isCOBOL Evolve: Application Server

Thin Client and distributed processing

Key Topics:

- Usage of isCOBOL Server
- Usage of isCOBOL Client
- Client deployment
- Remote objects
- isCOBOL File Server
- isCOBOL Load Balancer

Overview

isCOBOL programs can run locally or be distributed from a server with isCOBOL Application Server, simplifying distribution steps and improving time-to-market for key applications. With isCOBOL Server, applications are easily distributed to take full advantage of today's multi-threaded processing capabilities on a variety of platforms including UNIX, Linux and Windows.

Getting Started

The setup of a Application Server environment requires the following steps:

1. Download and install Java (JDK or JRE)
2. Download and install isCOBOL Evolve
3. Activate the License

In order to activate your isCOBOL Evolve products, you will need the e-mail you received from Veryant containing your license key. Contact your Veryant representative for details.

Download and install Java (JDK or JRE)

Java must be installed on your machine in order to use isCOBOL Server. For best results and performance, install the latest Java version available for your platform. isCOBOL Server is certified to work correctly with both Oracle JDK and OpenJDK from version 7 to version 11.

A JRE (Java Runtime Environment) is enough to run isCOBOL Server, however a JDK (Java Development Kit) is suggested in order to take advantage of advanced monitoring features.

Self-extracting setups are provided for the Windows platform.

On Unix/Linux platforms Java may be already installed. If it's not the case, you can install it using the appropriate system commands (e.g. yum, or apt-get).

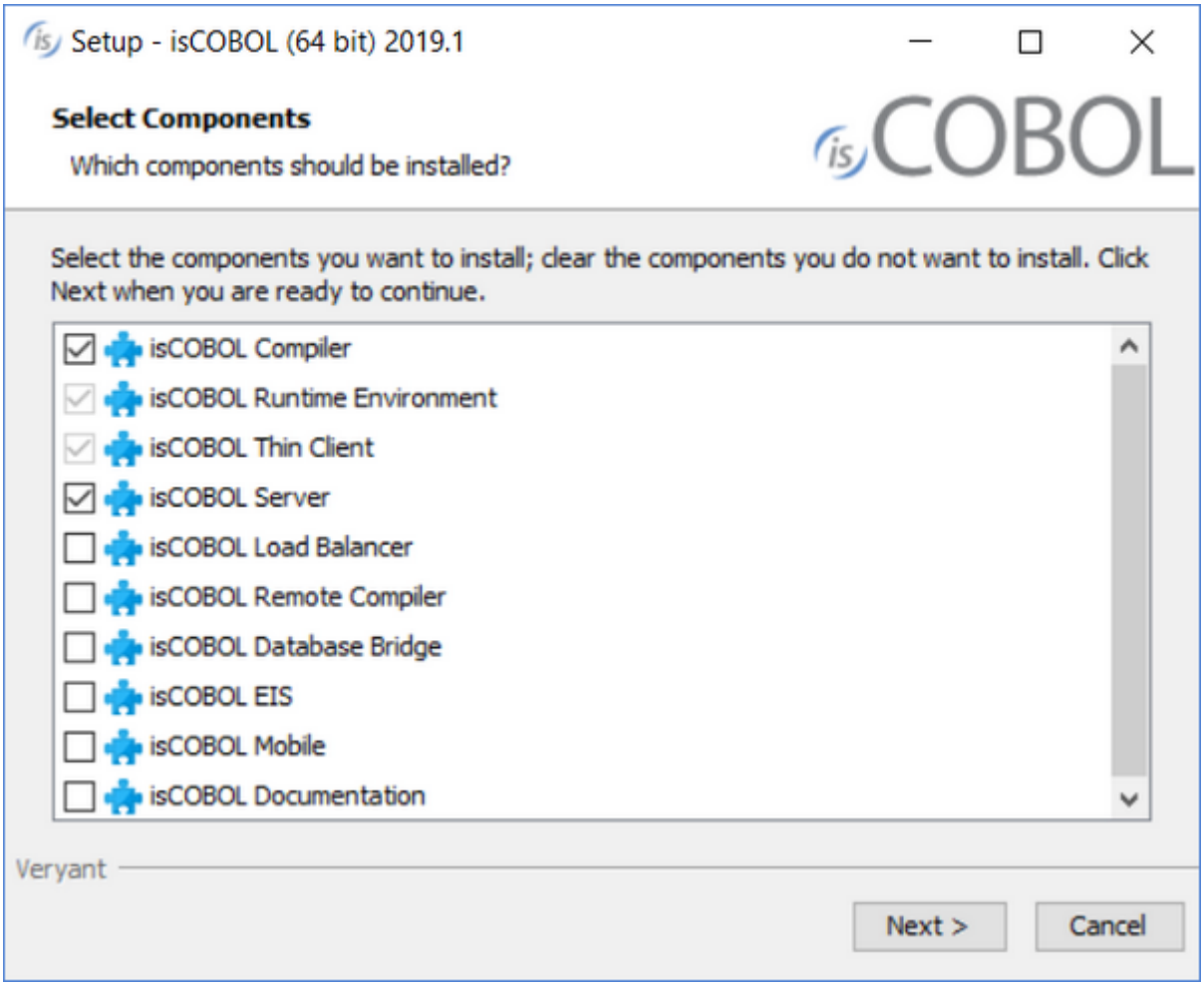
Download and install isCOBOL Evolve

Windows

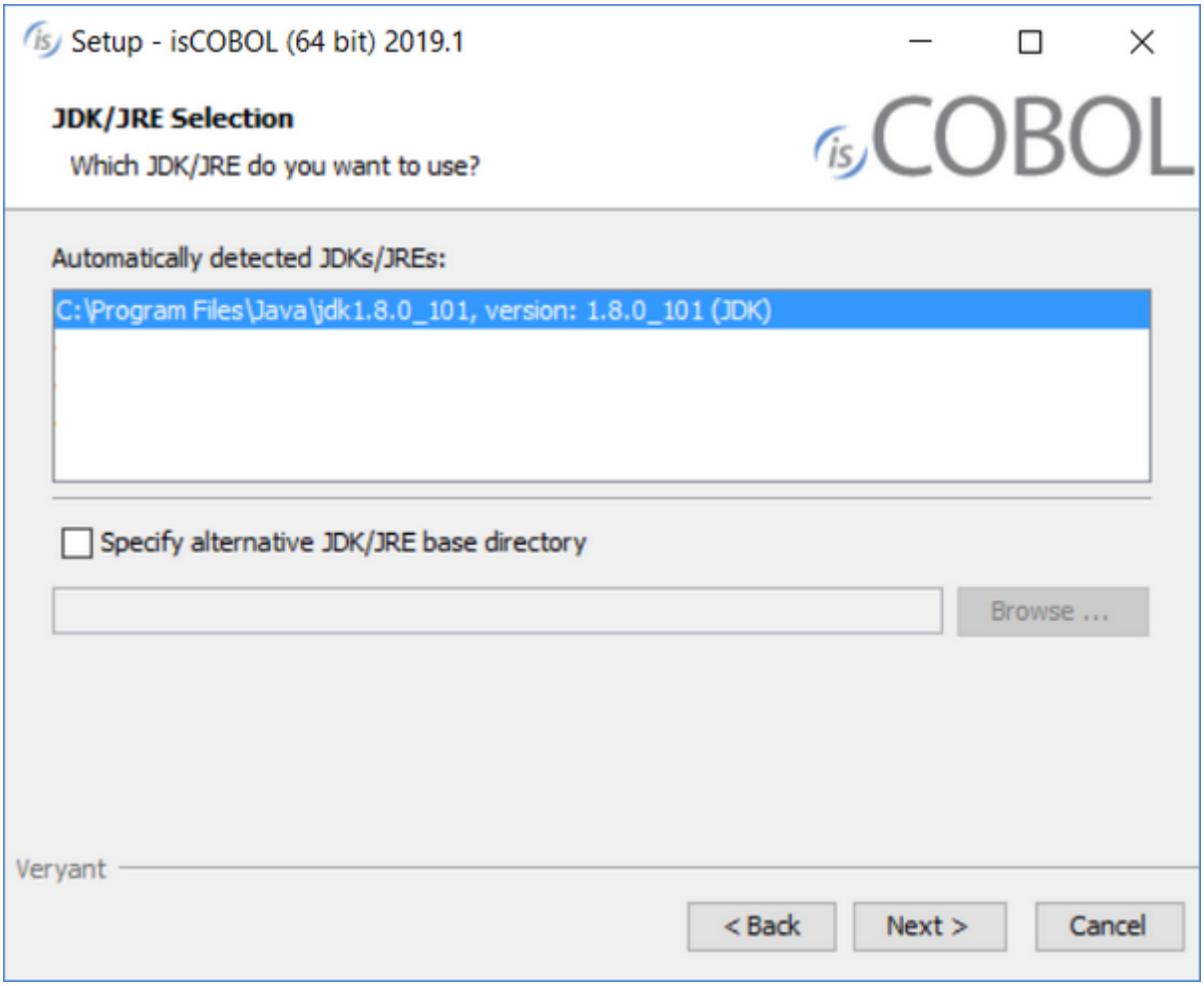
1. If you haven't already done so, Download and install Java (JDK or JRE).
2. Go to "http://www.veryant.com/support".
3. Sign in with your User ID and Password.
4. Click on the "Download Software" link.
5. Scroll down to the list of files for Windows x64 64-bit or Windows x86 32-bit. Select isCOBOL*yyyyR_n*_Windows*arc*.exe, where *yyyy* is the year, *r* is the release number, *n* is the build number and *arc* is the system architecture.
6. Run the downloaded installer to install the files.

Note - If your Windows has the option "Run as Administrator", you should run the setup with that option, otherwise the setting of environment variables might silently fail. Environment variables setting is not necessary if you work from the isCOBOL Shell (explained later).

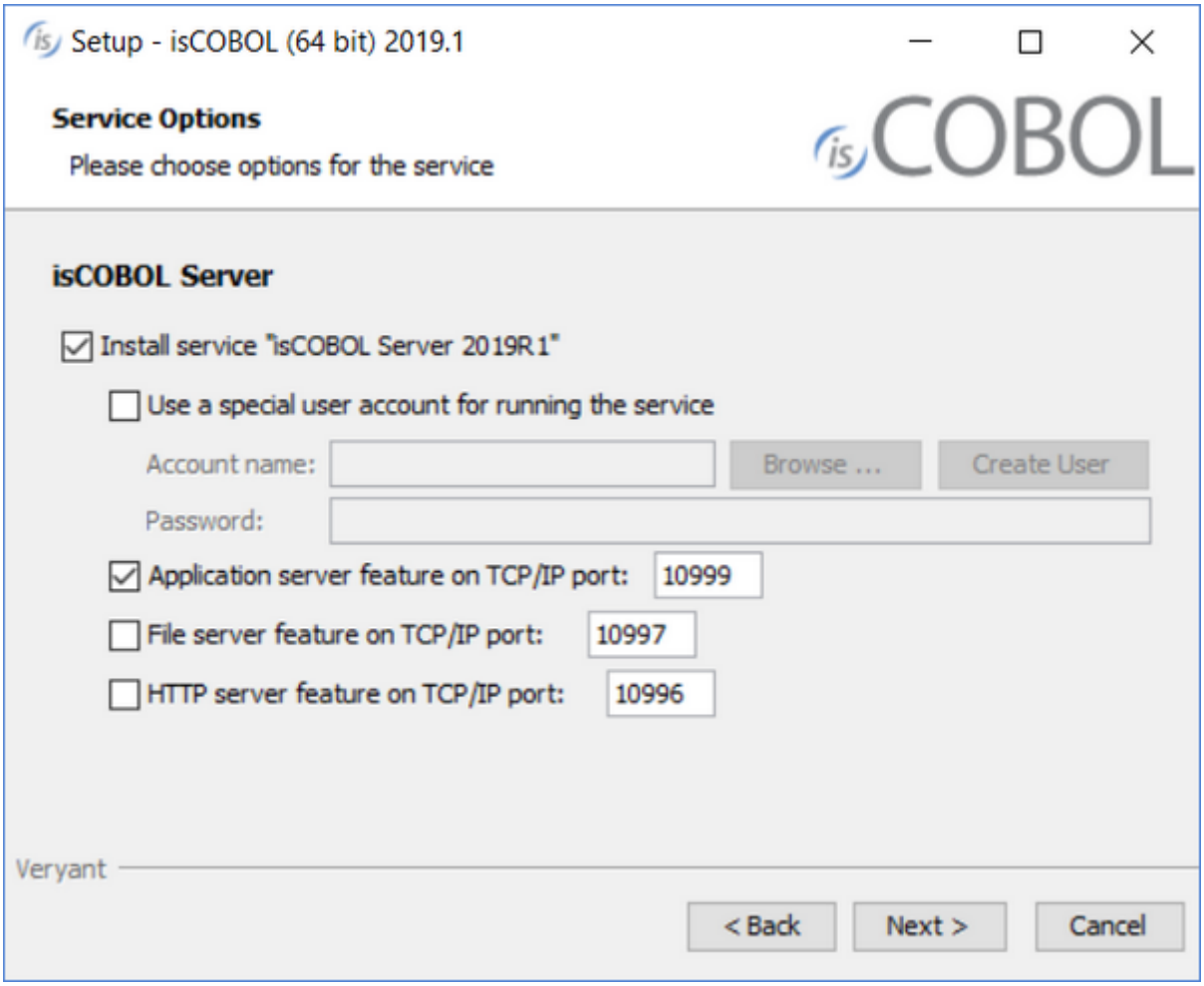
7. Select "isCOBOL Server" from the list of products when prompted.



8. Select your JDK/JRE when prompted



9. Choose if you want to install the Application Server as a system service or not. If you don't install the service, you will have to start the Application Server in foreground mode from a command prompt as explained in Usage of isCOBOL Server. See Windows service and Unix daemon for details about the system service.



10. Follow the wizard procedure to the end. In the process you will be asked to provide license activation and perform it later, as explained in Activate the License.

Linux, Mac OSX and OpenServer

1. If you haven't already done so, Download and install Java (JDK or JRE).
2. Go to "http://www.veryant.com/support".
3. Sign in with your User ID and Password.
4. Click on the "Download Software" link.
5. Scroll down, and select the appropriate .tar.gz file for the product and platform you require.
6. Extract all contents of the archive. For example,
   on Linux 32 bit:

```
gunzip isCOBOL_2019_R1_*_Linux.32.i586.tar.gz
tar -xvf isCOBOL_2019_R1_*_Linux.32.i586.tar
```

on Linux 64 bit:

```
gunzip isCOBOL_2019_R1_*_Linux.64.x86_64.tar.gz
tar -xvf isCOBOL_2019_R1_*_Linux.64.x86_64.tar
```

on Mac OSX:

```
gunzip isCOBOL_2019_R1_*_MacOSX.64.x86_64.tar.gz
tar -xvf isCOBOL_2019_R1_*_MacOSX.64.x86_64.tar
```

on OpenServer:

```
gunzip isCOBOL_2019_R1_*_FreeBSD.64.tar.gz
tar -xvf isCOBOL_2019_R1_*_FreeBSD.64.tar
```

7. Change to the "isCOBOL2019R1" folder and run "./setup", you will obtain the following output:

```
================================================================================

                          isCOBOL EVOLVE Installation
                          For isCOBOL Release 2019R1
                       Copyright (c) 2005 - 2019 Veryant

================================================================================

Install Components:

    [0] All products..................................... (no)
    [1] isCOBOL Compiler (includes [2] & [3])............ (yes)
    [2] isCOBOL Runtime Environment (includes [3])....... (no)
    [3] isCOBOL Thin Client.............................. (no)
    [4] isCOBOL Server................................... (no)
    [5] isCOBOL Load Balancer............................ (no)
    [6] isCOBOL Remote Compiler.......................... (no)
    [7] isCOBOL Database Bridge.......................... (no)
    [8] isCOBOL EIS...................................... (no)
    [9] isCOBOL Mobile................................... (no)

Install Path:
    [P] isCOBOL parent directory: UserHome

JDK Path:
    [J] JDK install directory: JavaHome

[S] Start Install        [Q] Quit

================================================================================
Please press [ 1 2 3 4 5 6 7 8 P J S Q ]
```

8. Type "4", then press Enter to select isCOBOL Server.
9. (optional) Type "P", then press Enter to provide a custom installation path, if you don't want to keep the default one.
10. Type "S", then press Enter to start the installation.

Note - if the setup script is not available for your Unix platform or you don't want to use it, just extract the tgz content to the folder where you want isCOBOL to be installed.

isCOBOL Evolve for UNIX/Linux provides shell scripts in the isCOBOL "bin" directory for compiling, running, and debugging programs. These scripts make use of two environment variables, ISCOBOL to locate the isCOBOL installation directory and ISCOBOL_JDK_ROOT to locate the JDK installation directory. To use these scripts set these environment variables and add the isCOBOL "bin" directory to your PATH.

For example, if you install isCOBOL in "/opt/isCOBOL" and your JDK is in "/opt/java/jdk1.8.0":

```
export ISCOBOL=/opt/isCOBOL
export ISCOBOL_JDK_ROOT=/opt/java/jdk1.8.0
export PATH=$ISCOBOL/bin:$PATH
```

Other Unix

A dedicated setup is provided for the following Unix platforms:

- Linux 32 bit
- Linux 64 bit
- Mac OSX 64 bit
- OpenServer

For any other UNIX platform, the MULTI setup can be used.

Extract the tar with the following command

```
gunzip isCOBOL_Version_multi.tar.gz
tar -xvf isCOBOL_Version_multi.tar
```

These two files are extracted:

- isCOBOL_*Version*.tar
- setup

Run the setup

```
./setup
```

The setup script produces an output like:

```
================================================================================

                            isCOBOL EVOLVE Installation
                           For isCOBOL Release Version
                        Copyright (c) 2005 - 2019 Veryant

 ================================================================================

 Install Components:

    [1] isCOBOL Evolve platform independent files......... (yes)
    [2] isCOBOL ISAM Client component..................... (yes)

Generate Components:

    [3] isCOBOL native libraries.......................... (no)
    [4] isCOBOL support for dummy terminal................ (no)
    [5] isCOBOL File Connectors........................... (no)

Platform:

    [6] Operating System to generate..................... (Platform)

 Install Path:

    [7] isCOBOL parent directory: UserHome

 JDK Path:

    [8] JDK install directory: JavaHome

 [S] Start Install        [Q] Quit

 ================================================================================

Please press [ 1 2 3 4 5 6 7 8 S Q ]
```

The following text depends on the current environment:

Version	version of the isCOBOL components installed by the setup
Platform	current operating system detected by the setup script
UserHome	current user home directory
JavaHome	current JDK/JRE directory detected by the setup script

If points 1 to 8 contain accurate information, you can start the installation process by typing “S” and pressing Enter.

If you want to change any of the points, type the corresponding number and press Enter, then answer to the question. The output shown in the above snippet will be updated to reflect the change you made.

For example, if you want to avoid the generation of isCOBOL ISAM Client component

1. type “2”
2. press Enter
3. type “N”
4. press Enter

When every setting reflect your needs, type “S” and press Enter to start the installation process.

