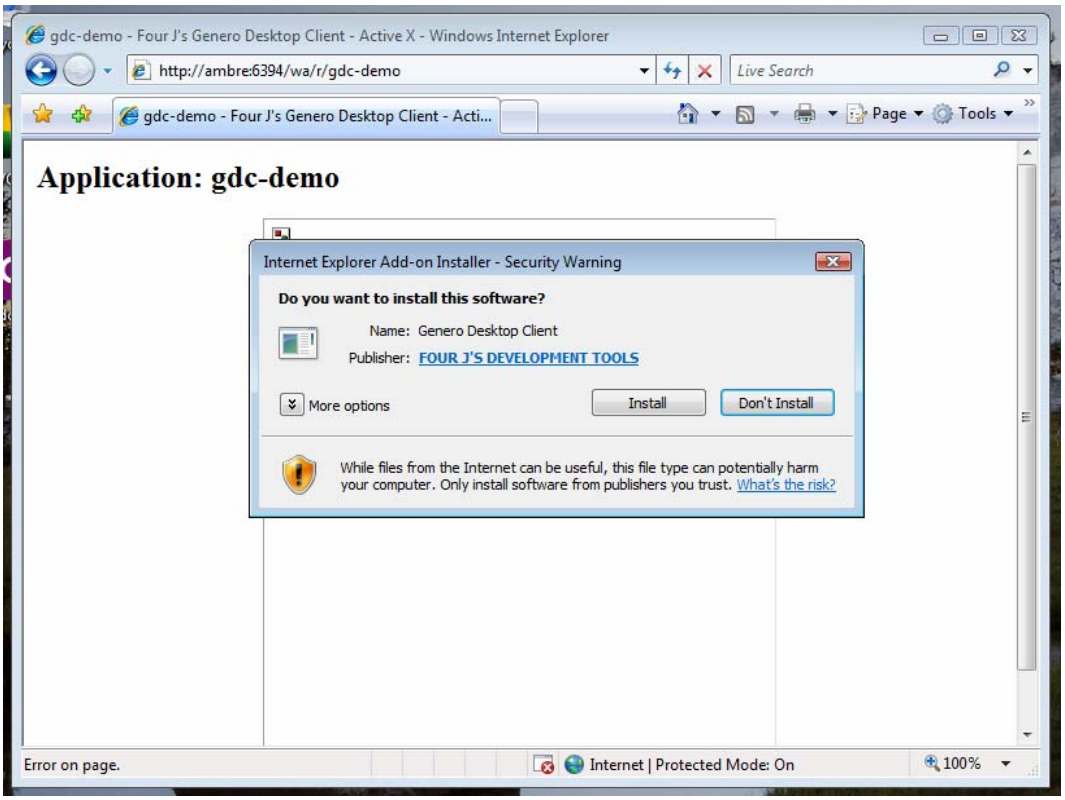
---

## Active X

### Installation

A lot of protection has been added to Vista to prevent spyware or other malicious programs from being installed without the agreement of the end user. Therefore, installing an Active X needs, by default, strong user agreement.

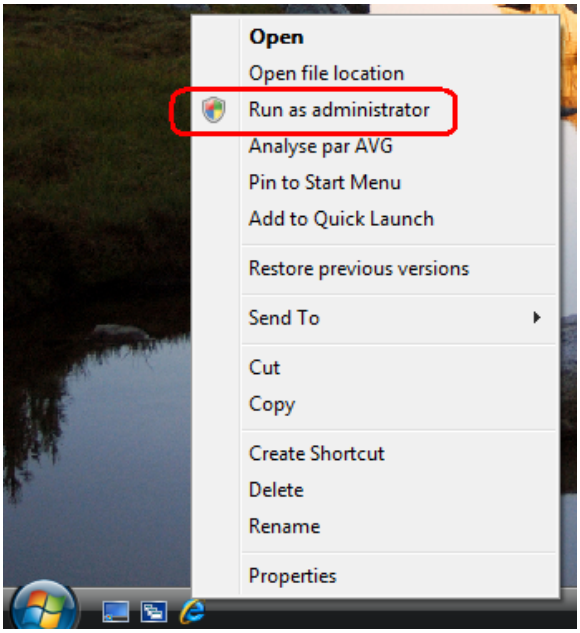
Active X are run by *Internet Add-on Installer*, a special tool used by Internet Explorer 7 to install Active X. Running this tool requires the agreement of the end-user and UAC Administrator User rights:



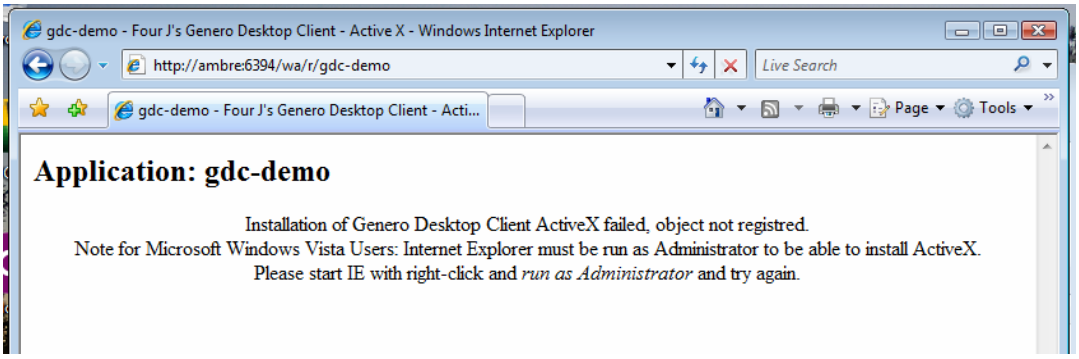

The first time you start GDC Active X, two steps are performed:

- Deploying files (simple copy) in %Program Files%
- Registering the Active X (which is done by gdc.exe /regserver), to tell Internet Explorer how to execute the ActiveX.