Point 1 can’t be changed while point 6 shouldn’t be set to an operating system different than the one where we’re running the script.

A C compiler is required for generating components (points 3 to 5).

If the MULTI setup completes without error, the following folder is generated:

```
isCOBOLVersion
    bin
    etc
    include
    javadoc
    lib
    native
    sample
```

The content of the folders varies depending on the choices you made before issuing the “S” command.

Distribution Files

For information on a specific distribution file, please see the README file installed with the product.

Activate the License

If you provided license keys during the installation, on Windows, you should skip reading this chapter.

isCOBOL Server looks for the following configuration property for the license key:

```
iscobol.license.2019=<license_key>
```

The key should be stored in one of the following files (if they exist):

Windows

1. \etc\iscobol.properties in the drive where the working directory is
2. C:\Users\<username>\iscobol.properties (the setup wizard saves licenses here, if you don’t skip activation)
3. iscobol.properties found in the Java Classpath
4. %ISCOBOL%\iscobol.properties
5. a custom configuration file passed on the command line

Unix/Linux

1. /etc/iscobol.properties
2. $HOME/iscobol.properties
3. iscobol.properties found in the Java Classpath
4. $ISCOBOL/iscobol.properties

5. a custom configuration file passed on the command line

NOTE - Files are listed in the order they're processed. If the license key appears in more than one of the above files, then the last occurrence is considered.

Usage of isCOBOL Server

Application Server architecture runs most of the application on the server, deploying only the user interface on the client. Some applications may be programmed to run on the client rather than the server.

Before starting the Application Server, you should ensure that your application runs properly on the server as a stand-alone program. Then the Application Server daemon can be started with the following command:

```
iscserver [-c config_file | -conly config_file] [-port port] [-hostname host] [-force]
```

When a TCP connection is closed the connection may remain in a timeout state for a period of time after the connection is closed (typically known as the TIME_WAIT state or 2MSL wait state). For applications using a well known socket address or port, it may not be possible to bind a socket to the required SocketAddress if there is a connection in the timeout state involving the socket address or port. Use the –force option to achieve it.

Config_file should include the standard configuration, that is the same for every client. See Usage of isCOBOL Client for information about how to use a customized client configuration.

Hostname and port can also be specified in the configuration by setting the following properties:

```
iscobol.hostname host
iscobol.port port
```

The following command starts the Application Server on the local PC on the default port 10999.

```
iscserver
```

A correct startup will produce an output similar to this:

```
Application Server started and listening on port 10999
```

On Unix/Linux machines the iscserver command requires either ISCOBOL_JDK_ROOT or ISCOBOL_JRE_ROOT along with ISCOBOL environment variables set in order locate the Java and isCOBOL installation directories and start correctly. If you installed the product using the wizard setup procedure, you can rely on the following alternate command:

```
appserver.sh
```

This script takes care of setting the necessary environment variables and starts the isCOBOL Server with a preset configuration found in the file $ISCOBOL/etc/appserver.properties.

On client machines, the following command should be used:

```
iscclient [-port port] [-hostname host] ProgramName
```

When the client machine is Windows, the following command can also be used:

```
isclient [-port port] [-hostname host] ProgramName
```

The difference is that iscclient keeps the console busy while isclient runs in a separate task.

The isclient command output is stored in two files, *isclient_out.log* and *isclient_err.log*, located in the isCOBOL bin directory. Ensure that the bin directory of isCOBOL has write permissions.

Host name, port, and a remote configuration file can also be set in the configuration file, as shown below.

```
iscobol.hostname host
iscobol.port port
iscobol.remote_conf config
```

See Usage of isCOBOL Client for all the possible options provided by iscclient.

In the thin client architecture every client that connects to the Application Server becomes a thread in the JVM process associated to the Application Server and is assigned with a uniqure progressive identificator number (thread id) in the range between 1 and 2147483647.

The following Java property can be set to avoid unexpected lock errors when two clients open the same relative/sequential file having Lock Mode Automatic:

```
sun.nio.ch.disableSystemWideOverlappingFileLockCheck=1
```

Setting the hostname

If the hostname parameter is not specified neither on the command line nor among configuration properties, then the isCOBOL Server will accept connections from all the local addresses. The following table summarizes the connection result in different combinations of isCOBOL Client and isCOBOL Server hostname parameter values:

Server side hostname value	Client side hostname value	Connection result
none	none	Successful
	127.0.0.1	Successful
	localhost	Successful
	LAN address	Successful
127.0.0.1	none	Successful
	127.0.0.1	Successful
	localhost	Successful
	LAN address	Refused

Server side hostname value	Client side hostname value	Connection result
localhost	none	Successful
	127.0.0.1	Successful
	localhost	Successful
	LAN address	Refused
LAN address	none	Refused
	127.0.0.1	Refused
	localhost	Refused
	LAN address	Successful

Note - The above information is accurate assuming that localhost is mapped to the IP address 127.0.0.1 in the system.

Users count

The Application Server allows multiple client connections depending on the license. The number of allowed concurrent clients is traced in the commentary area of the license file, after the license ID. The following snippet has been taken from a 15 user license:

```
# Company: XXXXX
# License ID: 902368/015
# Expiration Date: none
```

The Application Server counts the different IP addresses that ask for connection. It's possible to connect as many IP addresses as the number of users traced in the license. If a client machine launches different sessions of the COBOL application, it's counted as a single user.

Connections count

The Application Server allows a maximum of 512 concurrent connections by default. This limit can be increased or decreased by setting the iscobol.as.max_connections property in the configuration.

The Application Server supports up to 2147483647 concurrent connections. Each connection is identified by a thread ID, that is a progressive number. The first connection will have a thread ID equal to 1, the second will have a thread ID equal to 2 and so on until the thread ID 2147483647 is reached, thereafter the next connection will use the first free thread ID starting from 1.

Client and Server info

The following sample commands show different ways to obtain information about a client/server environment. To show information about an active thread, use the following command:

```
iscserver -info [-port port] [-hostname host]
```

The following command displays the server version:

```
iscserver -v
```

To show the client version, use the following command:

```
iscclient -v
```

When running on Windows, the following command can also be used to display the client version:

```
isclient -v
```

Usage of isCOBOL Client

Format 1

To execute a program, use the following command:

```
iscclient [--system | --metal | --motif | --GTK | --nimbus] [-port port] [-hostname
host] [-c remote-config] [-lc local-config] [-uc updater-config] [-nodisconnecterr]
[-noupdate] program [ arg1 [arg2] ... ]
```

where:

- *--system, --metal, --motif, --GTK,* and --nimbus specify the look and feel for the GUI screen displayed on the client side.

--system	current system Look and Feel
--metal	Metal Look and Feel
--motif	Motif Look and Feel
--GTK	GTK Look and Feel; not available on Windows
--nimbus	Nimbus Look and Feel

If none of these options is used, then the --system is assumed.

- *port* is the port number the server is listening to.
- *host* is the host machine where the server is running.

port and *host* can specify multiple values separated by comma. See Specifying multiple hostnames and ports for details.

- *remote-config* is the configuration file that the client should use. This file is loaded server-side and its entries are appended to the configuration file used to start the isCOBOL Application Server. This option allows you to have different configurations for different clients.
- *local-config* is the configuration file that hosts client-side settings, for example configuration properties for programs called through the CALL CLIENT statement.

- *updater-config* is a custom configuration file to be passed to the isUpdater that is automatically invoked by the isCOBOL Client at startup unless the -noupdate option is used. If the -uc option is not used, then the isCOBOL Client looks for a file named *isupdater.properties* in Classpath.
- *-nodisconnecterr*, if used, avoids a notification message box to appear when the connection between client and server is lost. This option should always be used when a X Window System (X11) is not available on the client. This option is automatically set by the kill command described in Format 3.
- *-noupdate*, if used, makes the client avoid looking for updates before starting.
- *program* is the program to be executed. It must be a standard COBOL program with PROGRAM-ID. Paths are not allowed in this parameter.
- *arg1* and *arg2* are the arguments passed to the program.

Format 2

To show information about an Application Server module, use the following command:

```
iscclient [-port port] [-hostname host] [-user usr] [-password pwd] -info
```

where:

- *port* is the port number the server is listening to.
- *host* is the host machine where the server is running.

  port and *host* can specify multiple values separated by comma. See Specifying multiple hostnames and ports for details.
- *usr* and *pwd* are the administrator user credentials, that are necessary to access the administration panel under the default configuration. If not passed, then a login prompt will be shown. See Login for more information.

Format 3

To kill a thread running on the specified server, use the following command:

```
iscclient [-port port] [-hostname host] [-user usr] [-password pwd] -kill {threadID }
                                                                        {AS       }
```

where:

- *port* is the port number the server is listening to.
- *host* is the host machine where the server is running.

  port and *host* can specify multiple values separated by comma. See Specifying multiple hostnames and ports for details.
- *usr* and *pwd* are the administrator user credentials, that are necessary to access the administration panel under the default configuration. If not passed, then a login prompt will be shown. See Login for more information.
- *threadID* is the ID of thread to be killed. (Use the -info option to return a list of currently running threads).
- *AS* (an alternate parameter of threadID) indicates that the Application Server should stop. All alive clients are automatically disconnected when the Application Server stops.

Format 4

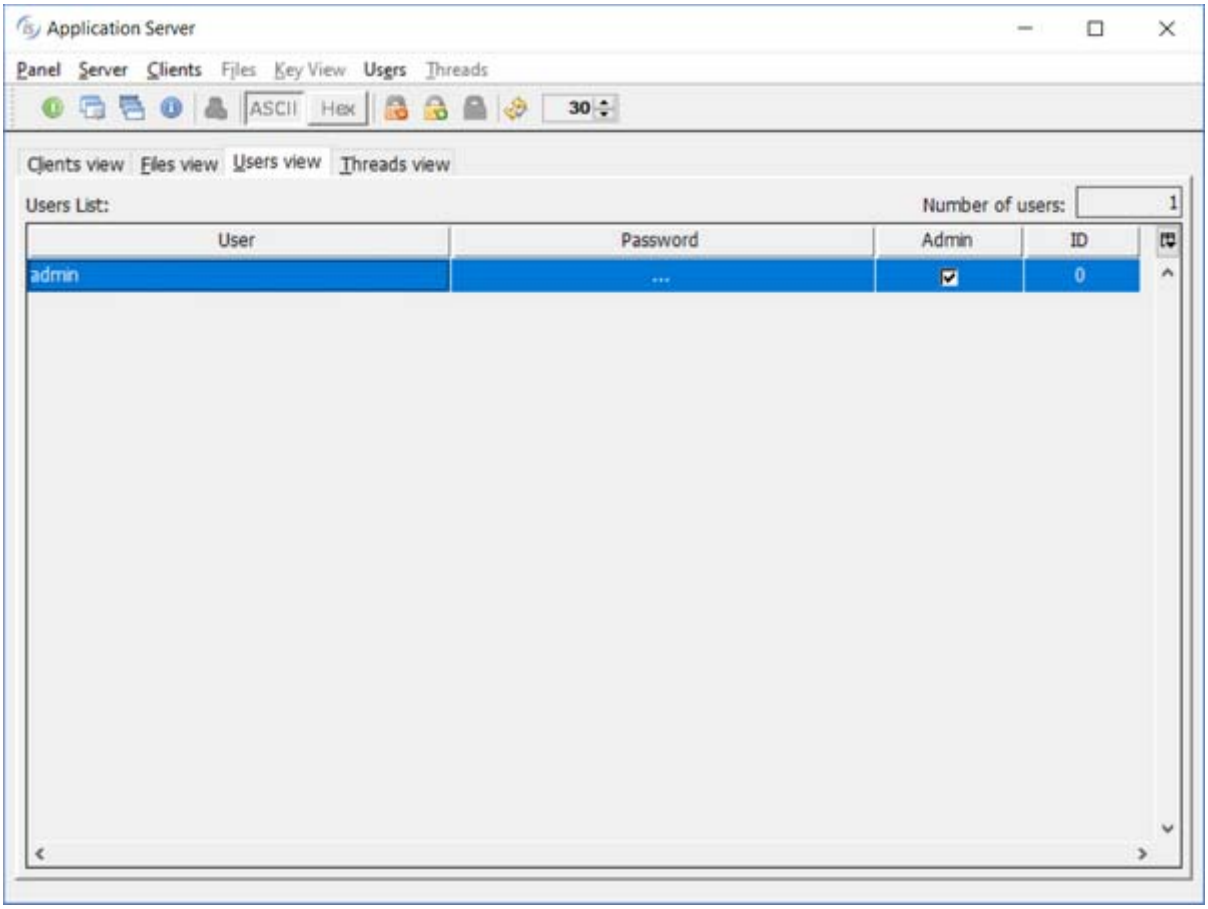
To open a window in which users can be managed, use the following command:

```
iscclient [-port port] [-hostname host] [-user usr] [-password pwd] -admin
```

where:

- *port* is the port number the server is listening to.
- *host* is the host machine where the server is running.

  port and *host* can specify multiple values separated by comma. See Specifying multiple hostnames and ports for details.
- *usr* and *pwd* are the administrator user credentials, that are necessary to access the administration panel under the default configuration. If not passed, then a login prompt will be shown. See Login for more information.



A row for each registered user is shown. Columns have the following meaning:

User	User name
Password	User password

Admin	User privileges. It can be *Admin* or not.
ID	User ID. Admin users have ID=0.

Tool-bar buttons and menu items allow you to

- Add a new user
- Delete an existing user
- Force the garbage collector on the server JVM
- Shut down the Application Server

The table where users are listed is editable. Double click in the cells in order to edit their value.

Format 5

To open a window in which client sessions are managed, use the following command. The administrator can see a list of connected clients, kill clients, and even shutdown the Application Server.

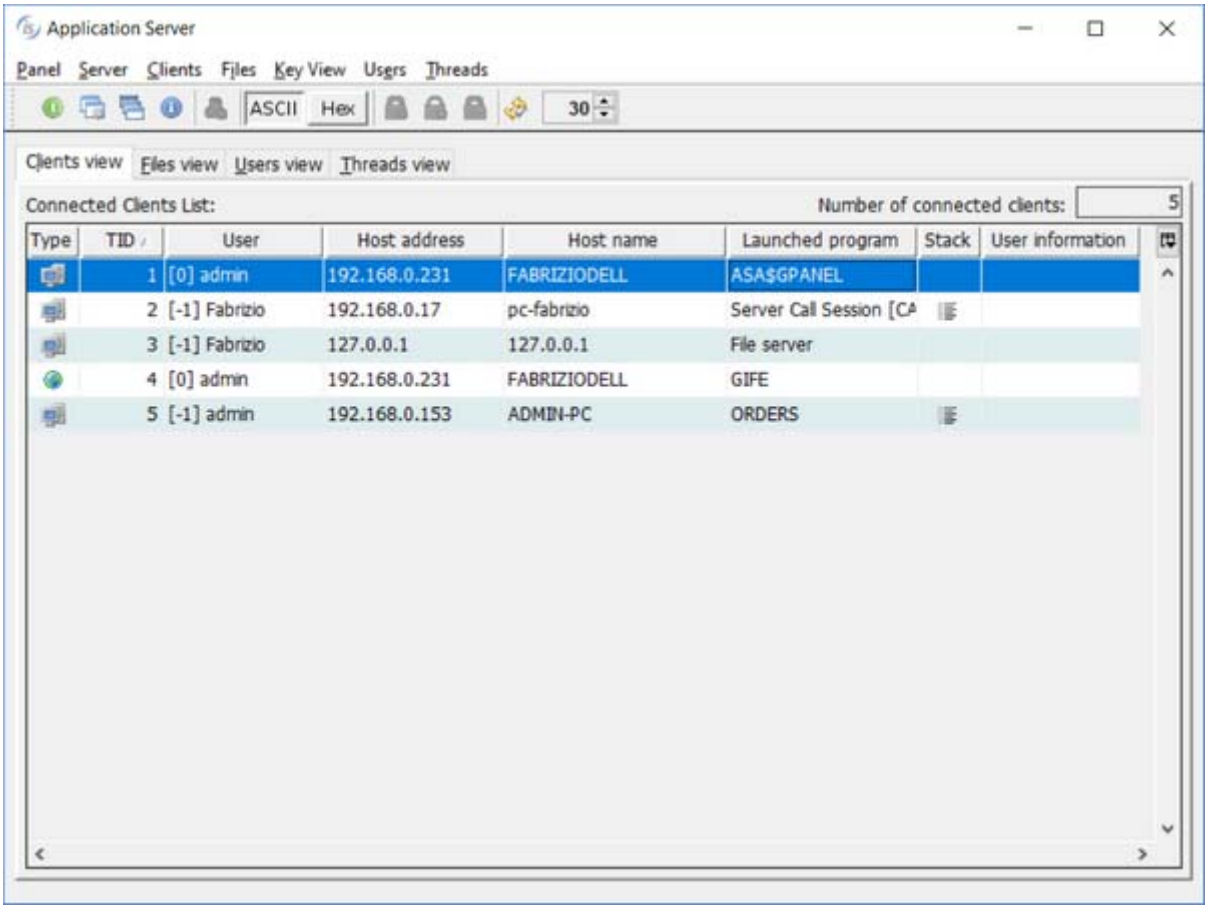
```
iscclient [-port port] [-hostname host] [-user usr] [-password pwd] -panel
```

where:

- *port* is the port number the server is listening to.
- *host* is the host machine where the server is running.

  port and *host* can specify multiple values separated by comma. See Specifying multiple hostnames and ports for details.
- *usr* and *pwd* are the administrator user credentials, that are necessary to access the administration panel under the default configuration. If not passed, then a login prompt will be shown. See Login for more information.