The first step is done by *Internet Explorer Add-on Installer*, but the second requires IE to be started *as Administrator*:



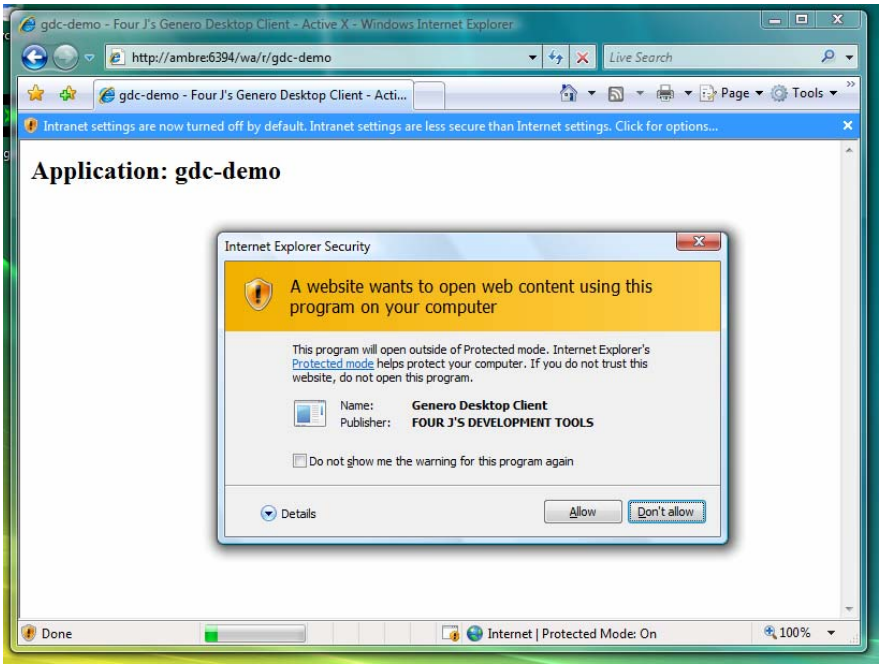

If not, GDC will not be registered and Internet Explorer will not be able to load GDC Active X:



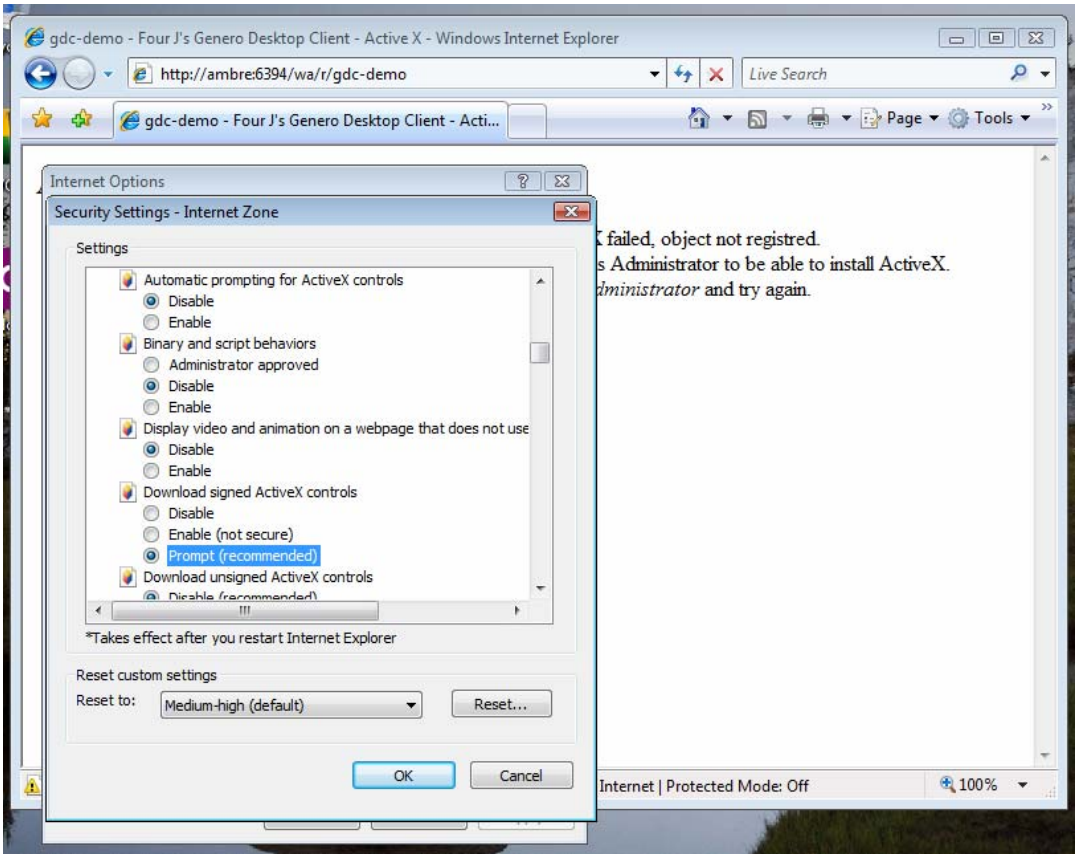

**Note:** This message has been introduced in the template in GDC 2.00.1e or greater.

### Running Active X

It is not necessary to run IE as Administrator in order to use GDC Active X, only to install it. Once the Active X has been installed, the user will be prompted to allow Internet Explorer to start Genero Desktop Client Active X:




Depending on your Security Settings, Internet Explorer may not be able to start ActiveX ; the default security settings have been changed to allow less freedom to Active X for Vista. You may have to change these settings to allow Genero Desktop Client Active X to be used on your computer.