The standard dialog that appears with this command looks like this:

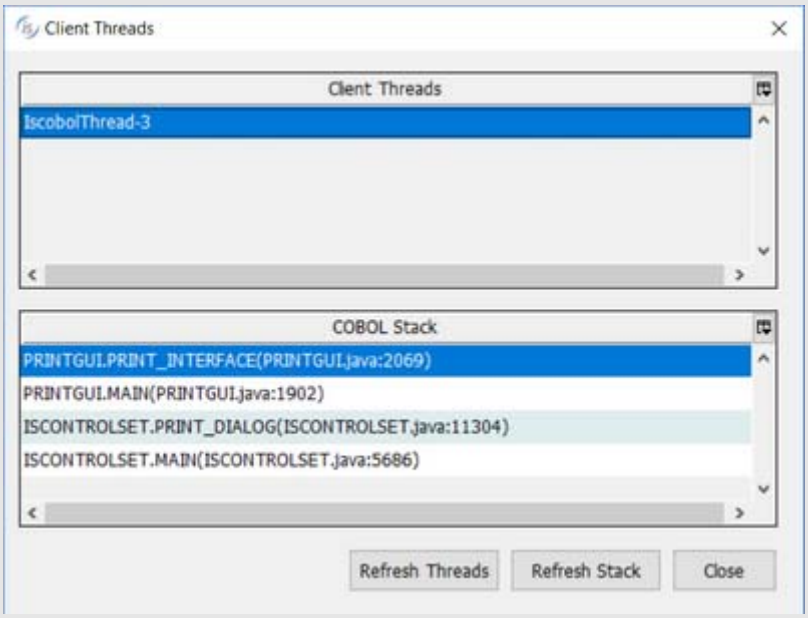


Note that the *Stack* column and the *Threads view* are available only if the isCOBOL Server is running with a JDK (Java Develpment Kit) and the JDK's tools.jar library is in the Classpath. If you find errors on the isCOBOL Server console, like for example "Error opening zip file or JAR manifest missing", then you also have to specify the full path of isCOBOL's utility.jar using the javaagent option, e.g.

```
iscserver -J-javaagent:/path/to/utility.jar
```

A row for each connected client (including the client you used to start the panel) is shown. Columns have the following meaning:

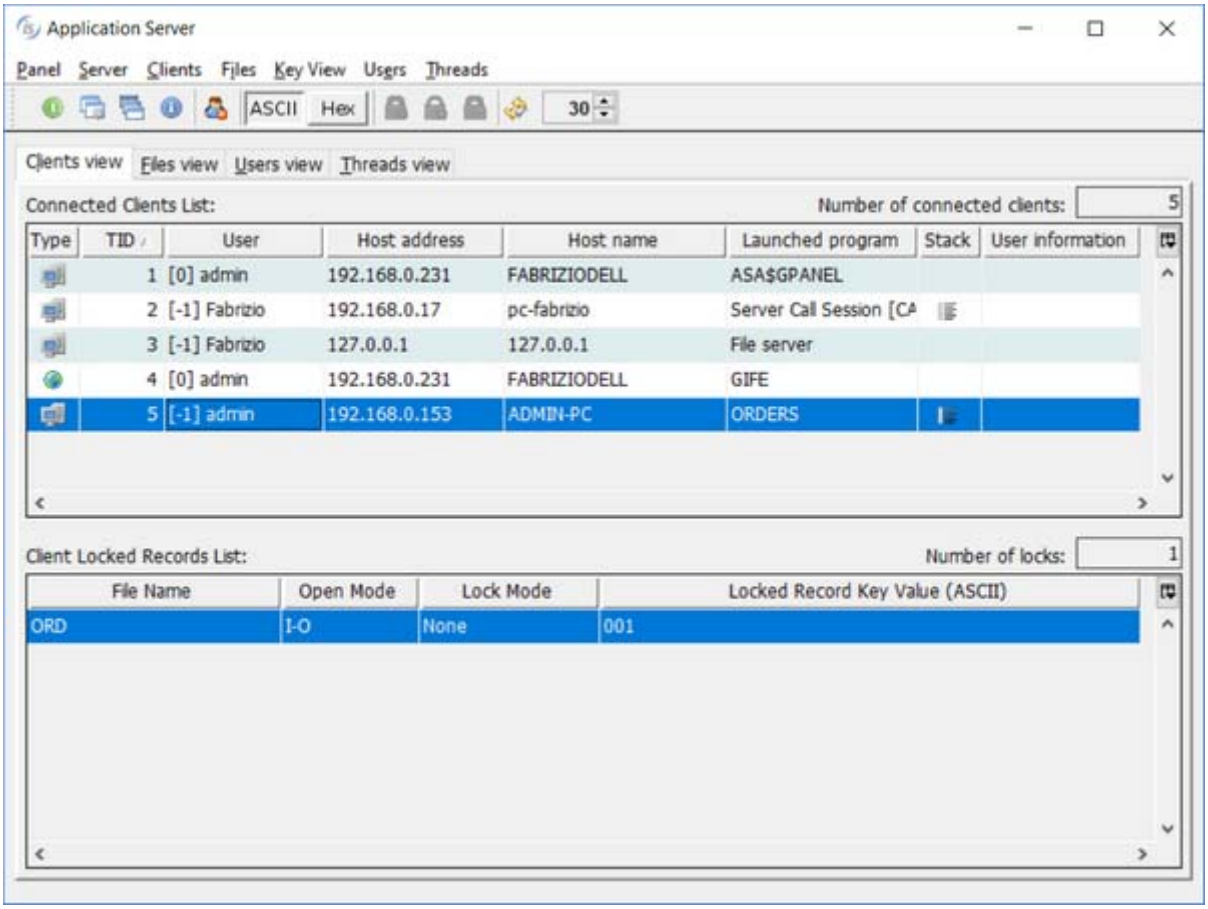
Type	A computer icon indicates that the client is running the program as desktop application via isCOBOL Client. A world icon indicates that the client is running the program in a web browser via WebClient.
TID	Thread ID of the client
User	User name. The number between square brackets is the ID specified in the user administration panel (see Format 4). A value of -1 means that the user has not been registered using the administration panel. In this case the operating system user name is shown
Host address	IP address of the client pc

Host name	Host name of the client pc. If the host name can't be retrieved, the IP address is shown
Launched program	Program name passed in the client command-line or the last program called through CHAIN statement. The special value "File server" identifies a connection to the isCOBOL File Server. This kind of connection cannot be killed from the panel. The special value "Server Call Session" identifies a remote call. The text between square brackets tells the name of the program that was remotely called. See Remote objects for details. This kind of connection cannot be killed from the panel as well
Stack	If the stack icon is available, click on it or press Enter in order to show a dialog that lists the COBOL threads started by that Client. For every thread you can see the stack. The screenshot below shows the info dialog generated for the isCOBOL Demo running in thin client while it was printing. You can see from the stack that the runtime is executing the paragraph ST_ADDRESS in PRINTPROG called by ISCONTROLSET:  The stack information is not available for File Server clients, isCOBOL Utilities and clients running in a separate JVM due to the iscobol.as.multitasking setting.
User information	Custom information stored by calling A$USERINFO

Tool-bar buttons and menu items allow you to

- Shut down the Application Server
- Force the garbage collector on the server JVM
- List loaded programs and unload them (see Unloading programs below for more information)
- Show server settings
- Kill client connections
- Refresh the list of connected clients

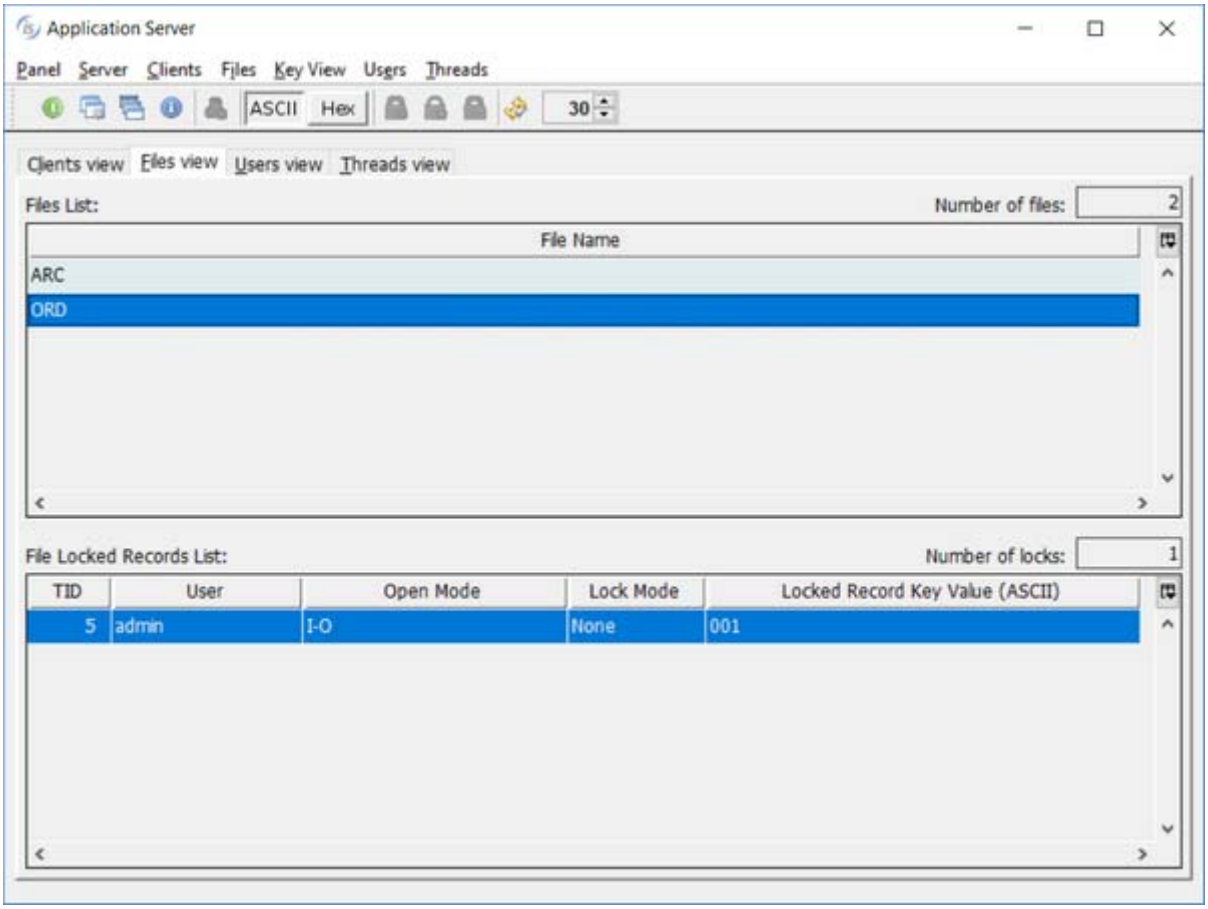
If iscobol.file.lock_manager * is set in the server configuration, then the panel dialog looks more complex:



A row for each record locked by the selected client is shown in the second table. Columns have the following meaning:

File Name	Name of the archive where the lock is found
Open Mode	Open mode of the archive where the lock is found
Lock Mode	Lock mode of the archive where the lock is found
Locked Record Key Value	Value of the locked record primary key

The *File View* page looks like this:



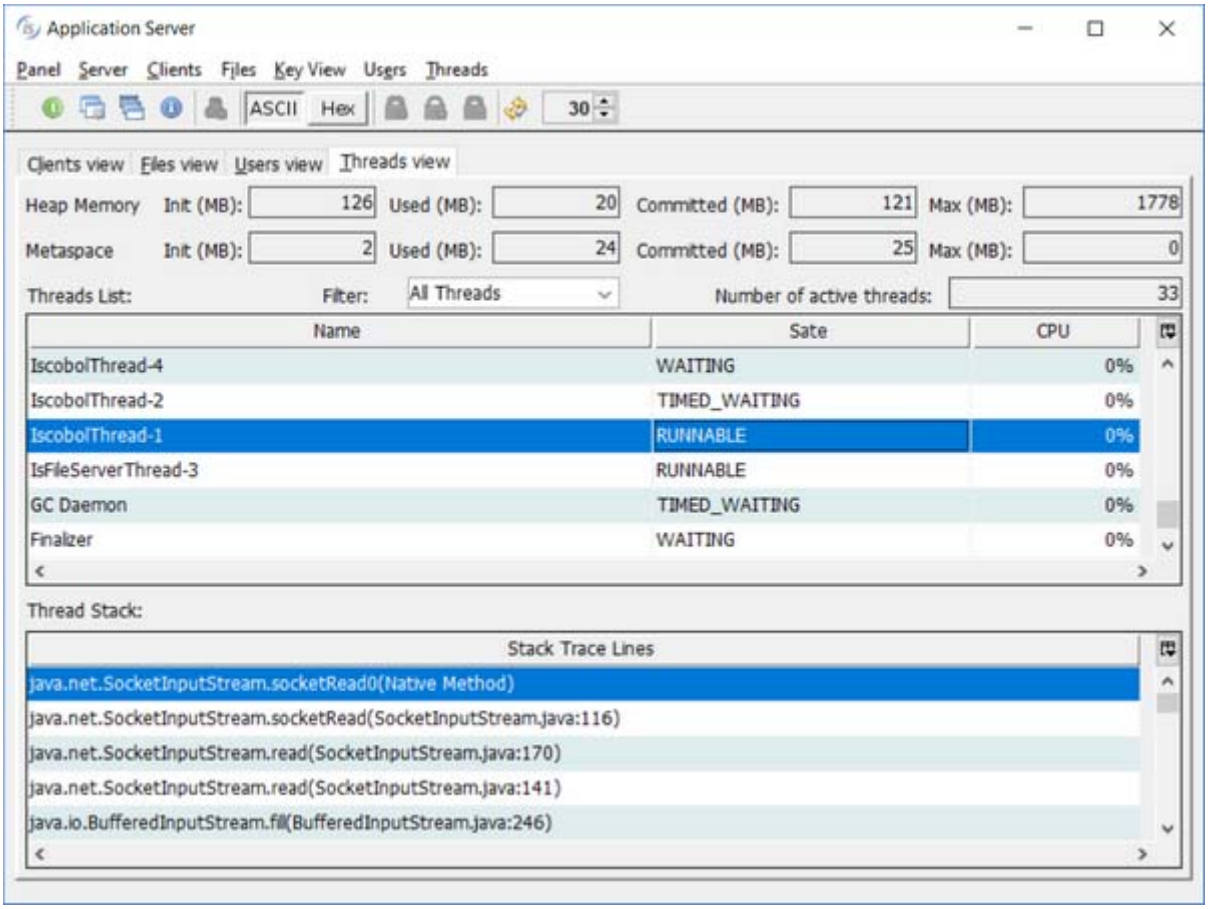
A row for each file with locks is shown at the top. The table below is populated with details about the locked records in the selected file. Columns have the following meaning:

TID	Thread ID of the client that is holding the lock
User	User name used by the client to log in the Application Server.
Open Mode	Open mode used by the client program
Lock Mode	Lock mode used by the client program
Locked Record Key Value	Value of the locked record primary key

The additional tool-bar buttons and menu items allow you to

- Refresh the list of active locks
- Switch between ASCII and hexadecimal visualization of the key value

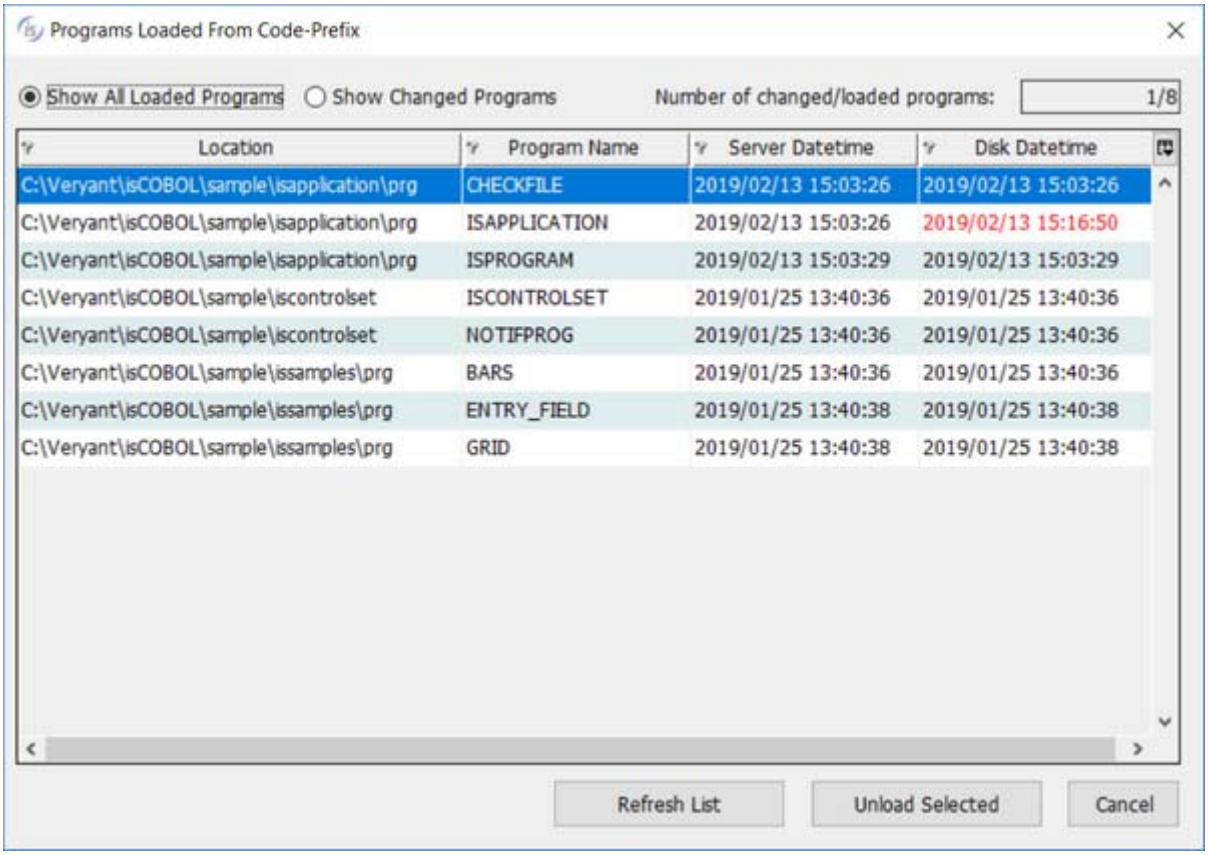
The *Threads view* page shows the list of all the active threads running in the Application Server. For every thread you can see the status, the CPU usage and the stack. It's possible to filter the list in order to see only the COBOL programs threads. It's also possible to terminate a thread, despite this operation is not suggested and should be performed only in critical situations where a thread cannot be terminated in a clean way.



The “Auto refresh” check box in the tool-bar allows you to automatically refresh the lists. The refresh is performed every 30 seconds by default, but the time can be changed using the spinner field or by setting the configuration property iscobol.as.panel.refresh_timeout *.

Unloading programs

Clicking on *Server* in the menu bar and choosing *Loaded Programs* or clicking the corresponding tool-bar button, the following dialog appears:



The table lists all the programs loaded from the paths specified by `iscobol.code_prefix`.

If the property iscobol.code_prefix.reload (boolean)* is set to false, then it's possible to unload the program classes. Select the desired programs by clicking on the corresponding row (hold Ctrl to select multiple rows) then click the button *Unload Selected*. Once a program has been unloaded, the isCOBOL Server will re-load the class from disk the next time this program is called. This feature is useful to update COBOL programs while the isCOBOL Server is running.

Server Datetime holds the last modification timestamp of the class file at the time it was loaded by isCOBOL Server. *Disk Datetime* holds the current last modification timestamp of the class file. If *Disk Datetime* is more recent than *Server Datetime*, a red color is used to highlight that this program has been modified. Programs with a red *Disk Datetime* are suitable to be unloaded. You can activate the *Show Changed Programs* option in order to filter the table content and see only the modified programs.

Programs loaded from the Classpath are not shown in this dialog and can't be unloaded.

Format 6

To debug a remote application from a client pc, use the following command.

```
iscclient [-J-Discobol.debug.code_prefix=src1\n[src2\n]...] [-debugport dport] [-port
   port] [-hostname host] [-c remote-config] [-lc local-config]-d program [ arg1
   [arg2] ... ]
```

where:

- *port*, *host*, *remote-config*, *local-config*, *program*, *arg1* and *arg2* are the same as in Format 1

port and *host* can specify multiple values separated by comma. See Specifying multiple hostnames and ports for details.

- *dport* is the port on which the Remote Debugger will listen for connections (by default: 9999). There is no need to define this port in the server side configuration as iscclient takes care of informing the isCOBOL Server that a Debugger will connect on that specific port, and the isCOBOL Server opens the port automatically.

- *src1* and *src2* are the directories containing the source files for the remote classes. The source files must be found by the Debugger client side. *src1* and *src2* may be separated by \n as shown above if you need a cross-platform command, otherwise you can separate them using the current operating system path separator (e.g. ";" on MS Windows).
  The *iscobol.debug.code_prefix* setting can be stored in *local-config* instead of being passed as a command-line option.

  If the source files are not available client side, you can ask the server to send the source through TCP/IP. In order to activate this feature set iscobol.debug.remote_source (boolean) to true on the client and iscobol.debug.remote_source_enabled (boolean) to true on the server. The server will look for source files in its Classpath and `iscobol.debug.code_prefix` setting.

  Notes:

  - In order to debug, the *iscobol.jar* library is required in the client Classpath
  - By default, during a debug session other client connections are blocked. Set iscobol.as.multitasking=2 in the isCOBOL Server's configuration in order to avoid it.

Format 7

To run a utility in thin client mode, use the following command:

```
iscclient [-port port] [-hostname host] [-user usr] [-password pwd] -utility
utility_name
```

where:

- *port* is the port number the server is listening to.
- *host* is the host machine where the server is running.

  port and *host* can specify multiple values separated by comma. See Specifying multiple hostnames and ports for details.
- *usr* and *pwd* are the administrator user credentials, that are necessary to access the administration panel under the default configuration. If not passed, then a login prompt will be shown. See Login for more information.
- *utility_name* can be any of the following (case insensitive):
  - cobfileio
  - cpk
  - gife
  - isl
  - ismigrate
  - jdbc2fd

- jutil
- xml2wrk

A possible scenario in which this command makes sense is if you have some indexed files stored on a Linux/Unix server machine without desktop and you wish to manage them with GIFE or convert them with ISMIGRATE having the utility GUI displayed on your Windows client PC. E.g.

```
iscclient -hostname 192.168.0.101 -utility ismigrate
```

Specifying multiple hostnames and ports

The *port* and *host* parameters can specify multiple values separated by comma. The client will attempt to connect to the fist available hostname and port pair. Hostnames and ports are paired from the first in the list to the last, such as hostname1:port1, hostname2:port2 and so on. Consider the following command, for example:

```
iscclient -hostname 192.168.0.1,192.168.0.2 -port 5555,5556 MENU
```

The Client will try to connect to IP 192.168.0.1 port 5555 first. If the connection fails, then the Client will try to connect to IP 192.168.0.2 port 5556. If the numbers of specified hostnames and ports do not match, the last in the shorter list will be used for creating all remaining pairs. The following command, for example,

```
iscclient -hostname 192.168.0.1 -port 5555,5556 MENU
```

is equivalent to

```
iscclient -hostname 192.168.0.1,192.168.0.1 -port 5555,5556 MENU
```

while the following command

```
iscclient -hostname 192.168.0.1,192.168.0.2 -port 5555 MENU
```

is equivalent to

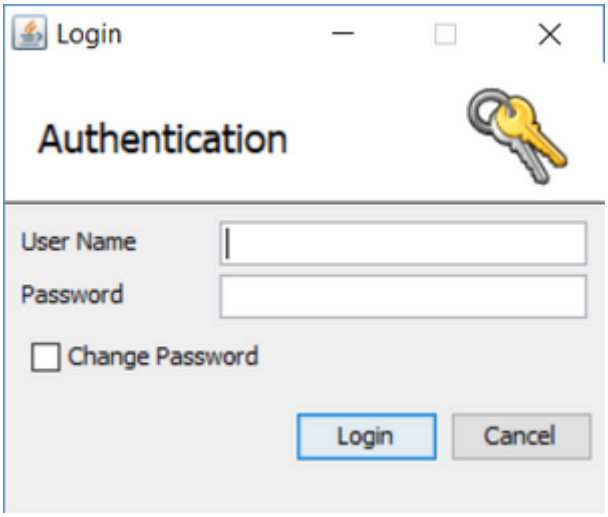
```
iscclient -hostname 192.168.0.1,192.168.0.2 -port 5555,5555 MENU
```

Login

A login is usually only required when a client connects to the Application Server to launch an administrative routine (such as -panel, -info, -kill, or -admin) or a utility. If the property `iscobol.as.authentication *` is set to "2" on the server side, a login is always required when the client connects.

The user credentials can be passed on the Client command line through the options -user and -password. Alternatively they can be set in the Client configuration through the properties `iscobol.user.name` and iscobol.user.password.

If user credentials were not provided at Client startup, the following dialog will open:



The default Administrator credentials are:

User Name = 'admin'

Password = 'admin'

You can change them by launching the Client with the -admin option.

By default user credentials are stored in a file named *password.properties* in the server working directory. You can change the name and location of this file by setting the configuration property iscobol.as.password_file.

Note - On Linux/Unix, in order to encrypt passwords, Java access to /dev/random, a special file that serves as a random number generator. It allows access to environmental noise collected from device drivers and other sources. The bits of noise are stored in a pool. When the pool is empty, reads from /dev/random will block until additional environmental noise is gathered. A counterpart to /dev/random is /dev/urandom which reuses the internal pool to produce more pseudo-random bits. This means that the call will not block, but the output may contain less entropy than the corresponding read from /dev/random.

If your client needs too much time to connect when the authentication is required, you might consider to instruct Java to use /dev/urandom instead of /dev/random, by adding the following option to the Application Server startup command-line:

```
-Djava.security.egd=file:///dev/urandom
```

Custom Login

isCOBOL offers the ability to create a custom login, which displays a custom window or no window at all. Before showing the default login window, the Application Server calls A$CUSTOM_LOGIN on the client machine. If this program is found, it is used instead of the default.

This program must be called A$CUSTOM_LOGIN, must be reachable in the client CLASSPATH, and must use the following linkage code.

```
----
LINKAGE SECTION.

  77 login-user pic n any length.
  77 random-value pic x any length.
  77 password-hashed-hash pic x any length.
  77 new-password-crypted-hash pic x any length.
  77 flags pic 9.
  77 new-password-min-length pic 99.
----
```

The following table describes the parameters for the linkage code:

Parameter	Description
login-user (output parameter)	Returns the username for the login.
random-value (input parameter)	Use this value to obtain the digest of the password.
password-hashed-hash (output parameter)	Returns a hashed password.
new-password-crypted-hash (output parameter)	Returns the encrypted hash of the new password, or spaces if the password is unchanged.
flags (optional input parameter)	May contain one of the following values: 0 => password change optional (default) 1 => password change mandatory 2 => check password weakness
new-password-min-length (optional input parameter)	Contains the minimum length of the password, use this value to check your password before returning it.

The program returns “0” if the login has been confirmed, or “-1” if the login has been cancelled.

An example of a custom login GUI is installed with isCOBOL. You can find it in the folder $ISCOBOL_HOME/sample/as/custom-login.

Note - The custom login program is called only in replacement of the standard login dialog. If user credentials were passed on the command line or set in the configuration, then the custom login program is not called.

Automatic Client update

If iscobol.as.clientupdate.site is set in the server side configuration, each time a thin client is launched, it looks for updates before starting the COBOL program.

The automatic update process can be skipped by adding the -noupdate option to the client command line.

The automatic update process is described below:

1. The isCOBOL Server sends to the following information to the thin client:

- URL of the Update Server as specified by iscobol.as.clientupdate.site in the server side configuration. This URL can point to either a third party HTTP server or an isCOBOL Server started with the -hs option, as described in isCOBOL Server as an HTTP server. This server is configured by a file named *swupdater.properties* where the following entries must be provided:

Entry	Meaning
swupdater.version.iscobol	Build number of the jar libraries.
swupdater.version.iscobolNative	Build number of the native libraries. It should match with the build number of the jar libraries. See Configuring different repositories for native libraries for details about platform and bitness.
swupdater.lib.iscobol	Folder or zip file containing the jar libraries. Note - if the HTTP server is not the isCOBOL Server, it must be a zip and can't be a folder.
swupdater.lib.iscobolNative	Folder or zip file containing the native libraries. See Configuring different repositories for native libraries for details about platform and bitness. Note - if the HTTP server is not the isCOBOL Server, it must be a zip and can't be a folder.

See the Update Facility Server configuration for more information about the HTTP server setup.

- Current client build number as specified by iscobol.as.clientupdate.version in the server side configuration. If this property is omitted, then the thin client will use the build number of its jar libraries.

2. The thin client runs the Update Facility to connect to the Update Server and check for updates. It uses the following default configuration:

Default configuration for Windows Clients	Default configuration for Linux/Mac OS Clients

`swupdater.version.iscobol=<clientBuildNumber>`
`swupdater.version.iscobolNative=<clientBuildNumber>`
`swupdater.directory.iscobol=<isCOBOLInstallDir>/lib`
`swupdater.directory.iscobolNative=<isCOBOLInstallDir>/bin`
`swupdater.directory.clean.iscobol=true`
`swupdater.directory.clean.iscobolNative=true`
	`swupdater.version.iscobol=<clientBuildNumber>`
	`swupdater.version.iscobolNative=<clientBuildNumber>`
	`swupdater.directory.iscobol=<isCOBOLInstallDir>/lib`
	`swupdater.directory.iscobolNative=<isCOBOLInstallDir>/native/lib`
	`swupdater.directory.clean.iscobol=true`
	`swupdater.directory.clean.iscobolNative=true`

Note - It is possible to customize the above configuration by putting a *isupdater.properties* configuration file in the client side Classpath or using the -uc option in the Client command line. See Client Configuration (isupdater.properties) for more information about the possible configuration entries.

a. If the server is down or no update is necessary, the thin client execution continues normally

b. If some updates were executed, the thin client is automatically restarted with the -noupdate option

The need of updating is determined by comparing the build numbers specified by the swupdater.version properties used by the client with the build numbers specified by the swupdater.version properties in the server side *swupdater.properties* file.

Configuring different repositories for native libraries

The Update Server may need to provide different native libraries to different clients depending on the client operating system and architecture. This is achieved by extending the *iscobol.version.iscobolNative* and *iscobol.lib.iscobolNative* property names as demonstrated below.

In the snippet below we show how to fill the *swupdater.properties* file for a Linux server that needs to update Windows clients both 32 and 64 bit, having all the necessary items stored in the /repository1 folder on the HTTP server:

```
swupdater.version.iscobol=875.6
swupdater.lib.iscobol=/repository1/lib
swupdater.version.iscobolNative=875.6
swupdater.lib.win.32.iscobolNative=/repository1/native/win32/lib
swupdater.lib.win.64.iscobolNative=/repository1/native/win64/lib
```

We notice that the 'iscobol' package is the same for all operating systems and architectures, instead the 'iscobolNative' packages are specified with different directories, one for each possible operating system and architecture combination.

Note - It is possible also to specify only the operating system, e.g.

```
swupdater.lib.win.iscobolNativeWin=<directory>
```

For example, if the client is Windows 64 bit, it checks if there is a library specified for Windows 64 bit; if not, it checks if there is a library specified for Windows; if not, then it requires the generic library.

No action is required client side. The client receives from the server the list of the available libraries and automatically detects its operating system and architecture, so it downloads the proper libraries.

TSL/SSL support

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols designed to provide communication security over a connection. All the data being sent is encrypted by one side, transmitted, then decrypted by the other side before processing. This protocol relies on asymmetric cryptography, so to enable a SSL connection the Application Server needs to have a Digital Certificate which will allow clients to trust the server authenticity. This Digital Certificate may be issued by a Certificate Authority (CA) or you can create your own Certificate (so called self-signed Certificate): the difference is that many of the Certificate Authorities are known by the JavaTM Runtime Environment (more than 80 in version 8), so that you don't need to install anything on the client, while if you use a self-signed certificate, you must install it on the client too.

isCOBOL Application Server relies on JSSE (JavaTM Secure Socket Extension). In the Sun/Oracle version you need to get also the JCE (JavaTM Cryptography Extension) in order to get unlimited strength cryptography. In the JSSE specification, certificates are stored in a file called keystore: according to JavaTM documentation:

"A keystore is a database of key material. Key material is used for a variety of purposes, including authentication and data integrity. Generally speaking, keystore information can be grouped into two different categories: key entries and trusted certificate entries. A key entry consists of an entity's identity and its private

key, and can be used for a variety of cryptographic purposes. In contrast, a trusted certificate entry only contains a public key in addition to the entity's identity". Thus you need to have a keystore with a key entry (with both private and public key) onthe server side and a trusted certificate entry on the client side. JavaTM supports the JKS (JavaTM KeyStore) format and it may contain both key entries and trusted certificate entries. In order to handle this file format the command line program keytool is provided with the standard JDK distribution (a more user friendly tool can be freely downloaded from the Internet, i.e. KeyStore Explorer (http://keystore-explorer.org/).

If you need a Certificate issued by a CA then the procedure to get it may change from one organization to another. In any case you need a SSL certificate importable in a JKS keystore as well as any other Java server application, e.g. Tomcat. Note however that some Java server application may also use different formats while currently isCOBOL Application Server supports only the JKS format. So, let's see an example about how to create a self-signed Certificate using the keytool program. You can find all the information about this tool in the Oracle site, http://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html. The keytool program is located in the bin directory under the JavaTM Home.

For the sake of simplicity let's assume that we can invoke keytool supplying only the name. To create a new keystore from scratch, containing a single self-signed Certificate, execute the following from a terminal command line:

```
keytool -genkeypair -alias iscobol -keyalg RSA
```

After executing this command, you will first be prompted for the keystore password. You can chose any password you like at least 6 characters long. Then you will be asked about general information about this Certificate, such as company, contact name, and so on. This information will be displayed to users who attempt to access a secure page in your application, so make sure that the information provided here matches what they will expect.

Finally, you will be prompted for the key password, which is the password specifically for this Certificate (as opposed to any other Certificates stored in the same keystore file). The keytool prompt will tell you that pressing the ENTER key automatically uses the same password for the key as the keystore. The JSSE framework, and isCOBOL by consequence, requires these passwords to be identical.

If everything was successful, you now have a new file, named .keystore under your HOME directory. You can specify a different name and location using the -keystore option or use a different encryption algorithm through the -keyalg option.

Now you can establish a secure connection between client and server inserting in the isCOBOL configuration file the following entries:

```
iscobol.conf.var_delimiters=${.}
home=${user.home}
# server side
iscobol.net.ssl.key_store=${home}/.keystore
iscobol.net.ssl.key_store_password=mypassword
# client side
iscobol.net.ssl.trust_store=${home}/.keystore
iscobol.net.ssl.trust_store_password=mypassword
```

If you got a certificate from a CA known by the JavaTM Runtime Environment then you don't need to have that certificate on the client, however you need to instruct the client to use an encrypted connection. In order to do so you have to add the following line in the client configuration file:

```
iscobol.net.ssl.trust_store=*
```

This line instructs the client to use an encrypted communication and to use the standard default keystore to acknowledge the server.

TLS version

There are different versions of the TLS protocol. The default version used by isCOBOL depends on the underlying Java version:

Java7	uses TLSv1.0 by default.
Java8 and greater	uses TLSv1.2 by default.

In order to use TLSv1.2 with Java7, the *jdk.tls.client.protocols* Java property can be used on isCOBOL client and server command line, e.g.

```
iscserver -J-Djdk.tls.client.protocols=TLSv1.2  ....

iscclient -J-Djdk.tls.client.protocols=TLSv1.2  ....
```

Working with Aliases

Programs can be run in the Application Server through aliases. An alias is a logical name used client side to identify a specific program run with a specific configuration file.

In order to activate such feature, the following property must be set in the server side configuration:

```
iscobol.as.use_aliases=true
```

Aliases are defined in the server side configuration with properties in the format:

```
iscobol.as.alias.<alias_name>=<PROGRAM_NAME>,<configuration_file>
```

For example, the following server configuration file defines two aliases

- the first alias runs the program MAIN with the default configuration
- the second alias runs the program TEST with the configuration file /usr/test/config1.properties

```
iscobol.as.use_aliases=true
iscobol.as.alias.menu=MAIN
iscobol.as.alias.test_alias=TEST,/usr/test/config1.properties
```

On the client side the isCOBOL Client specifies the alias name instead of the program name in its command-line. For example, if a client wants to run the TEST program with the /usr/test/config1.properties configuration file, it will just run:

```
iscclient -hostname my-server TEST_ALIAS
```

The effect will be the same as running the equivalent more complex command:

```
iscclient -hostname my-server -c /usr/test/config1.properties TEST
```

Note - The alias name on the client command-line is case insensitive, it means that the following commands are all valid and produce the same effect:

- iscclient -hostname my-server TEST_ALIAS
- iscclient -hostname my-server test_alias
- iscclient -hostname my-server Test_Alias

Aliases for isCOBOL Client options

When `iscobol.as.use_aliases=true`, the following aliases must be defined in order to make all the isCOBOL Client options work correctly:

```
iscobol.as.alias.asa$gadmin=ASA$GADMIN
iscobol.as.alias.asa$gpanel=ASA$GPANEL
```

Tracing the Thin Client Activity

isCOBOL provides two kinds of log that allow you to trace the activity of the server and the clients.

Tracing Application Server Activity

In order to trace the activity of the Application Server, the following entries must appear in the configuration when you start it:

```
iscobol.as.logging=1
iscobol.as.logfile=AppServer.log
```

The log file contains information about the server startup and the clients that connect to it. The following snippet is the result of a correct startup on localhost:

```
17-mar-2010 16.23.57 com.iscobol.as.AppServerImpl main
INFO: Starting server on hostname: null with port number: 10999
17-mar-2010 16.23.57 com.iscobol.as.ServerHandler init
INFO: AppServer bound in registry
17-mar-2010 16.23.57 com.iscobol.as.ServerHandler init
INFO: LockManager: com.iscobol.io.DefaultLockManager
17-mar-2010 16.24.16 com.iscobol.as.ServerHandler <init>
INFO: new AppServerImpl
```

Tracing Clients Activity

In order to trace the activity of the clients, the following entries must appear in the configuration when you start the Application Server:

```
iscobol.logfile=Client.log
iscobol.tracelevel=11
```

A log file for each client connection will be generated along with a file with the lck extension that Java uses to avoid unexpected overwrite. The name of the log file is composed of the value of iscobol.logfile followed by a progressive number. The file whose name matches with the iscobol.logfile setting contains the configuration read by the Application Server. For example, using the above settings, if three clients connect to the Application Server, the following files will be generated:

```
Client.log
Client.log.lck
Client.log.1
Client.log.1.lck
Client.log.2
Client.log.2.lck
Client.log.3
Client.log.3.lck
```

The content of these files varies depending on iscobol.tracelevel value. With the above setting, which uses a value of 11, the log contains: client configuration, i/o and programs executed.

If you wish to trace the activity of a single client, you can set iscobol.logfile and iscobol.tracelevel properties in the remote configuration file used by the client instead of the standard configuration file used by the Application Server. See Format 1 for details about remote configuration.