**Note:** The gdc.exe file has been signed in version 2.00.1e or greater.

---

# Front End Extensions

### Topics

- Overview
- How does it work ?
- Initialize and Finalize
- Front End Extension function
- Front End Interface
- Environment Variables
- Example: basic extension

---

## Overview

The Front End allows you to call external functions. Theses functions are dynamically loaded by the front end when needed; they are within a DLL (Dynamic Linked Library under Windows systems), so (Shared Object under Linux), or dyLib (Dynamic Library under Mac Os X).

This allows you to create your own extensions and use them from your 4GL code. The parameters are sent to the front end, and a stack is used to transmit in and out parameters to the extension.

---

## How does it work ?

This small tutorial, "how-to create your own extension", explains the mechanism of Front End Extensions.

#### STEP 1: the 4GL code.

Front end extensions can be called using the `ui.Interface.frontCall()` built-in function. See the Runtime System documentation for more information about this function.

```
call ui.Interface.frontCall( module, function, <in-list>, <out-
list>)
```

For example, if the extension is called '`myExt`', the function used is `makeSum`, and the function takes 2 parameters in and returns an integer and a string:

```
DEFINE a,b INTEGER -- the two IN parameters
DEFINE c INTEGER -- the integer returned
```

```
DEFINE res STRING -- the string returned

call ui.Interface.frontCall( "myExt", "makeSum", [a,b], [c,res])
```

### STEP 2: Internal GDC mechanism - the stack.

To transmit parameters to the extension, GDC uses a stack ; `a` and `b` will then be pushed on that stack.

### STEP 3: Call the external function.

All the information needed by the front end is transmitted to the extension using a `front end interface structure`. This structure contains a list of function pointers to :

- manage the stack (push or pop for each handled data type)
- get information on the function (number of IN or OUT parameters)
- get information about the front end

For this reason the prototype of each function should be the same.

### STEP 4: the extension function is running.

The classic process is:

1. Check that the number of parameters is correct
2. Retrieve the parameters using the stack functions
3. Do what the function has to do
4. Use the stack functions to stack the return values
5. Return 0 in case of success

### STEP 5: Internal GDC mechanism - the stack.

The GDC uses the stack to un-stack the returned values and transmit them to the Runtime System, which dispatches the values into the right variable

---

## Initialize and Finalize

The Front End will automatically perform two functions when using a Dynamic Library:

`void initialize();` which is called the first time the library has been loaded, and `void finalize();`, which is called when the front end stops. These two functions allow you to have better control of your extension.

## Front End Extension Function

A Front End Extension Function must have the following prototype:

```
int <name> ( const struct frontEndInterface &<fx>);
```

**Notes:**

1. `<name>` is the name of your function.
2. `<fx>` is the structure for the frontEndInterface
3. This function return 0 if it is successful, -1 if not.

Parameters are transmitted via a stack. This stack can be managed using the functions provided by the frontEndInterface structure. `getParamCount();` and `getReturnCount();` can be used to check if the number of parameters is correct.

## Front End Interface structure

The Front End will dynamically load the library and call the given function. IN and OUT parameters are managed using a stack. All the operations concerning this stack are managed by functions within the Front End. Pointers to these functions are gathered into a front end interface structure:

```
struct frontEndInterface
{
    short (* getParamCount) ();
    short (* getReturnCount) ();
    void ( * popInteger) (long & , short & );
    void ( * pushInteger) (const long , short );
    void ( * popString) (char *, short &, short &);
    void ( * pushString) (const char *,short , short );
    void ( * getFrontEndEnv) (const char * , char *, short & );
    /*new in 2.00*/
    void ( * popWString) (wchar_t *, short &, short &);
    void ( * pushWString) (const wchar_t*,short , short );
};
```

| Function | Description |
|---|---|
| `short getParamCount();` | This function returns the number of parameters given to the function called. |

| | |
|---|---|
| `short getReturnCount();` | This function returns the number of returning values of the function called. |
| `void popInteger( long & value, short & isNull );` | This function is used to get an integer from the stack. `value` is the reference to where the popped integer will be set; `isNull` indicates whether the parameter is null. |
| `void pushInteger( const long value, short isNull );` | This function is used to push an integer on the stack; `value` is the value of the integer, `isNull` indicates whether the value is null. |
| `void popString( char * value, short & length, short & isNull );` | This function is used to get a string from the stack; `value` is the pointer where the popped string will be set, `length` is the length of the string, `isNull` indicates whether the parameter is null. |
| `void pushString( const char * value, short length, short isNull );` | This function is used to push a string on the stack; `value` is the value of the string, `length` the length of the string, `isNull` indicates whether the parameter is null. A length of -1 indicates that the length is detected based on the content of the string. |
| `void ( * getFrontEndEnv )(const char * , char *, short & );` | This function is used to get information from the front end. The table in Environment Variables indicates which "front end environment variable" is set. |
| `void ( * popWString) (wchar_t *value, short &length, short &isNull);` | This function is used to get a unicode string from the stack; `value` is the pointer where the popped string will be set, `length` is the length of the string, `isNull` indicates whether the parameter is null. |
| `void ( * pushWString) (wchar_t *value, short length, short isNull);` | This function is used to push a unicode string on the stack;. `value` is the value of the string, `length` the length of the string, `isNull` indicates whether the parameter is null. A length of -1 indicates that the length is detected based on the content of the string. |

---

## Environment Variables

| <> Name | Meaning |
|---|---|
| `frontEndPath` | The path where the front end is installed |

## Example: basic extension

This very basic extension illustrates how to create a DLL. It has been created using Visual C++ 6.

This DLL has only one function, called makeSum. Two Integers are given in parameters, and the function returns a String and an Integer. The Integer contains the sum, the String a small text.

**Header file : myDLL.h**

```
/*the interface structure*/
struct frontEndInterface {
short (* getParamCount) ();
short (* getReturnCount) ();
void ( * popInteger) (long & , short & );
void ( * pushInteger) (const long , short );
void ( * popString) (char *, short &, short &);
void ( * pushString) (const char *,short , short );
void ( * getFrontEndEnv(const char *, char *, short&);
void ( * popWString) (wchar_t *, short &, short &);
void ( * pushWString) (const wchar_t*,short , short );
};

// a small macro used to declare the "exportable" functions
#ifdef WIN32
/*dllexport is only valid (and mandatory) under windows*/
#define EXPORT extern "C" __declspec(dllexport)
#else
#define EXPORT extern "C"
#endif

// the functions we want to be available from the front end
EXPORT void initialize();
EXPORT void finalize();
EXPORT int makeSum(const frontEndInterface &fx);
```

**Source file : myDLL.cpp**

```
// some includes
#include "myDLL.h"
#include <stdio.h>

// this function will be called by the Front End the first time
the DLL is loaded
void initialize() {
}

// this function will be called by the Front End when the Front
End stops
void finalize() {
}
```

```
// our makeSum function.
// --> in parameters the front End Interface Structure
int makeSum(const struct frontEndInterface &fx) {
  // initialize the status
   short status = -1;

  // check if the in and out parameters are correct
   if (fx.getParamCount() == 2 && fx.getReturnCount() == 2) {
     long param1, param2;
     short isNull1, isNull2;

     // get from the stack each parameter
    fx.popInteger(param2, isNull2);
    fx.popInteger(param1, isNull1);

    // check if they are not null
    if (!isNull1 && !isNull2) {

      // create the answer
      long sum = param1 + param2;
      char msg[255];
      sprintf(msg, "%d + %d = %d", param1, param2, sum);

      // push the answer on the stack
      fx.pushInteger(sum, 0);
      fx.pushString(msg, strlen(msg), 0);

    // successful -> status = 0
      status = 0;
     }
  }
  return status;
}
```

Running the program gives the following ouput:

```
$fglrun testDLL
1 + 3 = 4
4
```

# Windows DDE Support

### Topics

- What is DDE?
- Using DDE API
- The DDE API
- Example

---

## What is DDE?

DDE is a form of inter-process communication implemented by Microsoft for Windows platforms. DDE uses shared memory to exchange data between applications. Applications can use DDE for one-time data transfers and for ongoing exchanges in applications that send updates to one another as new data becomes available.

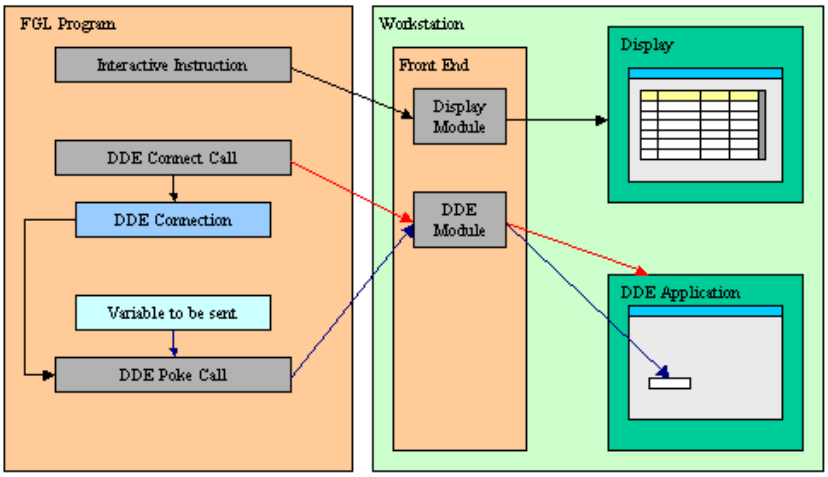
Please refer to your Microsoft documentation for DDE compatibility between existing versions. As an example, DDE commands were changed between Office 97 and Office 98.

We provide a DDE interface as a Front-End Extension: WinDDE.DLL

---

## Using DDE API

With DDE Support, you can invoke a Windows application and send or receive data to/from it. To use this functionality, the program must use the Windows Front End. Before using the DDE functions, the TCP communication channel between the application and the front end must be established with OPEN WINDOW, MENU, or DISPLAY TO.



The DDE API is used in a four-part procedure, as described in the following steps:

1. The application sends to the Front End the DDE order using the TCP/IP channel.
2. The Front End executes the DDE order and sends the data to the Windows application through the DDE API.
3. The Windows application executes the command and sends the result, which can be data or an error code, to the Front End.
4. The Windows Front End sends back the result to the application using the TCP/IP channel.

A DDE connection is uniquely identified by two values: The name of the DDE Application and the document. Most DDE functions require these two values to identify the DDE source or target.

---

## The DDE API

The DDE API is based on the front call technique as described in Front End Functions. All DDE functions are grouped in the **WINDDE** front end function module.

| Function name | Description |
|---|---|
| DDEConnect | This function opens a DDE connection |
| DDEExecute | This function executes a command in the specified program |
| DDEFinish | This function closes a DDE connection |
| DDEFinishAll | This function closes all DDE connections, as well as the DDE server program |
| DDEError | This function returns DDE error information about the last DDE operation |
| DDEPeek | This function retrieves data from the specified program and document using the DDE channel |
| DDEPoke | This function sends data to the specified program and document using the DDE channel |

---

## DDEConnect

### Purpose:

This function opens a DDE connection.