Client deployment

When you share your application through the network using a thin client architecture, one of the issues to address is the the deployment of the client part. Every client PC that is going to run the programs that reside on the server needs to install and run the client components.

The most common ways to deploy the client part are:

- Deployment through automatic client update and istc files
- Deployment through setup programs
- Deployment through isCOBOL EIS WebClient
- Deployment through Java Web Start (JavaWS)

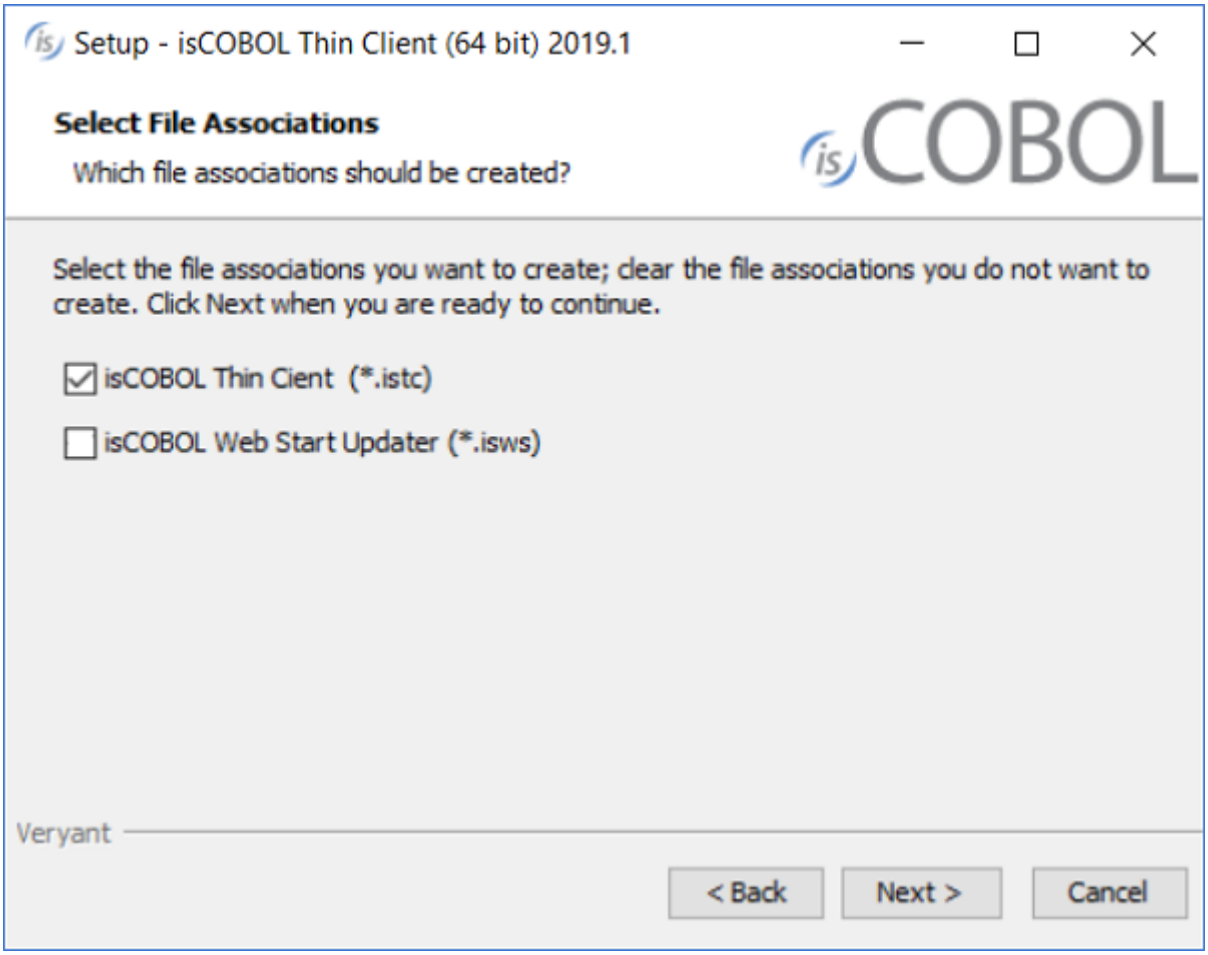
Deployment through automatic client update and istc files

The information in this chapter is applicable to thin client environments where the client machines are Windows.

To deploy the application through the Automatic Client update feature of isCOBOL Server, the isCOBOL Client must be installed on the client machines.

Install either isCOBOL_*yyyy*_*R*_*n*_Windows_*arc*.exe (it requires Java installed on the machine) or isCOBOL_*yyyy*_*R*_*n*_THIN_Windows_*arc.ext* (it doesn't require Java on the machine as it installs its own JVM) where where *yyyy* is the year, *R* is the release number, *n* is the build number, *arc* is the system architecture and *ext* is either "exe" or "msi".

When prompted, choose to associate the istc extension to the isCOBOL Client:



The istc files are property files that include the isCOBOL configuration entries that you would use client side in order to instruct the isCOBOL Client about the server location, the remote configuration file and the main program as well as custom entries that are inquired by the client side logic of the COBOL application. These files could be passed to isCOBOL Client via the -lc option. The installer creates an association between the istc extension and the command:

```
isclient -lc %1
```

In the following chapters we describe how to set up the deployment through automatic client update and istc files in two different scenarios:

Scenario 1 – Server and Client of the same kind or no native parts involved

Scenario 2 – Server and Client of the different kind and native parts involved

Scenario 1 – Server and Client of the same kind or no native parts involved

The simplest scenario where the the isCOBOL update feature can be used is a thin client environment where the isCOBOL Client and the isCOBOL Server have been installed on machines with the same operating system and architecture (e.g. both client and server are Windows 64 bit) by keeping the folder structure proposed by the isCOBOL setup.

Having different operating systems on the server and on the client shouldn't be an issue because isCOBOL is based on Java and Java is cross-platform. This is not 100% true: although most of the isCOBOL runtime is written in Java and therefore is cross-platform, there are some native components that are native libraries and so platform-dependent. Some of them may be required client side and so be included in the automatic update process that the isCOBOL Client performs at startup. These components are the dyncall library that isCOBOL uses to call dynamic link libraries and the SWT libraries used by the default web-browser control implementation.

If your COBOL application doesn't use any of these items (e.g. it doesn't use any browser or uses the JavaFX browser implementation and it doesn't call DLLs client-side with CALL CLIENT statements), then the information included in this chapter will be enough for your needs, otherwise refer to the information provided in Scenario 2 – Server and Client of the different kind and native parts involved.

What to do server side

You should consider to delete the file *swt-###-##.jar* from the isCOBOL lib directory, in order to avoid garbling the client's lib folder if it includes a different version of this library.

The isCOBOL Server must be started with the options –as –hs in order to activate both the Application Server and the HTTP Server features, e.g.

```
iscserver -J-Discobol.as.clientupdate.site=http://serverNameOrIp:10996 –as –hs
```

Where *serverNameOrIp* is the server name or IP address of the server machine.

If you're starting the isCOBOL Server as a Windows service, then the two command line options can't be used, so you need to edit the file *isserver.vmoptions* in the isCOBOL bin folder and put the following entries into it:

```
-Discobol.as.clientupdate.site=http://serverNameOrIp:10996
-Discobol.as.appserver=1
-Discobol.as.httpserver=1
```

Create a file named swupdater.properties in the isCOBOL Server's working directory and put the following entries into it:

```
swupdater.version.iscobol=###
swupdater.lib.iscobol=C:/Program Files/Veryant/isCOBOL2019R1/lib
swupdater.version.iscobolNative=###
swupdater.lib.iscobolNative=C:/Program Files/Veryant/isCOBOL2019R1/bin
```

Where ### il the build number of the isCOBOL Server. For example, for "release 2019 R1 build#977-20181026-26441" you would use "977".

If you use third party jar libraries that need to be updated also client-side, consider to copy them to the isCOBOL *lib* folder so they're included in the automatic update process.

If you have programs that should be installed client-side as they're called via CALL CLIENT statement, consider to create a jar library for them and copy this library to the isCOBOL *lib* folder so it's included in the automatic update process.

What to do client side

Create a file with istc extension, e.g. *myapp.istc* and put the following entries into it:

```
iscobol.hostname=serverNameOrIp
iscobol.port=10999
iscobol.default_program=MYPROG
```

Where *MYPROG* is the name of the program that you wish to execute. The class of this program must be found either in the server's Classpath or in the server side `iscobol.code_prefix` setting.

Double clicking on *myapp.istc* will trigger the program execution. It also will update the local copy of the isCOBOL runtime if necessary.

Double clicking on a istc file is equivalent of running the isCOBOL Client with the option –lc that points to the istc file. So, if you're replacing a isCOBOL Client command, and you used the –lc option in your original command, then you should include all the entries of the file pointed by –lc into the new istc file.

The file *myapp.istc* could be distributed via internet in the form of a file to be downloaded and executed.

Scenario 2 – Server and Client of the different kind and native parts involved

In this second scenario we describe a more complex situation where the server machine has a different operating system or architecture than the client machines. A common situation is to have a Linux server the serves clients of different kind, e.g. Windows clients with either 32 or 64 bit architecture.

What to do server side

The user should gather the platform-dependent components from the corresponding isCOBOL setups and place them in specific folders on the server machine.

Here is a suggestion: create the following subfolders in the isCOBOL installation directory (that we assume as */home/veryant/isCOBOL2019R1*):

Directory	What to copy inside
libWin32	Content of the lib folder of isCOBOL for Windows 32 bit
libWin64	Content of the lib folder of isCOBOL for Windows 64 bit
binWin32	Content of the bin folder of isCOBOL for Windows 32 bit
binWin64	Content of the bin folder of isCOBOL for Windows 64 bit

The isCOBOL Server must be started with the options –as –hs in order to activate both the Application Server and the HTTP Server features, e.g.

```
iscserver -J-Discobol.as.clientupdate.site=http://serverNameOrIp:10996 –as –hs
```

Where *serverNameOrIp* is the server name or IP address of the server machine

Create a file named *swupdater.properties* in the isCOBOL Server's working directory and put the following entries into it:

```
swupdater.version.iscobol=###
swupdater.lib.win.32.iscobol=/home/veryant/isCOBOL2019R1/libWin32
swupdater.lib.win.64.iscobol=/home/veryant/isCOBOL2019R1/libWin64
swupdater.version.iscobolNative=###
swupdater.lib.win.32.iscobolNative=/home/veryant/isCOBOL2019R1/binWin32
swupdater.lib.win.64.iscobolNative=/home/veryant/isCOBOL2019R1/binWin64
```

Where ### il the build number of the isCOBOL Server. For example, for "release 2019 R1 build#977-20181026-26441" you would use "977".

If you use third party jar libraries that need to be updated also client-side, consider to copy them to the isCOBOL *lib* folder so they're included in the automatic update process.

If you have programs that should be installed client-side as they're called via CALL CLIENT statement, consider to create a jar library for them and copy this library to the isCOBOL *lib* folder so it's included in the automatic update process.

If you have custom native libraries that should be installed client side, copy them to the proper "bin<Platform>" folder (e.g. if you have a library named *mylib.dll* for both Windows 32 bit and Windows 64 bit, copy the 32 bit version to *binWin32* and copy the 64 bit version to *binWin64*).

If you would like that the clients use specific Java options (e.g. –Xmx to set the maximum amount of memory), put these options in the file *isclient.vmoptions* stored in the folders *binWin32* and *binWin64*.

What to do client side

Create a file with istc extension, e.g. *myapp.istc*, and put the following entries into it:

```
iscobol.hostname=serverNameOrIp
iscobol.port=10999
iscobol.default_program=MYPROG
```

Where *MYPROG* is the name of the program that you wish to execute. The class of this program must be found either in the server's Classpath or in the server side `iscobol.code_prefix` setting.

Double clicking on *myapp.istc* will trigger the program execution. It also will update the local copy of the isCOBOL runtime if necessary.

Double clicking on a istc file is equivalent of running the isCOBOL Client with the option –Ic that points to the istc file. So, if you're replacing a isCOBOL Client command, and you used the –Ic option in your original command, then you should include all the entries of the file pointed by –Ic into the new istc file.

The file *myapp.istc* could be distributed via internet in the form of a file to be downloaded and executed.

Deployment through setup programs

Veryant provides executable setup programs for Windows and tgz archives for Unix/Linux. The client machines should be provided with the proper setup files and the user should follow these steps in order to run programs in thin client.

Windows

1. install either isCOBOL_*yyyy_R_n*_Windows_*arc*.exe (it requires Java installed on the machine) or isCOBOL_*yyyy_R_n*_THIN_Windows_*arc.ext* (it doesn't require Java on the machine as it installs its own JVM) where where *yyyy* is the year, *R* is the release number, *n* is the build number, *arc* is the system architecture and *ext* is either "exe" or "msi".
2. open the isCOBOL Shell from the Windows Start menu. The isCOBOL Shell is available in the isCOBOL programs group.
3. Run one of the commands documented in Usage of isCOBOL Client.

Unix/Linux

1. Unpack the tgz in a folder of your choice
2. add the isCOBOL bin directory to the $PATH
3. Run one of the commands documented in Usage of isCOBOL Client.

Deployment through isCOBOL EIS WebClient

If you're looking for a zero client installation, then isCOBOL EIS WebClient is the way to go.

With this kind of solution, the isCOBOL Client runs on a web server and the end users can interact with the COBOL application GUI by just connecting to that web server using a web browser. The application GUI is displayed within the web browser.

The only limitation in this scenario is that the COBOL application cannot access client resources (e.g. create a file on the client machine) because the Client is running on the web server and not on the end user's PC.

See WebClient option for more information.

Deployment through Java Web Start (JavaWS)

Java Web Start (JavaWS) is a technology that allows users to start application software for the Java Platform directly from the Internet using a web browser.

Note - Java Web Start is not included in Java SE 11 (18.9 LTS) and later. Developers will need to transition to other deployment technologies. If you're using Java version 11 or later, consider the Deployment through automatic client update and istc files.

This chapter explain how to set up JavaWS on the server machine so that users can run the launch script from their browser.

Requirements:

- classes (and dynamic link libraries, if any) must be provided through jar library files
- in order to avoid errors related to security checks performed by the latest JVMs, *Permissions* should be set to "all-permissions" in the MANIFEST file. To achieve it, proceed as follows:

    a. create a text file, e.g. *mymanifest.txt* and put the following line into it:

```
Permissions: all-permissions
```

    b. add an empty line after it

    c. update the jar library files and include the text file as new manifest, for example:

```
jar -ufm iscobol.jar mymanifest.txt
```

- involved jar library files must be signed

Steps:

1. Digitally sign the iscobol.jar file. For more detailed information, read http://docs.oracle.com/javase/7/docs/technotes/guides/javaws/developersguide/contents.html

   For development and demonstration you can use a self-signed test certificate. (A trust-worthy certificate can be obtained from a certificate authority, such as VeriSign or Thawte, and should be used when the application is put into production).

   For example,

    a. Make sure that you have an JDK 1.5 or later keytool and jarsigner in your path. These tools are located in the JDK bin directory.

b. Create a new key in a new keystore as follows:

```
keytool -genkey -keystore myKeystore -alias myself
```

You will get prompted for information about the new key, such as password, name, etc. This will create the myKeystore file on disk.

c. Then create a self-signed test certificate as follows:

```
keytool -selfcert -alias myself -keystore myKeystore
```

This will prompt for the password. Generating the certificate may take a few minutes.

d. Check to make sure that everything is okay. To list the contents of the keystore, use this command:

```
keytool -list -keystore myKeystore
```

It should list something like:

```
Keystore type: jks
Keystore provider: SUN
```

Your keystore contains 1 entry:

```
myself, Tue Jan 23 19:29:32 PST 2001, keyEntry,
Certificate fingerprint (MD5):
C2:E9:BF:F9:D3:DF:4C:8F:3C:5F:22:9E:AF:0B:42:9D
```

e. Finally, sign the JAR file with the test certificate as follows:

```
jarsigner -keystore myKeystore iscobol.jar myself
```

Note: For most cases, iscobol.jar contains all of the classes necessary for the client. If your application requires other jar files on the client, then you must also sign those jar files by repeating the jarsigner command line above. However, it may be more convenient to combine everything into one jar file.

2. Edit the *isclient.jnlp* file:

    a. Create a file named *isclient.jnlp* with the following contents:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp codebase="http://127.0.0.1" href="isclient.jnlp">
 <information>
  <title>isCOBOL sample</title>
  <vendor>Veryant</vendor>
 </information>
 <security>
  <all-permissions/>
 </security>
 <resources>
  <j2se version="1.6+"/>
  <j2se version="1.5+"/>
  <jar href="iscobol.jar"/>
 </resources>
 <application-desc main-class="com.iscobol.gui.client.Client">
  <argument>-hostname</argument>
  <argument>192.168.0.17</argument>
  <argument>-port</argument>
  <argument>1234</argument>
  <argument>-c</argument>
  <argument>myapp.properties</argument>
  <argument>MYAPP</argument>
 </application-desc>
</jnlp>
```

    b. Change the URL in codebase="http://127.0.0.1" to the URL location of your jar file on the web server machine. For example, if iscobol.jar is located at http://www.mycompany.com/myapp/iscobol.jar then set codebase="http://www.mycompany.com/myapp"
    Note - the jnlp pointed by the *href* attribute is the one that is actually executed.

    c. Change title and vendor

    d. (Optional) Add additional <jar href=.../> lines if you have more than one jar file to deploy to the client.

    e. Change the hostname (192.168.0.17), port number (1234), remote properties file (myapp.properties), and program name (MYAPP) to the appropriate values for your isCOBOL Server and COBOL application. You can delete lines relating to optional arguments that you don't use.

3. Place *isclient.jnlp* and your *iscobol.jar* file in the directory you have chosen on your web server. You may rename *isclient.jnlp*, though the extension should remain *.jnlp*.

4. Configure your web server software to return application/x-java-jnlp-file as the MIME-type (Content-Type) for JNLP files. For example, for Apache Web Server, edit /etc/apache/httpd.conf and add the following line:

```
AddType application/x-java-jnlp-file .jnlp
```

5. Open port 10999 or other port that you choose to run the isCOBOL Server in the firewall settings on your server.

6. Now test your setup by visiting the URL of the .jnlp file (e.g. http://www.mycompany.com/myapp/isclient.jnlp).

If you see the "Java Starting" splash screen and after answering the security warning dialog nothing seems to happen, then there is likely to be a connection or isCOBOL Server configuration problem. To get diagnostic information you can configure Java to show the Java Console. For example, select "Java" from the Windows

control panel and select “Java Console/Show Console” on the advanced tab. Then run your test again, the Java console will pop up and the specific error will appear in the console window

Security issues

Starting with Java 7 Update 51, Java doesn’t allow users to run applications that are not signed (unsigned), self-signed (not signed by trusted authority) or that are missing permission attributes.

If you followed the above steps, then you obtained a self-signed application, that may return one of these errors when ran with a recent Java:

- Java applications are blocked by your security settings.
- Missing Application-Name manifest attribute
- Missing required Permissions manifest attribute in main jar

As a workaround, you can use the Exception Site list feature to run the applications blocked by security settings. Adding the URL of the blocked application to the Exception Site list allows it to run with some warnings. The exception site list is managed in the *Security* tab of the *Java Control Panel*. The list is shown in the tab. To add, edit or remove a URL from the list, click *Edit Site List*.

Remote objects

The isCOBOL Application Server can also work as a repository for backend programs that can be called from client machines. Programs running in stand-alone as well as program running on a client pc in Thin Client environment can call programs that reside on a server computer where an isCOBOL Application Server is running. This objective is achieved through a simple CALL Statement. The connectivity with the Application Server that hosts the remote objects is configured by setting `iscobol.remote.code_prefix`.

Programs loaded from remote.code_prefix are executed server-side using server resources. These programs can communicate with the calling program through LINKAGE SECTION items. They cannot have a user interface, they cannot display anything and must not accept user input. Only backend programs can run correctly as remote objects.

COBOL programs as well as C functions can be called remotely.

A program called in thread is never searched among remote objects.

Example.

Consider the following setting:

```
iscobol.remote.code_prefix=isc://192.168.0.1:10999
```

And the following statement:

```
CALL "PROG1" USING param-1, param2 GIVING rc.
```

isCOBOL will try to load PROG1 from the local CLASSPATH and code_prefix first. If the program is not found, then isCOBOL will try to load PROG1 through an Application Server running on the ip 192.168.0.1 on port 10999. The program will be searched remotely in all the paths listed in the CLASSPATH and code_prefix of the remote machine (ip 192.168.0.1). If the program is still not found, then a "ClassNotFound" error is returned.

Any Exception thrown on the server is returned to the client, including the internal StopRunException, so that a client program can be interrupted when there is a STOP RUN statement or an exception (e.g. wrong linkage section) in the server program: in the former case the program will stop silently without showing any message.

If the remote programs are compiled with the `-cp` option, then this different syntax must be used to set the `iscobol.remote.code_prefix`:

```
iscobol.remote.code_prefix=iscp://192.168.0.1:10999
```

Using Aliases

Remote calls can be done through aliases. An alias is a logical name used client side to identify a specific program run with a specific configuration file.

In order to activate such feature for remote calls, the following property must be set in the server side configuration:

```
iscobol.as.call.use_aliases=true
```

Aliases are defined in the server side configuration with properties in the format:

```
iscobol.as.call.alias.<alias_name>=<PROGRAM_NAME>,<configuration_file>
```

For example, the following server configuration file defines two aliases

- the first alias runs the program PROG1 with the default configuration
- the second alias runs the program TEST with the configuration file /usr/test/config1.properties

```
iscobol.as.call.use_aliases=true
iscobol.as.call.alias.utility1=PROG
iscobol.as.call.alias.utility2=TEST,/usr/test/config1.properties
```

On the client side the program calls the alias name instead of the program name. For example, if you want to run the TEST program remotely using the remote configuration file /usr/test/config1.properties, you will just do:

```
CALL "utility2".
```

User Authentication

If `iscobol.as.authentication *` is set to 2 in the server configuration, `iscobol.user.name` and iscobol.user.password must be set client side in order to specify login information.

Hook program

The isCOBOL Server provides the ability to define a hook.

A hook is a program that is automatically executed when a Client starts and when it exits.

This feature provides entry points to define additional operations that should be done for each client session, for example a custom logging of thin client activity.

The hook program is defined through the iscobol.as.hook property.

For example, if you want the Application Server to run the program MYHOOK for each client session, you will set the following entry in the server configuration:

```
iscobol.as.hook=MYHOOK
```

The hook program must be found in the server Classpath, it's never loaded from `iscobol.code_prefix` paths.

The hook program can retrieve useful information about the client session by inquiring the following configuration properties:

Property Name	Type	Value
iscobol.as.info.entering	numeric	1 -> Program starting 0 -> Program exiting
iscobol.as.info.userid	numeric	user ID
iscobol.as.info.username	alphanumeric	user name
iscobol.as.info.program	alphanumeric	called program
iscobol.as.info.arguments	alphanumeric	arguments of the program
iscobol.as.info.host	alphanumeric	client host address

In addition, the program can retrieve the client thread id by calling the A$GETTHREAD library routine.

The installed sample, available in $ISCOBOL_HOME/sample/as/hook directory, shows two common useful usages of this feature:

1) Creating a custom log that traces login time and logout time of each client session.

2) Performing an automatic shutdown of the Application Server after a specific time-out if no client is still connected.

To test it, go to the sample/as/hook folder and launch:

```
iscserver
```

This will start the Application Server, reading the iscobol.properties file stored in the same directory, that contains the as.hook setting.

Now, launch the administration panel from a client. If you launch it from the same pc, it's enough to use:

```
iscclient -panel
```

If you launch it from a different pc, use:

```
iscclient -hostname ip-address -panel
```

Where *ip-address* is the IP address of the machine on which you started the Application Server.

Close the panel and look in the sample/as/hook folder, on the server. A new file named access.log will be there. If you edit it, you'll see something similar to this:

```
[001 ENTER - 31/03/2010 11:01:10] 00000 - admin - ASA$GPANEL - 127.0.0.1
[001 EXIT - 31/03/2010 11:03:24] 00000 - admin - ASA$GPANEL - 127.0.0.1
```

If you wait for three hours without connecting anymore, the Application Server will automatically shut down, with an exit status of 1.

This is useful if you plan to periodically re-start the Application Server when no one is working on it, in order to clean up memory.

The time-out of three hours is configured by using the following constant in the MYHOOK source code:

```
78 RUNNINGHOURS               value 3.
```

The special exit status (different than 0) that allows you to intercept if the Application Server was shut down by the hook program and not by the administrator user is set by using the following statement in MYHOOKTIMER source:

```
java-lang-System:>exit(1).
```

Internal lock management

isCOBOL Server allows locks on indexed files to be managed internally, without demanding the lock request to the file handler. In order to activate this feature, the following setting must appear in the server configuration:

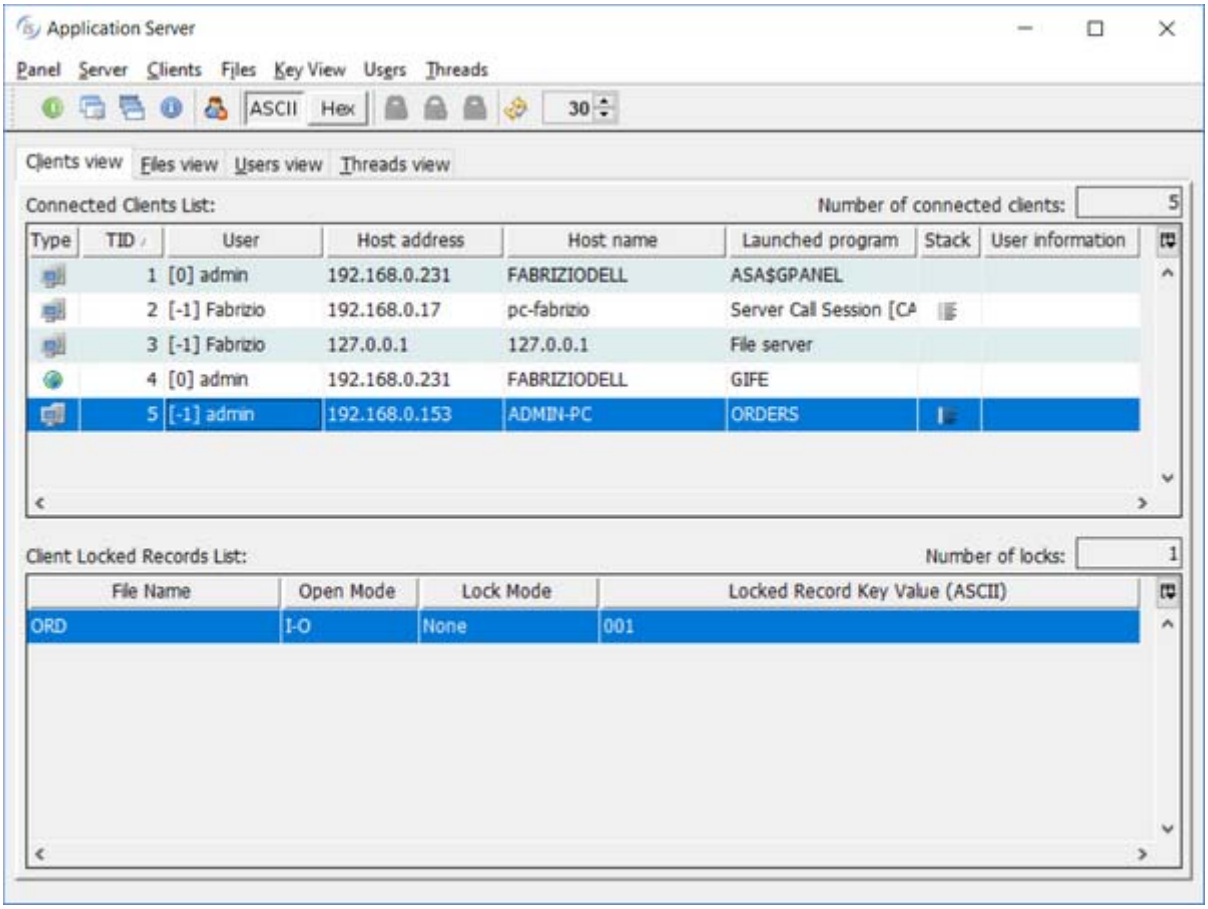
```
iscobol.file.lock_manager=com.iscobol.as.locking.InternalLockManager
```

isCOBOL Server can be either a iscserver process running as Application Server, a iscserver process running as File Server or a iscserver process running as both. In the third case, locks acquired by the Application Server clients are managed together with locks acquired by the File Server clients.

Note - For this feature to work correctly, the configuration property iscobol.as.multitasking must be either omitted or set to 0.

Making isCOBOL Server manage locks itself guarantees that all the lock clauses are supported, regardless of the file handler; this is particularly useful when working on databases via Database Bridge. Active locks can be monitored and managed through the server administration panel:

```
iscclient [-port port] [-hostname host] -panel
```



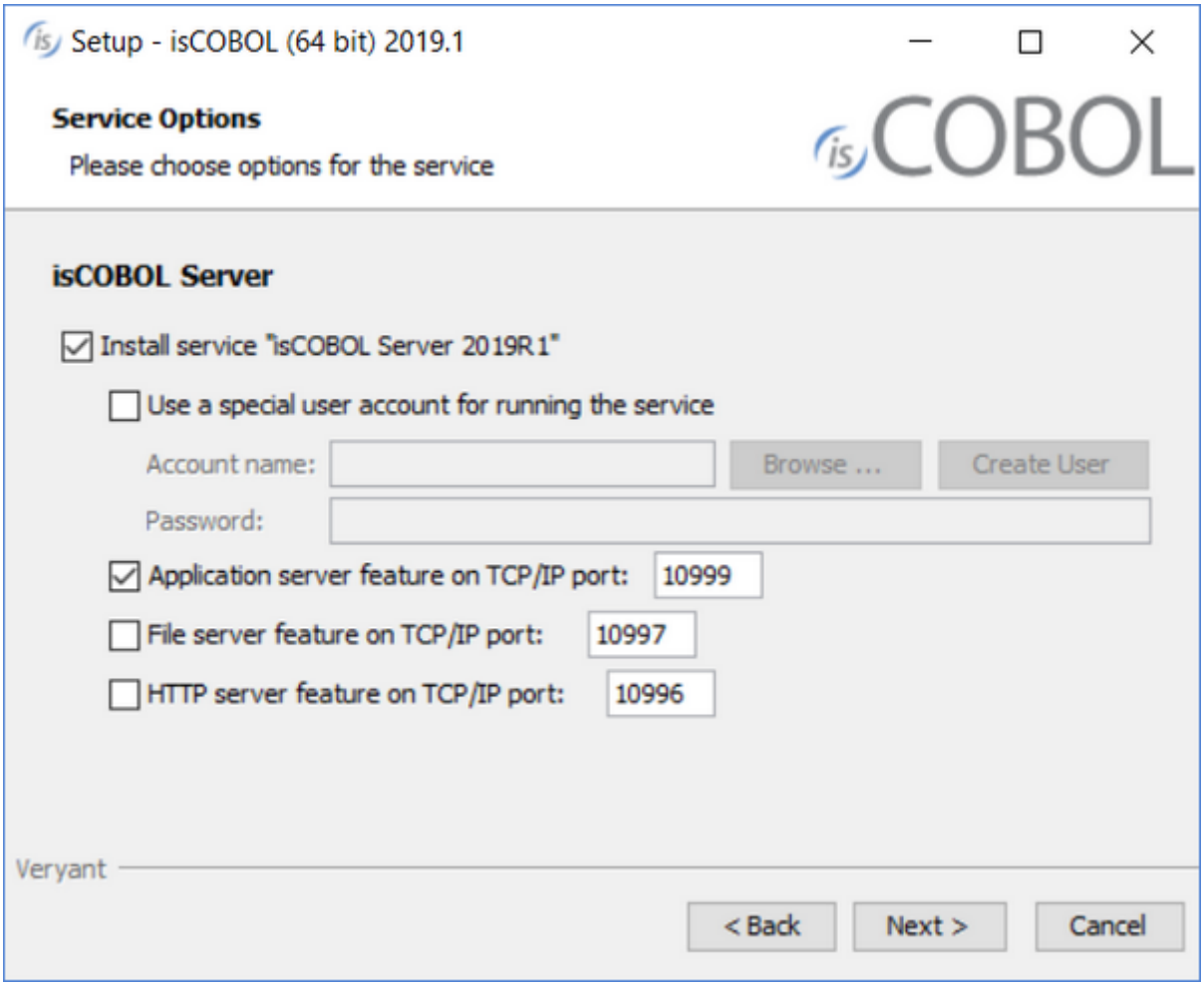
The only disadvantage is that locks are held only between the clients of the same isCOBOL Server and don't affect COBOL programs running outside of the isCOBOL Server as well as third party applications.

Windows service and Unix daemon

Windows service

On Windows it's possible to install isCOBOL Server as a Windows Service.

The isCOBOL Server service can be installed during the setup process:



When isCOBOL has been installed, the service can be installed, removed and managed through the isserver.exe command line utility.

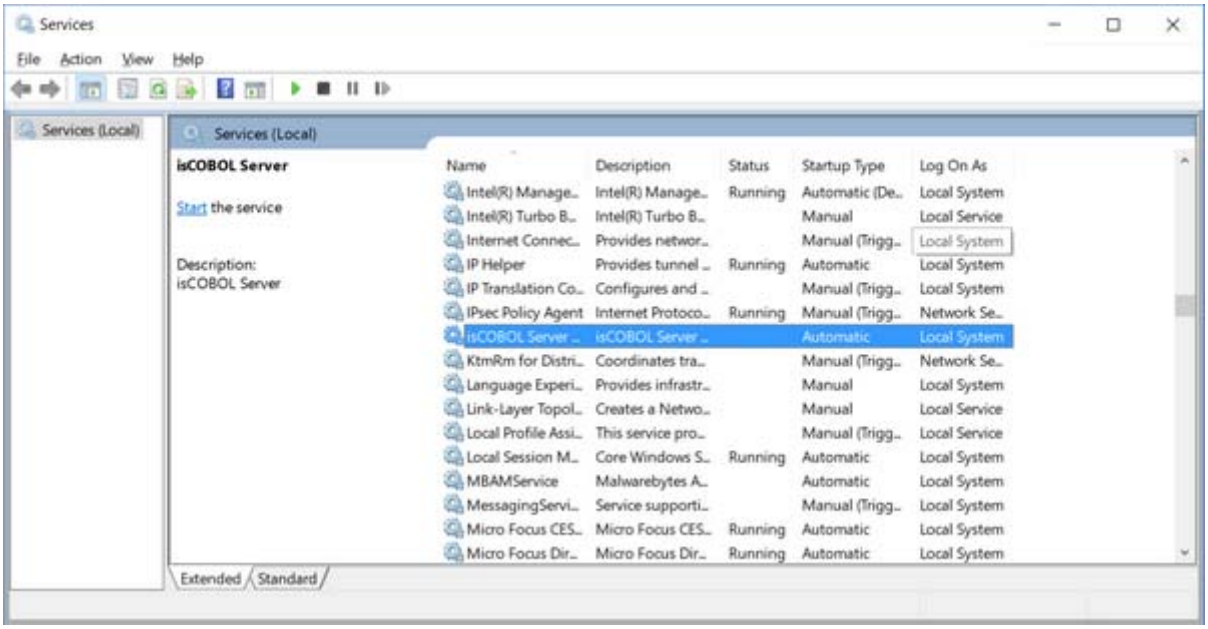
isserver.exe usage

The service maintenance is done through isserver.exe.

To install the service, use the command:

```
isserver -install
```

If the operation is successful, there will be a new entry in the Windows service manager.



The service is installed in auto mode, which means the service will automatically start along with the system.

To install the service in demand mode, use the command:

```
isserver -install-demand
```

In this mode, the service must be manually started by the user in the Windows service manager.

To retrieve the service status, use the command:

```
isserver -status
```

The exit code of this command is 0 when the service is running, 3 when it is not running and 1 when the state cannot be determined.

To start the service, use the command:

```
isserver -start
```

To stop the service, use the command:

```
isserver -stop
```

To uninstall the service, use the command:

```
isserver -uninstall
```

If the command is successful, the isCOBOL Server service will disappear from the Windows service manager.

In some situations, you might want to install a Windows service as a non-interactive service so that the service does not have any possibility to access the GUI subsystem. In order to do that, add the phrase non-interactive after the -install parameter. A custom service name can still be specified after the non-interactive parameter:

```
isserver -install non-interactive
```

It's also possible to specify a custom name for the service. This name should be added as last parameter of isserver.exe command line for all the options. For example, the following list of commands manges an isCOBOL Server service named "myservice":

```
isserver -install myservice
isserver -start myservice
isserver -status myservice
isserver -stop myservice
isserver -uninstall myservice
```

Output redirection

The isCOBOL Server service redirects all the console output (stderr and stdout) to two files named *isserver_err.log* and *isserver_out.log*. These files are located in the isCOBOL bin directory, which is the default directory of the service.

Service configuration

Java options must be put in the *isserver.vmoptions* file, located in the isCOBOL bin directory, which is the default directory of the service. In this file comments are prefixed by a hash and each option is on a separate line.

The following snippet shows how to configure memory limits, pass a custom configuration file and alter the Classpath for the isCOBOL Server service:

```
#memory settings
-Xmx256m
-Xms128m

#configuration
-Discobol.conf=/myapp/myconf

#classpath
-classpath/p .
-classpath/a C:\Program Files\Java\jdk1.8.0_161\lib\tools.jar
```

The isCOBOL Server service inherits the Classpath from the system and adds all jar libraries in the isCOBOL lib directory to it. Using the *-classpath* option you can add additional items to the active Classpath. The value of *-classpath/p* is prepended to the active Classpath. The value of *-classpath/a* is appended to the active Classpath.

Note: On some Windows distributions it's necessary to reboot the system in order to make services aware of modifications to the system environment.

isCOBOL configuration properties to configure port number, hostname, rundebug, etcetera, can be set either in *isserver.vmoptions* with the syntax "-Dproperty=value" or in a file named *iscobol.properties* that will be loaded from:

1. The \etc directory
2. The user home directory

3. The Classpath

Starting multiple services on the same machine

In order to manage multiple services on the same machine, the sc command must be used instead of the isserver command.

In this example we show how to start two isCOBOL Server services listening on different ports on the same machine.

1. Create two configuration files, e.g.

server1.properties:

```
iscobol.port=10999
iscobol.as.fileserver.port=10997
iscobol.as.httpserver.port=10996
```

server2.properties:

```
iscobol.port=10899
iscobol.as.fileserver.port=10897
iscobol.as.httpserver.port=10896
```

Note - the two snippets above shows the basic configuration, they should be completed with the rest of settings that usually appear in a isCOBOL runtime configuration. We notice that different ports for each isCOBOL Server feature are referenced in the two configuration files.

2. Create the two services as follows:

```
sc create "isCOBOL Server #1" start= auto binPath= "C:\Program
Files\Veryant\isCOBOL2019R1\bin\isserver.exe -c \path\to\server1.properties"

sc create "isCOBOL Server #2" start= auto binPath= "C:\Program
Files\Veryant\isCOBOL2019R1\bin\isserver.exe -c \path\to\server2.properties"
```

Unix daemon

On Unix systems, the isCOBOL Server can be installed as a daemon process and maintained using the isserver command.

isserver usage

The isserver command has the following options:

start	Run the isCOBOL Server service without keeping the console busy
stop	Stop the isCOBOL Server service
restart	Restart the isCOBOL Server service
status	Show the status of the isCOBOL Server service

You need to be root in order to use this command.

Daemon configuration

The isserver command looks for the file *default_java.conf* that is located in the isCOBOL bin directory.

This file is generated by the setup process and it includes the location of the isCOBOL SDK and the associated Java.

In this file comments are prefixed by a hash and each option is on a separate line.

Tuning and monitoring isCOBOL Server with JvisualVM

JvisualVM is a Java virtual machine monitoring, troubleshooting, and profiling tool.

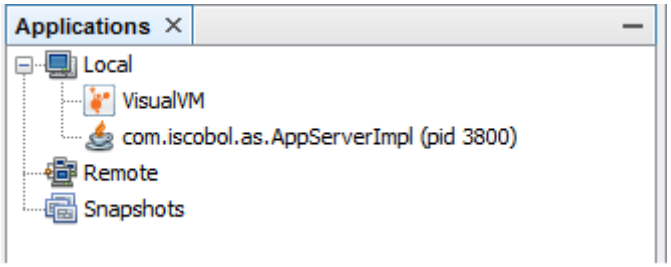
In Java version 8 and earlier, JvisualVM was installed along with the Java Development Kit (JDK). The JvisualVM executable file was found in the JDK bin directory. Since Java version 9, JvisualVM is no longer installed along with the JDK, but a bleeding-edge distribution of this tool can be downloaded from visualvm.github.io and installed separately.

In this chapter we're going to see how to monitor the isCOBOL Server activity and the load using JvisualVM.

Launching JvisualVM and attaching the isCOBOL Server process

JvisualVM is a Java tool itself. The Java Virtual Machine (JVM) behind this tool should be of the same version of the JDK or JRE you're going to attach and monitor. When using the tool installed along with the JDK, this rule is automatically respected. When using the alternative distribution from GitHub, you can configure the underlying JVM in the file *etc/visualvm.conf*. By default, the GitHub distribution of JvisualVM uses the default JVM in the system (the first java command found in the system Path).

When JvisualVM starts, it lists all the Java applications running on the local machine along with their PID. You should find the isCOBOL Server among these applications.



The name of the application changes depending on the command that you used to start the isCOBOL Server.

command	application name shown by JvisualVM
iscserver.exe (Windows)	C:.Program
Windows service	Local Application
iscserver (Linux/Unix)	com.iscobol.as.AppServerImpl
java com.iscobol.as.AppServerImpl	com.iscobol.as.AppServerImpl

In order to monitor the isCOBOL Server process, you have to attach it. Just double click on the application name in the tree or right click on it and choose "Open" from the pop-up menu.

Note - the process may not appear in the list if:

- the JDK version from which you started JvisualVM is different than the Java version used by isCOBOL Server
- JvisualVM was launched with different Administrator privileges than isCOBOL Server (Windows only).

Connecting to a remote isCOBOL Server process

Sometimes it's not possible to start JvisualVM on the same machine where isCOBOL Server is running. In this case you have to set up a remote JMX connection. The most common case is where the server is a Linux/Unix box where no graphical desktop is available, so in this guide we're going to see how to set up the JMX connection on Linux and how to attach to it from a Windows client. Assume that the IP address of the Linux server is 192.168.0.130.

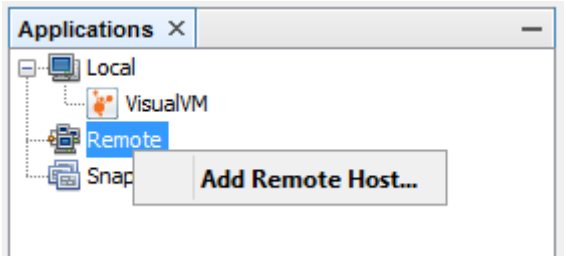
First of all, we need to dedicate a port to the JMX connection. In this example, we're using port 3333.

Change the isCOBOL Server startup command as follows:

```
iscserver -J-Dcom.sun.management.jmxremote.authenticate=false \
          -J-Dcom.sun.management.jmxremote.local.only=false \
          -J-Dcom.sun.management.jmxremote.port=3333 \
          -J-Dcom.sun.management.jmxremote.ssl=false \
          -J-Dcom.sun.management.jmxremote.rmi.port=3333 \
          -J-Djava.rmi.server.hostname=192.168.0.130
```

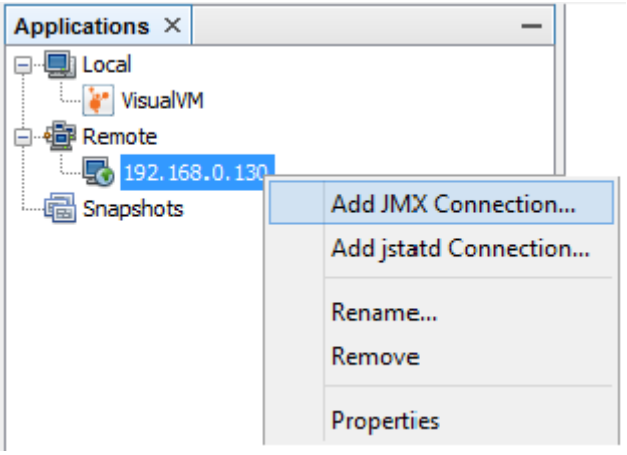
After the isCOBOL Server has been started,

- run JvisualVM on a Windows machine in the same network as your Linux server
- right click on "Remote" in the Applications tree and choose "Add Remote Host..."

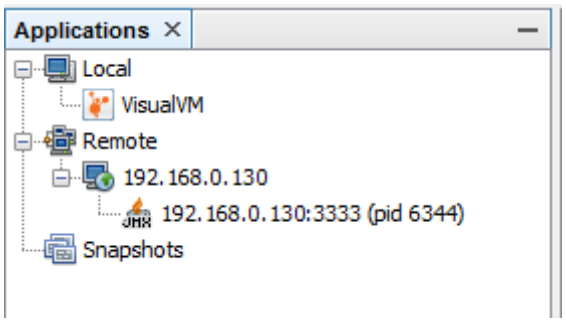


- fill the "Host name" field with the IP address of the Linux server and click OK

- the IP address appears as child item of Remote in the tree. Right click on it and choose "Add JMX Connection..."



- in the "Connection" field, put the port number 3333 after the colon and click OK.
- the isCOBOL Server application will appear as a new child in the tree:



In order to monitor the isCOBOL Server process, you have to attach it. Just double click on the application name in the tree or right click on it and choose "Open" from the pop-up menu.

JvisualVM's Monitor page

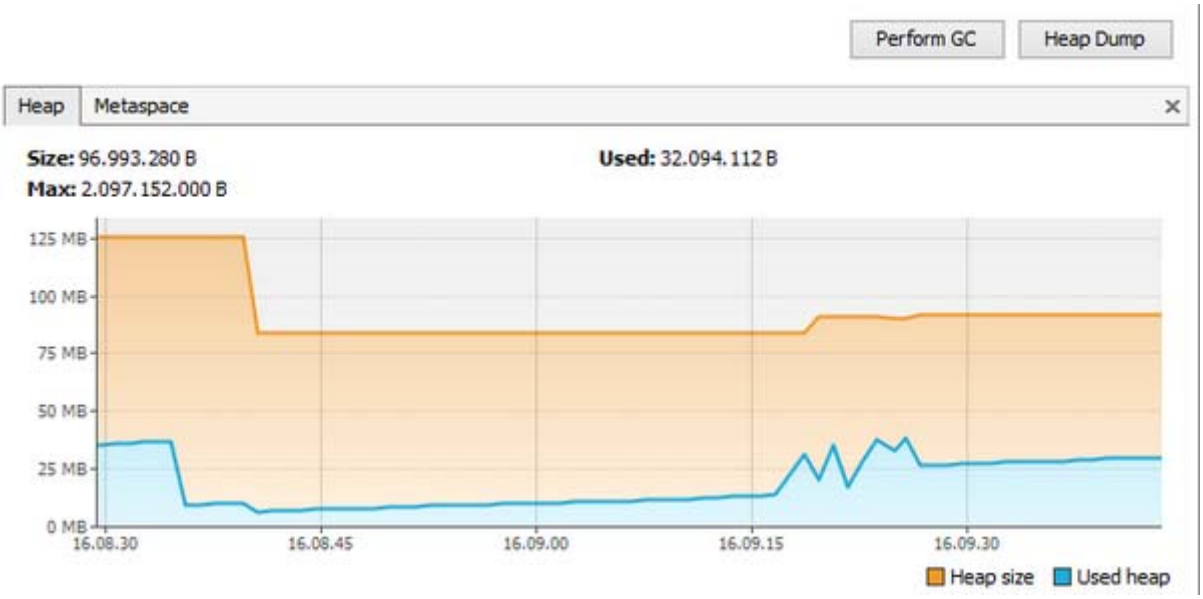
The Monitor page allows you to monitor the CPU and memory usage of the isCOBOL Server process in real time as well as the number of loaded classes and active threads.

You should pay particular attention to the Heap monitor.

The Heap is the amount of RAM memory that is allocated to store resources used by COBOL handles like bitmaps, fonts, ESQL cursors, etcetera. This memory has an initial size that can be configured through the -Xms Java option and a maximum size that can be configured through the -Xmx Java option. The Heap usage increases when the COBOL application loads a new resource and decreases when a garbage collection is performed.

The garbage collection is a procedure that takes care of removing inactive resources (e.g. a font or a bitmap whose handle has been destroyed) from the Heap memory. This procedure is automatically triggered by the JVM when the Heap usage is near the maximum amount of memory available or when the JVM is idle. You can force a garbage collection from the JvisualVM screen by clicking the "Perform GC" button. Be aware that this procedure consumes CPU and it may slow down connected clients that are working.

The "Heap Dump" button allows you to take a snapshot of the Heap in a given moment. By analyzing this snapshot you can find out which classes are using most of the memory.



JvisualVM's Threads page

The Threads page lists all the threads that were created in the isCOBOL Server's JVM. By default all threads are shown, including the ones that are not running anymore. You can filter the list using the "View" combo-box on top of the list.

The list includes a timeline that shows how the threads state changes as time goes by. There are five possible states:

State	Meaning
Running	The thread is running. In a COBOL application this is the typical state during an elaboration like a cycle that reads records from a file and processes them.
Sleeping	The thread is sleeping. In a COBOL application this is the typical state of a program that called the C$SLEEP routine and is waiting for the call to return.
Wait	The thread was blocked by a mutex or a barrier, and is waiting for another thread to release the lock. In a COBOL application this is the typical state of a program that is accepting user input with an ACCEPT statement.
Park	Parked threads are suspended until they are given a permit.
Monitor	The thread is waiting on a condition to become true to resume execution.

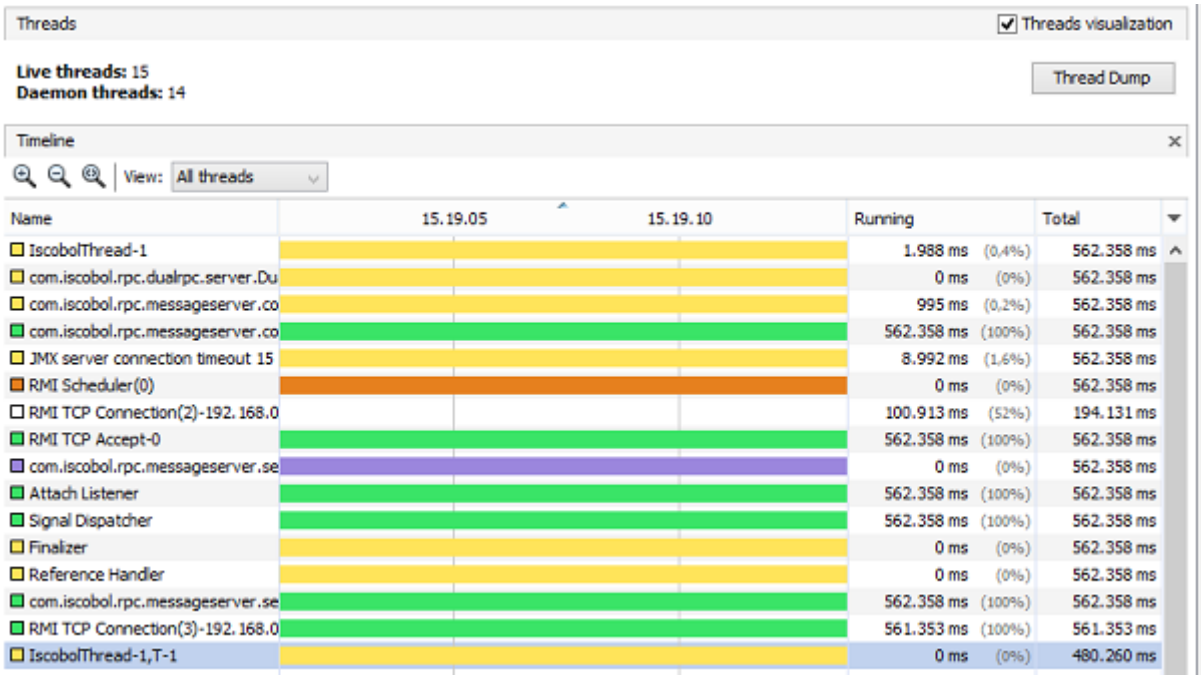
The isCOBOL Clients connected to the isCOBOL Server can be identified by their name:

IscobolThread-#	This is the main thread of the isCOBOL Client. # is the thread ID (TID) assigned to that Client. You can retrieve more information about the Client whose thread ID is # by running the isCOBOL Server's administration panel.

IscobolThread-#,T-#	This is a COBOL thread started from the COBOL application using either CALL THREAD or PERFORM THREAD statements. The first # is the TID of the Client, while the second # is a progressive number assigned by the isCOBOL Framework.
IscobolThread-#,R-#	This is a COBOL thread started from the COBOL application using the CALL RUN statement. The first # is the TID of the Client, while the second # is a progressive number assigned by the isCOBOL Framework.

An IscobolThread that stay in Running state without changing to Wait or Sleeping for a lot of time means that the Client may be in an infinite loop and it could slow down all the other connected Clients. You can take advantage of the administration panel in order to check which program and paragraph are being executed by that thread.

The "Thread Dump" button allows you to take a snapshot of the threads in a given moment. By analyzing this snapshot you can see the state and the stack of each thread.



Taking Heap and Thread dumps without JvisualVM

If neither JvisualVM nor a JMX connection is available on the machine where Java is running, it's still possible to take a Heap or a Thread dump by using command line tools installed in the JDK bin directory.

You just need to know the JVM process ID (PID) in the system. Use system tools to retrieve it, for example, on Linux/Unix you can use the ps command:

```
ps -ef |grep AppServerImpl
```

Once you know the PID of your JVM, use this command to take a Heap dump:

```
jmap -dump:live,format=b,file=/path/to/dump.hprof pid
```

To take a Thread dump, instead, use:

```
jstack pid > /path/to/dump.tdump
```

Measuring the load of your COBOL application

JvisualVM can help you calculate the amount of RAM that is necessary for your COBOL application to run in thin client mode with a given number of concurrent users.

Suppose that you're going to install your COBOL application on a production server where 100 users will connect and you need to suggest the amount of RAM that the server machine should provide. You can calculate it empirically in this way:

1. start the isCOBOL Server
2. attach isCOBOL Server with JvisualVM and take the current Heap usage (we'll call this value "A")
3. connect with a Client and move among the programs of your application simulating data elaboration, printing and other processes. Be sure to execute the most used functions of your application.
4. take the current Heap usage (we'll call this value "B").

Now you can make the following calculation: Necessary_RAM = A + ((B - A) * 100).

Looking for memory leaks

JvisualVM can help you find memory leaks.

The steps are similar to the ones described above:

1. start the isCOBOL Server
2. attach isCOBOL Server with JvisualVM and take the current Heap usage (we'll call this value "A")
3. connect with a Client and move among the programs of your application simulating data elaboration, printing and other processes. Be sure to execute the most used functions of your application.
4. exit from the application, the Client terminates.
5. force a garbage collection from JvisualVM
6. take the current Heap usage (we'll call this value "B").

If B is greater than A, then a memory leak is possible. You should repeat steps from 3 to 6 multiple times to see if the difference between A and B increases.

Tuning threads

A COBOL application generates a variable number of threads that run simultaneously. The number of threads generated depends on multiple factors. For example, threads may be created explicitly by the COBOL programs if they use PERFORM THREAD, CALL THREAD or CALL RUN statements, but threads may also be created internally by the runtime system to implement some features (e.g. the BEFORE TIME clause on the ACCEPT statements implies a thread that controls whenever the timeout expires).

JvisualVM can help you calculate the amount of concurrent threads generated by your COBOL application when running in thin client mode with a given number of concurrent users.

Suppose that you're going to install your COBOL application on a production server where 100 users will connect and you wish to know the amount of concurrent threads that will run. You can calculate it empirically in this way:

1. start the isCOBOL Server
2. attach isCOBOL Server with JvisualVM and take the current value of *Live peak* in the *Threads* area of the

Monitor page (we'll call this value "A")

3. connect with a Client and move among the programs of your application simulating data elaboration, printing and other processes. Be sure to execute the most used functions of your application.
4. take the current value of *Live peak* in the *Threads* area of the *Monitor* page (we'll call this value "B").

Now you can make the following calculation: Number_of_Threads = A + ((B - A) * 100).

If the number of threads is very high, consider reducing the memory that Java dedicates to the threads stack by using the -Xss option. This will allow you to save some memory.