### Syntax:

```
CALL ui.Interface.frontCall("WINDDE","DDEConnect",
        [ program, document ], [result] )
```

### Notes:

1. *program* is the name of the DDE application.
2. *document* is the document that is to be opened.
3. *result* is an integer variable receiving the status.
4. *result* is TRUE if the function succeeded, FALSE otherwise.
5. If the function failed, use DDEError to get the description of the error.

### Warnings:

1. If the function failed with "DMLERR_NO_CONV_ESTABLISHED", then the DDE application was probably not running. Use the execute or shellexec front call to start the DDE application.

---

### DDEExecute

### Purpose:

This function executes a DDE command.

### Syntax:

```
CALL ui.Interface.frontCall("WINDDE","DDEExecute",
        [ program, document, command ], [result] )
```

### Notes:

1. *program* is the name of the DDE application.
2. *document* is the document that is to be used.
3. *command* is the command that needs to be executed.
4. Refer to the *program* documentation to know the syntax of *command*.
5. *result* is an integer variable receiving the status.
6. *result* is TRUE if the function succeeded, FALSE otherwise.
7. If the function failed, use DDEError to get the description of the error.

### Warnings:

1. The DDE connection must be opened see DDEConnect.

### DDEFinish

#### Purpose:

This function closes a DDE connection.

#### Syntax:

```
CALL ui.Interface.frontCall("WINDDE","DDEFinish",
        [ program, document ], [result] )
```

#### Notes:

1. *program* is the name of the DDE application.
2. *document* is the document that is to be closed.
3. *result* is an integer variable receiving the status.
4. *result* is TRUE if the function succeeded, FALSE otherwise.
5. If the function failed, use DDEError to get the description of the error.

#### Warnings:

1. The DDE connection must be opened, see DDEConnect.

---

### DDEFinishAll

#### Purpose:

This function closes all DDE connections, as well as the DDE server program.

#### Syntax

```
CALL ui.Interface.frontCall("WINDDE","DDEFinishAll", [], [result] )
```

#### Notes:

1. Closes all DDE connections.
2. *result* is an integer variable receiving the status.
3. *result* is TRUE if the function succeeded, FALSE otherwise.

---

### DDEError

### Purpose:

This function returns the error information about the last DDE operation.

### Syntax:

```
CALL ui.Interface.frontCall("WINDDE","DDEError", [], [errmsg] )
```

### Notes:

1. *errmsg* is the error message. It is set to NULL if no error occurred.

---

### DDEPeek

### Purpose:

This function retrieves data from the specified program and document using the DDE channel.

### Syntax:

```
CALL ui.Interface.frontCall("WINDDE","DDEPeek",
        [ program, container, cells ], [ result, value ] )
```

### Notes:

1. *program* is the name of the DDE application.
2. *container* is the document or sub-document that is to be used.
   A sub-document can, for example, be a sheet in Microsoft Excel.
3. *cells* represents the working items; see the *program* documentation to know the format of *cells*.
4. *value* represents the data to be retrieved; see the *program* documentation to know the format of *values*.
5. *result* is an integer variable receiving the status.
6. *result* is TRUE if the function succeeded, FALSE otherwise.
7. If the function failed, use DDEError to get the description of the error.
8. *value* is a variable receiving the cells values.

### Warnings:

1. The DDE connection must be opened; see DDEConnect.
2. DDEError can only be called once to check if an error occurred.

### DDEPoke

### Purpose:

This function sends data to the specified program and document using the DDE channel.

### Syntax:

```
CALL ui.Interface.frontCall("WINDDE","DDEPoke",
        [ program, container, cells, values ], [result] )
```

### Notes:

1. *program* is the name of the DDE application.
2. *container* is the document or sub-document that is to be used.
   A sub-document can, for example, be a sheet in Microsoft Excel.
3. *cells* represents the working items; see the *program* documentation to know the format of *cells*.
4. *values* represents the data to be sent; see the *program* documentation to know the format of *values*.
5. *result* is an integer variable receiving the status.
6. *result* is TRUE if the function succeeded, FALSE otherwise.
7. If the function failed, use DDEError to get the description of the error.

### Warnings:

1. The DDE connection must be opened; see DDEConnect.
2. An error may occur if you try to set many (thousands of) cells in a single operation.

---

## Example

### dde_example.per

```
01 DATABASE formonly
02 SCREEN
03 {
04 Value to be given to top-left corner :
05 [f00                                 ]
06 Value found on top-left corner :
07 [f01                                 ]
08 }
09 ATTRIBUTES
```

```
10   f00 = formonly.val;
11   f01 = formonly.rval, NOENTRY;
```

### dde_example.4gl

```
01 MAIN
02   -- Excel must be open with "File1.xls"
03   CONSTANT file = "File1.xls"
04   CONSTANT prog = "EXCEL"
05   DEFINE val, rval STRING
06   DEFINE res INTEGER
07   OPEN WINDOW w1 AT 1,1 WITH FORM "dde_example.per"
08   INPUT BY NAME val
09   CALL ui.Interface.frontCall("WINDDE","DDEConnect", [prog,file],
[res] )
10   CALL checkError(res)
11   CALL ui.Interface.frontCall("WINDDE","DDEPoke",
[prog,file,"R1C1",val], [res] );
12   CALL checkError(res)
13   CALL ui.Interface.frontCall("WINDDE","DDEPeek",
[prog,file,"R1C1"], [res,rval] );
14   CALL checkError(res)
15   DISPLAY BY NAME rval
16   INPUT BY NAME val WITHOUT DEFAULTS
17   CALL ui.Interface.frontCall("WINDDE","DDEExecute",
[prog,file,"[save]"], [res] );
18   CALL checkError(res)
19   CALL ui.Interface.frontCall("WINDDE","DDEFinish", [prog,file],
[res] );
20   CALL checkError(res)
21   CALL ui.Interface.frontCall("WINDDE","DDEFinishAll", [], [res] );
22   CALL checkError(res)
23   CLOSE WINDOW w1
24 END MAIN
25
26 FUNCTION checkError(res)
27   DEFINE res INTEGER
28   DEFINE mess STRING
29   IF res THEN RETURN END IF
30   DISPLAY "DDE Error:"
31   CALL ui.Interface.frontCall("WINDDE","DDEError",[],[mess]);
32   DISPLAY mess
33   CALL ui.Interface.frontCall("WINDDE","DDEFinishAll", [], [res] );
34   DISPLAY "Exit with DDE Error."
35   EXIT PROGRAM (-1)
36 END FUNCTION
```

# Windows COM Support

**Topics**



---

## What is COM?

COM stands for Component Object Model. It allows anyone to directly access Windows Applications Objects. You can create instances of those objects, call methods on them, and get or set their properties.

---

## Using the COM API

With COM Support, you can invoke a Windows application and send or receive data to or from it. To use this functionality, the program must use the Windows Front End.

---

## The COM API

The COM API is based on the front call technique as described in Front End Functions. All COM functions are grouped in the **WinCOM** front end function module.

| Function name | Description |
|---|---|
| CreateInstance | This function creates an instance of a registered COM object |
| CallMethod | This function calls a method on a specified object |
| GetProperty | This function gets a property of an object |
| SetProperty | This function sets a property of an object |
| GetError | This function gets a description of the last error which occurred |
| ReleaseInstance | This function releases an Instance (also frees memory) |

### Supported syntax

COM language syntaxe is very flexible and allows lots of notation. Genero WinCOM API is slightly more strict:

- `:=` notation **is** only **allowed** in version 2.00.1e (or later) ; for instance: `myFunction(SourceType:=3)`
- "no parenthesis" notation **is not allowed** ; for instance: `myFunction 3` must be written `myFunction(3)`
- numeric constants **are** only **allowed** in version 2.00.1e (or later).
  The constant list depends on the application used via WinCOM ; therefore, the list is configurable: a file named etc/WinCOM.cst gathers all the constants provided today by Microsoft for Office XP. It can be modified to add user-defined constants.

---

### CreateInstance

#### Purpose:

This function creates an instance of a registered COM object.

#### Syntax:

```
CALL ui.Interface.frontCall("WinCOM","CreateInstance",
        [  classname ], [handle] )
```

#### Notes:

1. *program* is the classname of the registered COM object.
2. *handle* is an integer variable receiving the status.
3. *handle* is -1 if there as an error, otherwise an integer value that can be used for a later call to the API.
4. If the function failed, use GetError to get the description of the error.

---

### CallMethod

#### Purpose:

This function calls a method on a specified object.

### Syntax:

```
CALL ui.Interface.frontCall("WINCOM","CallMethod",
        [ handle, method, arg1, ... ], [result] )
```

### Notes:

1. *handle* is the handle returned by another frontcall (CreateInstance, CallMethod, GetProperty).
2. *method* is the member name to call.
3. *arg1 (and ...)* are the arguments to pass to the method call.
4. *result* is either a handle or a value of a predefined type.
5. *result* is -1 in case of error (use GetError to get the description of the error).

---

## GetProperty

### Purpose:

This function gets a property of an object.

### Syntax:

```
CALL ui.Interface.frontCall("WINCOM","GetProperty",
        [ handle, member ], [result] )
```

### Notes:

1. *handle* is the handle returned by another frontcall (CreateInstance, CallMethod, GetProperty).
2. *member* is the member property name to get.
3. *result* is either a handle or a value of a predefined type.
4. *result* is -1 in case of error (use GetError to get the description of the error).

---

## SetProperty

### Purpose:

This function sets a property of an object.

### Syntax

```
CALL ui.Interface.frontCall("WINCOM","SetProperty", [handle, member,
value], [result] )
```

### Notes:

1. *handle* is the handle returned by another frontcall (CreateInstance, CallMethod, GetProperty).
2. *member* is the member property name to set.
3. *value* is the value to which the property will be set.
4. *result* is -1 in case of error (use GetError to get the description of the error), otherwise it is 0.

---

### GetError

### Purpose:

This function gets a description of the last error which occurred.

### Syntax

```
CALL ui.Interface.frontCall("WINCOM","GetError", [], [result] )
```

### Notes:

1. *result* is the description of the last error.
2. the returned value is NULL if there was no error.

---

### ReleaseInstance

### Purpose:

This function releases an Instance of a COM object.

### Syntax:

```
CALL ui.Interface.frontCall("WINCOM","ReleaseInstance", [handle],
[result] )
```

**Notes:**

1. *handle* is the handle returned by another frontcall (CreateInstance, CallMethod, GetProperty).
2. *result* is -1 in case of error (use GetError to get the description of the error), otherwise it is 0.

---

## Example

The following example puts "foo" in the first row of the 1st column of an Excel Sheet.

### com_example.4gl

```
01 DEFINE xlapp INTEGER
02 DEFINE xlwb INTEGER
03
04 MAIN
05   DEFINE result INTEGER
06   DEFINE str STRING
07
08   --initialization of global variables
09   LET xlapp = -1
10   LET xlwb = -1
11
12   --first, we must create an Instance of an Excel Application
13   CALL ui.Interface.frontCall("WinCOM", "CreateInstance",
["Excel.Application"], [xlapp])
14   CALL CheckError(xlapp, __LINE__)
15   --then adding a Workbook to the current document
16   CALL ui.interface.frontCall("WinCOM", "CallMethod", [xlapp,
"WorkBooks.Add"], [xlwb])
17   CALL CheckError(xlwb, __LINE__)
18   --then, setting it to be visible
19   CALL ui.interface.frontCall("WinCOM", "SetProperty", [xlapp,
"Visible", true], [result])
20   CALL CheckError(result, __LINE__)
21   --then CALL SetProperty to set the value of the cell
22   CALL ui.Interface.frontCall("WinCOM", "SetProperty", [xlwb,
"activesheet.Range(A1).Value", "foo"],[result])
23   CALL CheckError(result, __LINE__)
24   --then CALL GetProperty to check the value again
25   CALL ui.Interface.frontCall("WinCOM", "GetProperty", [xlwb,
"activesheet.Range(A1).Value"], [str])
26   CALL CheckError(str, __LINE__)
27   DISPLAY "content of the cell is: " || str
28
```

```
29   --then Free the memory on the client side
30   CALL freeMemory()
31 END MAIN
32
33 FUNCTION freeMemory()
34   DEFINE res INTEGER
35   IF xlwb != -1 THEN
36     CALL ui.Interface.frontCall("WinCOM","ReleaseInstance", [xlwb],
[res] )
37   END IF
38   IF xlapp != -1 THEN
39     CALL ui.Interface.frontCall("WinCOM","ReleaseInstance", [xlapp],
[res] )
40   END IF
41 END FUNCTION
42
43 FUNCTION checkError(res, lin)
44   DEFINE res INTEGER
45   DEFINE lin INTEGER
46   DEFINE mess STRING
47
48   IF res = -1 THEN
49     DISPLAY "COM Error for call at line:", lin
50     CALL ui.Interface.frontCall("WinCOM","GetError",[],[mess])
51     DISPLAY mess
52     --let's release the memory on the GDC side
53     CALL freeMemory()
54     DISPLAY "Exit with COM Error."
55     EXIT PROGRAM (-1)
56   END IF
57 END FUNCTION
```

# Windows Mail extension

### Topics



---

### Send mail using MAPI

**MAPI** is an acronym for Messaging Application Programming Interface. The **MAPI** extension will create a new mail in the default mailer software, which needs to be "MAPI" compatible, and ask the user to send the mail. The mail sent using MAPI will be stored by the default mailer software in the same way as any other mail created by the user.

---

### Send mail using a SMTP server

Another method of sending mail is to connect directly to an **SMTP** server (Simple Mail Transfer Protocol is the de facto standard for email transmission across the Internet). The extension will connect to a given **SMTP** server and send the mail through this server. The mail is not kept on the client side.

---

## The WinMail API

The WinMail API is based on the front call technique as described in Front End Functions. All WinMail functions are grouped in the **WinMail** front end function module.

| Function name | Description |
|---|---|
| Init | This function prepares a message |
| Close | This function closes the message |
| SetBody | This function sets the body of the mail |
| SetSubject | This function sets the subject of the mail |
| AddTo | This function adds a "To" addressee |
| AddCC | This function adds a "CC" addressee |

| | |
|---|---|
| AddBCC | This function adds a "BCC" addressee |
| AddAttachment | This function adds an attachment |
| SendMail | This function sends the mail |
| GetError | This function retrieves the last error message |

The following functions are needed when you use SMTP server connections:

| Function name | Description |
|---|---|
| setSmtp | This function sets the SMTP server to use |
| setFrom | This function sets the sender address |

---

### Init

### Purpose:

This function initializes the module. It returns the identifier for the message, which will be used in other functions.

### Syntax:

```
CALL ui.Interface.frontCall("WinMail","Init", [], [id ] )
```

Notes:

1. *ret* is the identifier of the message initialized.
2. For each Init function, a Close must be called.

---

### Close

### Purpose:

This function clears all information corresponding to a message, and frees the memory occupied by the message.

### Syntax:

```
CALL ui.Interface.frontCall("WinMail","Close", [id], [ result ] )
```

Notes:

1. *id* is the message identifier
2. *result* is the status of the function.

---

### SetBody

#### Purpose:

This function sets the body of the mail.

#### Syntax:

```
CALL ui.Interface.frontCall("WinMail","SetBody", [ id, body ], [ result
] )
```

#### Notes:

1. *id* is the message identifier
2. *body* is the string text containing the body of the mail.
3. *result* is the status of the function.

---

### SetSubject

#### Purpose:

This function sets the subject of the mail.

#### Syntax:

```
CALL ui.Interface.frontCall("WinMail","SetSubject", [ id, subject ], [
result ] )
```

#### Notes:

1. *id* is the message identifier
2. *subject* is the string text containing the subject of the mail.
3. *result* is the status of the function.

### AddTo, AddCC, AddBCC

#### Purpose:

These functions add a "To" Addressee to the mail. The addressee can be in one of the following categories:

- "To" (to)
- "CC" (carbon copy)
- "BCC" (blind carbon copy)

The Addressee has a name and a mail address.

#### Syntax

```
CALL ui.Interface.frontCall("WinMail","AddTo", [ id, name, address ], [
result ] )

CALL ui.Interface.frontCall("WinMail","AddCC", [ id, name, address ], [
result ] )

CALL ui.Interface.frontCall("WinMail","AddBCC", [ id, name, address ],
[ result ] )
```

#### Notes:

1. *id* is the message identifier
2. *name* is the name to be displayed in the mail.
3. *address* is the mail address to be used for this addressee.
4. *result* is the status of the function.

---

### AddAttachment

#### Purpose:

This function adds a file as an attachment to the mail. The file must be located on the front-end.

#### Syntax

```
CALL ui.Interface.frontCall("WinMail","AddAttachment", [ id, fileName],
[ result ] )
```

#### Notes:

1. *id* is the message identifier
2. *fileName* is the path of the attachment; the path can be relative to the directory from which GDC is run, or absolute.
3. *result* is the status of the function.

---

### SendMail

### Purpose:

This function sends the mail. Using SMTP, the SMTP server is directly used. Using MAPI, the default mailer software is called to create the mail. The user must press the "send" button to send the mail.

### Syntax with SMTP:

```
CALL ui.Interface.frontCall("WinMail","SendMailSMTP", [ id ], [result]
)
```

### Syntax with MAPI:

```
CALL ui.Interface.frontCall("WinMail","SendMailMAPI", [ id ], [result]
)
```

### Notes:

1. *id* is the message identifier
2. *result* is TRUE in case of success; use GetError to get the description of the error when needed.

**Warnings:**

- MAPI needs to log-in to the mailer software. The first log-in could take time, depending on the mailer software. Your Genero application will be blocked until MAPI returns.
- MAPI depends on the mailer software for error management. For instance, Mozilla Thunderbird returns "success" when the mail is created, but Outlook 2002 only returns "success" when the mail is sent.

---

### GetError

### Purpose:

This function gets a description of the last error that occurred.

### Syntax

```
CALL ui.Interface.frontCall("WinMail","GetError", [ id ], [ result ] )
```

### Notes:

1. *id* is the message identifier
2. *result* is the description of the last error.
3. the returned value is NULL if there was no error.

---

### SetSmtp

### Purpose:

This function sets the SMTP server to be used.

### Syntax:

```
CALL ui.Interface.frontCall("WinMail","SetSmtp", [ id, smtp ], [ result
] )
```

### Notes:

1. *id* is the message identifier
2. *smtp* is the string text containing the SMTP server to be used.
3. *result* is the status of the function.

---

### SetFrom

### Purpose:

This function sets sender information. (This is needed for SMTP connections).

### Syntax:

```
CALL ui.Interface.frontCall("WinMail","SetFrom", [ id, name, address ],
[ result ] )
```

### Notes:

1. *id* is the message identifier
2. *name* is the name to be displayed in the mail.
3. *address* is the mail address to be used for this addressee.
4. *result* is the status of the function.

---

## Examples

### Mail using MAPI

The following example sends a mail using MAPI.

```
01 MAIN
02 DEFINE result, id INTEGER
03 DEFINE str STRING
04
05 -- first, we initialize the module
06 CALL ui.Interface.frontCall("WinMail", "Init", [], [id])
07
08 -- Set the body of the mail
09 CALL ui.interface.frontCall("WinMail", "SetBody", [id, "This is a
text mail using WinMail API - MAPI"], [result])
10
11 -- Set the subject of the mail
12 CALL ui.interface.frontCall("WinMail", "SetSubject", [id, "test mail
- ignore it"], [result])
13
14 -- Add an Addressee as "TO"
15 CALL ui.Interface.frontCall("WinMail", "AddTo", [id, "myBoss",
"boss@mycompany.com"], [result])
16 -- Add another Addresse as "BCC"
17 CALL ui.Interface.frontCall("WinMail", "AddBCC", [id, "my friend",
"friend@mycompany.com"], [result])
18
19 -- Add Two attachments
20 CALL ui.Interface.frontCall("WinMail", "AddAttachment", [id,
"c:\\mydocs\report.doc"], [result])
21 CALL ui.Interface.frontCall("WinMail", "AddAttachment", [id,
"c:\\mydocs\demo.png"], [result])
22
23 -- Send the mail via the default mailer
24 CALL ui.Interface.frontCall("WinMail", "SendMailMAPI", [id],
[result])
25 IF result == TRUE THEN
26   DISPLAY "Message sent succesfuly"
27 ELSE
28   CALL ui.Interface.frontCall("WinMail", "GetError", [id], [str])
```

```
29   DISPLAY str
30 END IF
31 CALL ui.Interface.frontCall("WinMail", "Close", [id], [result])
32 END MAIN
```

### Mail using SMTP server

The following example sends a mail using an SMTP server:

```
01 MAIN
02 DEFINE result, id INTEGER
03 DEFINE str STRING
04
05 -- first, we initialize the module
06 CALL ui.Interface.frontCall("WinMail", "Init", [], [id])
07
08 -- Set the body of the mail
09 CALL ui.interface.frontCall("WinMail", "SetBody", [id, "This is a
text mail using WinMail API - MAPI"], [result])
10
11 -- Set the subject of the mail
12 CALL ui.interface.frontCall("WinMail", "SetSubject", [id, "test mail
- ignore it"], [result])
13
14
15 -- Set the mail sender
16 CALL ui.Interface.frontCall("WinMail", "SetFrom", [id, "mySelf",
"me@mycompany.com"], [result])
17
18 -- Set the SMTP server
19 CALL ui.Interface.frontCall("WinMail", "SetSmtp", [id,
"smtp.mycompany.com"], [result])
20
21 -- Add an Addressee as "TO"
22 CALL ui.Interface.frontCall("WinMail", "AddTo", [id, "myBoss",
"boss@mycompany.com"], [result])
23
24 -- Add another Addressee as "BCC"
25 CALL ui.Interface.frontCall("WinMail", "AddBCC", [id, "my friend",
"friend@mycompany.com"], [result])
26
27 -- Add Two attachments
28 CALL ui.Interface.frontCall("WinMail", "AddAttachment", [id,
"c:\\mydocs\report.doc"], [result])
29 CALL ui.Interface.frontCall("WinMail", "AddAttachment", [id,
"c:\\mydocs\demo.png"], [result])
30
31 -- Send the mail via smtp
32 CALL ui.Interface.frontCall("WinMail", "SendMailSMTP", [id],
[result])
33 IF result == TRUE THEN
34   DISPLAY "Message sent succesfuly"
35 ELSE
36   CALL ui.Interface.frontCall("WinMail", "GetError", [id], [str])
37   DISPLAY str
```

```
38 END IF
39 CALL ui.Interface.frontCall("WinMail", "Close", [id], [result])
40 END MAIN
```

---