```
iscserver -J-Xss128k other_options...
```

isCOBOL File Server

isCOBOL Server can be used as a file server in order to perform I/O operations on remote files.

isCOBOL File Server usage

The File Server daemon can be started with the following command:

```
iscserver -fs [-c config_file] [-fsport FSport] [-hostname host] [-as] [-port ASport]
[-force]
```

Setting iscobol.as.fileserver (boolean) to true in the configuration produces the same effect as using the -fs option while setting iscobol.as.appserver (boolean) to true in the configuration produces the same effect as using the -as option.

The following command starts the File Server on the local pc on the default port 10997:

```
iscserver -fs
```

A correct startup will produce an ouptut similar to this:

```
Application Server (file services) started and listening on port 10997
```

You can start both the Application Server and the File Server at the same time with the command:

```
iscserver -fs -as
```

Starting the Application Server in addition to the File Server allows you to connect the Administration Panel (see Format 5 of the isCOBOL Client command) and monitor the active connections to the File Server.

Both services can start on different ports than the defaults. Use the -fsport and -port options to control the port where the services will be listening. The following command, for example, starts the File Server on the port 1234 and the Application Server on the port 1235:

```
iscserver -fs -fsport 1234 -as -port 1235
```

The File Server host name and port can also be set in the configuration file, as shown below.

```
iscobol.hostname host
iscobol.as.fileserver.port port
```

Config_file should include the standard configuration, that is the same for every client. See Usage of isCOBOL Client for information about how to use a customized client configuration.

Client-side Configuration

There are two ways a program can use a remote file on the File Server.

1. By specifying server name and port in the file name with the ISF protocol, or
2. By using the "com.iscobol.io.DynamicRemote" class as file handler

The ISF protocol

It is possible to specify File Server connection information in the physical file name through the URL syntax as follows:

```
isf://hostname[:port]:path/to/file
```

Where

- *hostname* is the server name or IP address where the File Server is listening
- *port* is the port where the File Server is listening. If omitted, the port specified by iscobol.file.remote.port * (whose default value is 10997) is used
- *path/to/file* is the name of the remote file to open. It can be either a full path, a relative path or just the file name. If omitted, the root folder is assumed, so a path like "isf://localhost:" is equivalent to "isf://localhost:/".

The URL can be put entirely in the file name or can be built by combining the FILE-PREFIX setting and the file name.

When the FILE-PREFIX setting includes paths starting with "isf://", multiple paths must be separated by a line feed character, e.g.

```
iscobol.file.prefix=isf://192.168.0.1:/usr/data\nC:\\Temp
```

The above setting specifies two paths for the FILE-PREFIX:

1. the folder */usr/data* on the remote server whose IP is 192.168.0.1 where the File Server is listening
2. the local folder *C:\Temp*

The following code snippets show two ways to open FILE1 through the File Server listening on 192.168.0.1 on the default port. The File Server will search in the /usr/data folder on the server:

```
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FILE1 ASSIGN TO "isf://192.168.0.1:/usr/data/FILE1"
                  ORGANIZATION INDEXED
                  ACCESS DYNAMIC
                  RECORD KEY FILE1-KEY.

       FILE SECTION.
       FD FILE1.
       01 FILE1-RECORD.
          03 FILE1-KEY  PIC 9(3).
          03 FILE1-DATA PIC X(50).

       PROCEDURE DIVISION.
       MAIN.
           OPEN INPUT FILE1.
```

```
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FILE1 ASSIGN TO "FILE1"
                  ORGANIZATION INDEXED
                  ACCESS DYNAMIC
                  RECORD KEY FILE1-KEY.

       FILE SECTION.
       FD FILE1.
       01 FILE1-RECORD.
          03 FILE1-KEY  PIC 9(3).
          03 FILE1-DATA PIC X(50).

       PROCEDURE DIVISION.
       MAIN.
           SET ENVIRONMENT "file.prefix" TO "isf://192.168.0.1:/usr/data".
           OPEN INPUT FILE1.
```

The DynamicRemote class

In order to access remote files, client programs can assign their files to the "com.iscobol.io.DynamicRemote" class. In the configuration, it's also possible to use the alias "remote".

The class can be assigned in the SELECT statement. See FILE-CONTROL Paragraph, rule 35 for information about how to assign a file to a specific class.

For example, the following sequential file will be opened through the File Server, regardless of the file name that the program puts in the ARC-NAME variable:

```
SELECT ARC ASSIGN TO ARC-NAME
           ORGANIZATION LINE SEQUENTIAL
           CLASS "com.iscobol.io.DynamicRemote"
           .
```

The assignment can be done also through the following configuration properties:

- `iscobol.file.index` and `iscobol.file.index.FileName`
- iscobol.file.linesequential and iscobol.file.linesequential.FileName
- `iscobol.file.relative` and iscobol.file.relative.FileName
- `iscobol.file.sequential` and iscobol.file.sequential.FileName

For example, the following configuration entry causes all the indexed files to be opened through the File Server:

```
iscobol.file.index=remote
```

Client programs are made aware of the File Server location through the configuration properties iscobol.file.remote.host * and iscobol.file.remote.port *.

Since the full-path of the file is built client-side by the runtime before sending the i/o request to the File Server, `iscobol.file.prefix` must be set in the client configuration and must specify the directories on the server where files will be opened. If the server operating system uses a different file separator than the client, the property `iscobol.file.prefix_separator` must be set in the client configuration as well.

The following client configuration, for example, handles only indexed files remotely on the Linux server whose IP is 192.168.0.1 assuming that the File Server is listening on the default port:

```
iscobol.file.index=remote
iscobol.file.remote.host=192.168.0.1
iscobol.file.prefix=/usr/data
iscobol.file.prefix_separator=/
```

The following more complex sample configuration, instead, handles indexed, sequential and relative files remotely on the Linux server whose IP is 192.168.0.1 having the File Server listening on the port 12345:

```
iscobol.file.index=remote
iscobol.file.sequential=remote
iscobol.file.linesequential=remote
iscobol.file.relative=remote
iscobol.file.remote.host=192.168.0.1
iscobol.file.remote.port=12345
iscobol.file.prefix=/usr/data
iscobol.file.prefix_separator=/
```

User Authentication

If `iscobol.as.authentication *` is set to 2 in the server configuration, `iscobol.user.name` and iscobol.user.password must be set client side in order to specify login information.

Stored Procedures

In a File Server environment, COBOL subroutines can be called by remote programs to serve as stored procedures.

Calling stored procedures is permitted only after the connection to the File Server has been enstablished, that means after opening the first remote file.

The feature is provided through the StoreProcedure Class (com.iscobol.lib.StoreProcedure) internal class.

The local program calls the remote stored procedure using a code like the following:

```
       configuration section.
       repository.
           class sp as "com.iscobol.lib.StoreProcedure"
           .
       working-storage section.
       77  p1 pic x(128).
       77  p2 pic s9(9).
       77  p3 pic 9(5).
       77  rc pic s9(9).

       procedure division.

           set rc to sp:>call("remote-sub"):>input(p1)
                                           :>output(p2)
                                           :>inout(p3)
                                           :>end().
```

In this case three parameters are passed, one of each type.

If a subroutine needs to be called very often then better performance can be obtained by creating the object only once, for example:

```
       configuration section.
       repository.
           class sp as "com.iscobol.lib.StoreProcedure"
           .
       working-storage section.
       77  spo object reference sp.
       77  p1 pic x(128).
       77  p2 pic s9(9).
       77  p3 pic 9(5).
       77  rc pic s9(9).

       procedure division.

          set spo to sp:>call("remote-sub"):>input(p1)
                                          :>output(p2)
                                          :>inout(p3)
          perform until rc = 0
             set rc to spo:>end()
          end-perform.
```

The called subroutine on the server is a standard COBOL program that receives the parameters through the Linkage Section and optionally returns an exit status upon GOBACK.

```
       program-id. remote-sub.

       working-storage section.
      *> routine variables here

       linkage section.
       77  p1 pic x(128).
       77  p2 pic s9(9).
       77  p3 pic 9(5).

       procedure division using p1, p2, p3.
       main-logic.
      *> routine logic here
           goback.
```

isCOBOL Graphical Terminal

The isCOBOL Graphical Terminal, included with the isCOBOL Runtime Environment, enables users signing in with an SSH based terminal emulator such as PuTTY to launch graphical applications from the command line without an X Server or X Desktop client software. This provides applications with access to a user's environment variables and home directory, and the ability to call external programs (e.g. with CALL "SYSTEM") that can use the terminal emulator for stdin, stout and stderr.

The isCOBOL Graphical Terminal includes a client-side component, the isCOBOL Client Listener.

isCOBOL Client Listener

The isCOBOL Client Listener allows you to use a common SSH based terminal emulator with tunneling capabilities to run COBOL programs so that programs developed for UNIX environments with character terminals can be executed by the isCOBOL Framework taking advantage of the isCOBOL Server architecture.

The main advantages provided by this feature are:

1. Due to the thin client architecture, there is only one Java process on the server serving multiple clients. This allows you to save resources. Without the thin client technology, each client terminal would create a separate Java process on the server, wasting resources.
2. If the COBOL program calls programs written in other languages (such as C programs, for example) these programs can accept user input and display data on the same client terminal. It wouldn't be possible in a standard thin client architecture.

Note: Veryant recommends Putty as the proper SSH emulator to be used. It can be downloaded for free at http://www.chiark.greenend.org.uk/~sgtatham/putty/.

Configuring the server

The isCOBOL Client listener requires that an isCOBOL Server has been started and is listening on the UNIX server machine. See Usage of isCOBOL Server for information about how to start the isCOBOL Server.

In addition, on the UNIX server machine the following environment variable must be set:

```
ISCOBOL_DISPLAY=X
```

This configuration assumes that the server machine has the program "xauth" installed; this program is installed on all the UNIX/Linux machines that have an X interface.

Note: In order for the above settings to work correctly, other X11 services must be turned off. Otherwise you may experience unexpected behaviors.

If the X interface it's not present, save the following script in the file $HOME/.ssh/rc

```
if read proto cookie && [ -n "$DISPLAY" ]
then
   dispnum=`echo $DISPLAY | sed 's/\(.*\:\)\([0-9]*\)\(.*\)/\2/'`
   echo unix:$dispnum $proto $cookie > $HOME/.iscobol_xauth_$dispnum
fi
```

isCOBOL Client Listener usage

The isCOBOL Client listener is activated on the client machine with the following command:

```
iscclientd { [-displayport dport] } [-port port]
             [-x]
```

- *dport* is the port on the client machine that will be used to communicate between the UNIX shell and the Client Listener (by default it's 10998).
- *-x* is equivalent of -displayport 6000. The port 6000 is necessary to take advantage of the X11 Tunneling, explained in Configuring Putty to use isCOBOL Client Listener .
  Note: In order for the -x option to work correctly, any other X11 services must be turned off. Otherwise you may experience unexpected behaviors.
- *port* is the port used for connecting to the isCOBOL Server (by default it's 10999)

A correct startup will produce an output similar to this:

```
Starting client listener on hostname: localhost, on display port: 10998, AS port: 10999
```

The isCOBOL Client listener can be launched in the following alternate way to avoid keeping the console busy:

```
isclientd { [-displayport dport] } [-port port]
            [-x]
```

Once started, the isCOBOL Client listener is identified by an icon in the system tray.

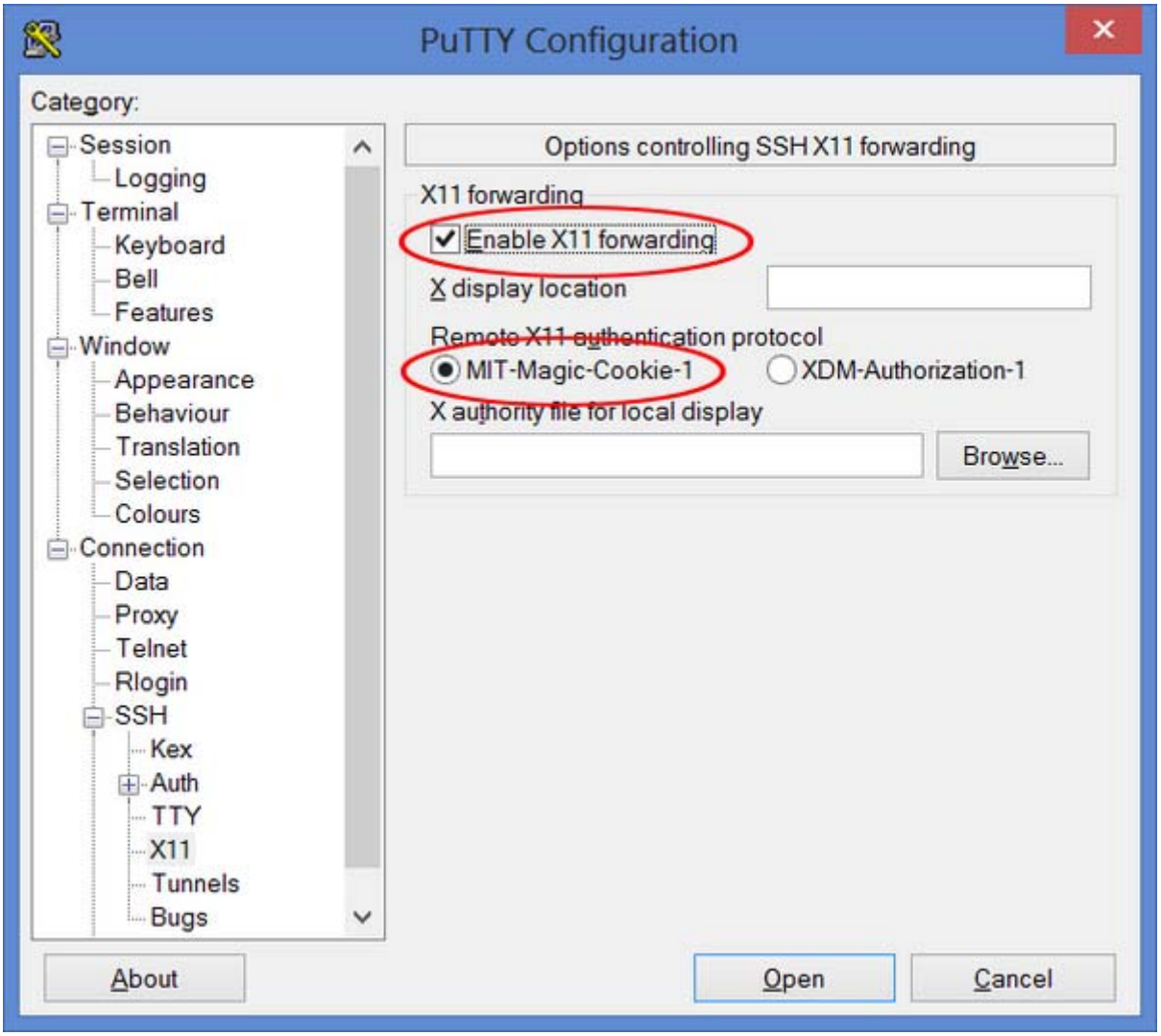
Right click on the icon to show a pop-up menu with the following options:

- **Info:** Display a balloon with the result of the Client Listener startup.

- **About:** Display a balloon with version information.
- **Exit:** Closes the Client Listener.

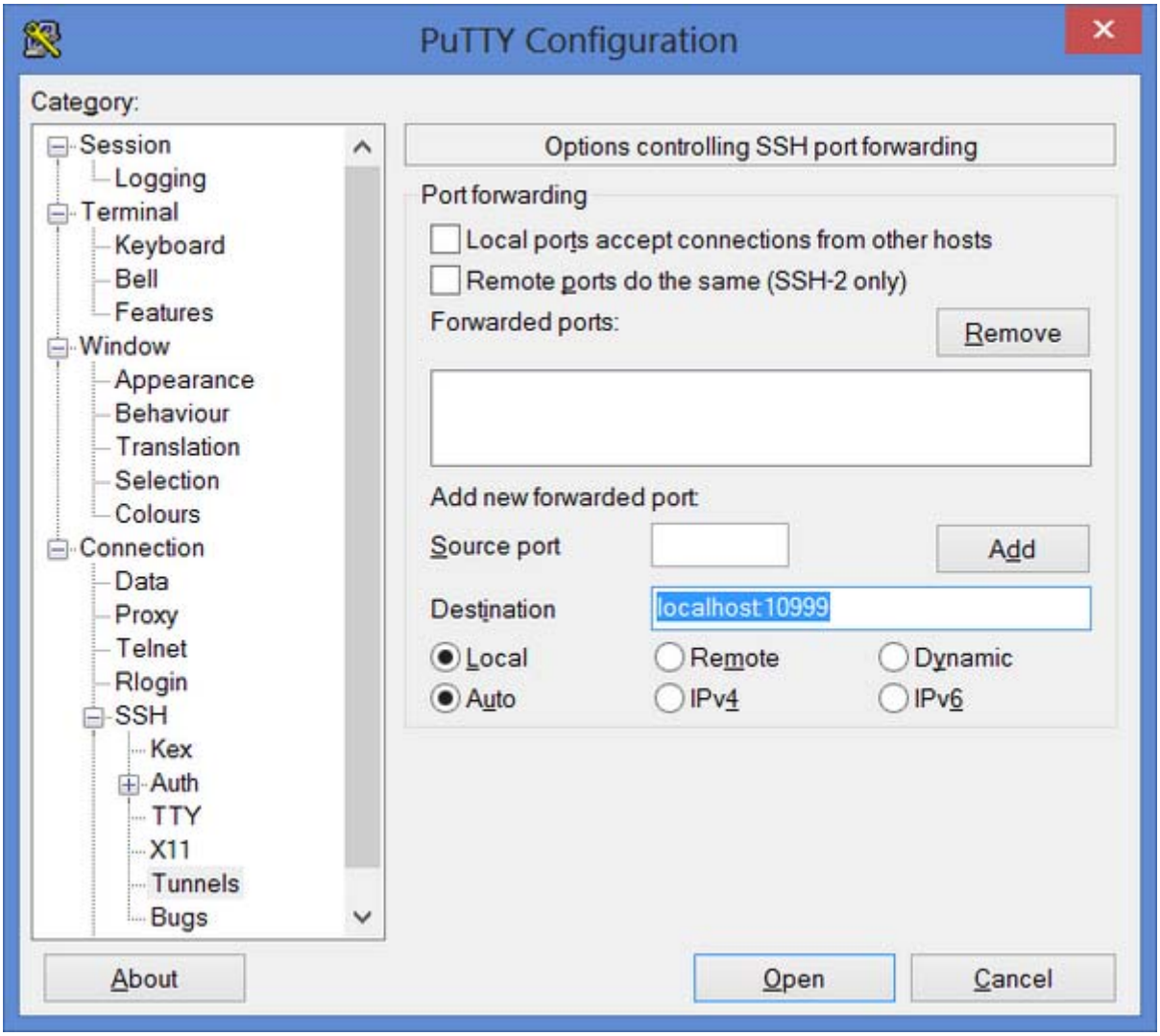
Configuring Putty to use isCOBOL Client Listener

Enable X11forwarding and choose MIT-Magic-Cookie-1 as the authentication protocol.

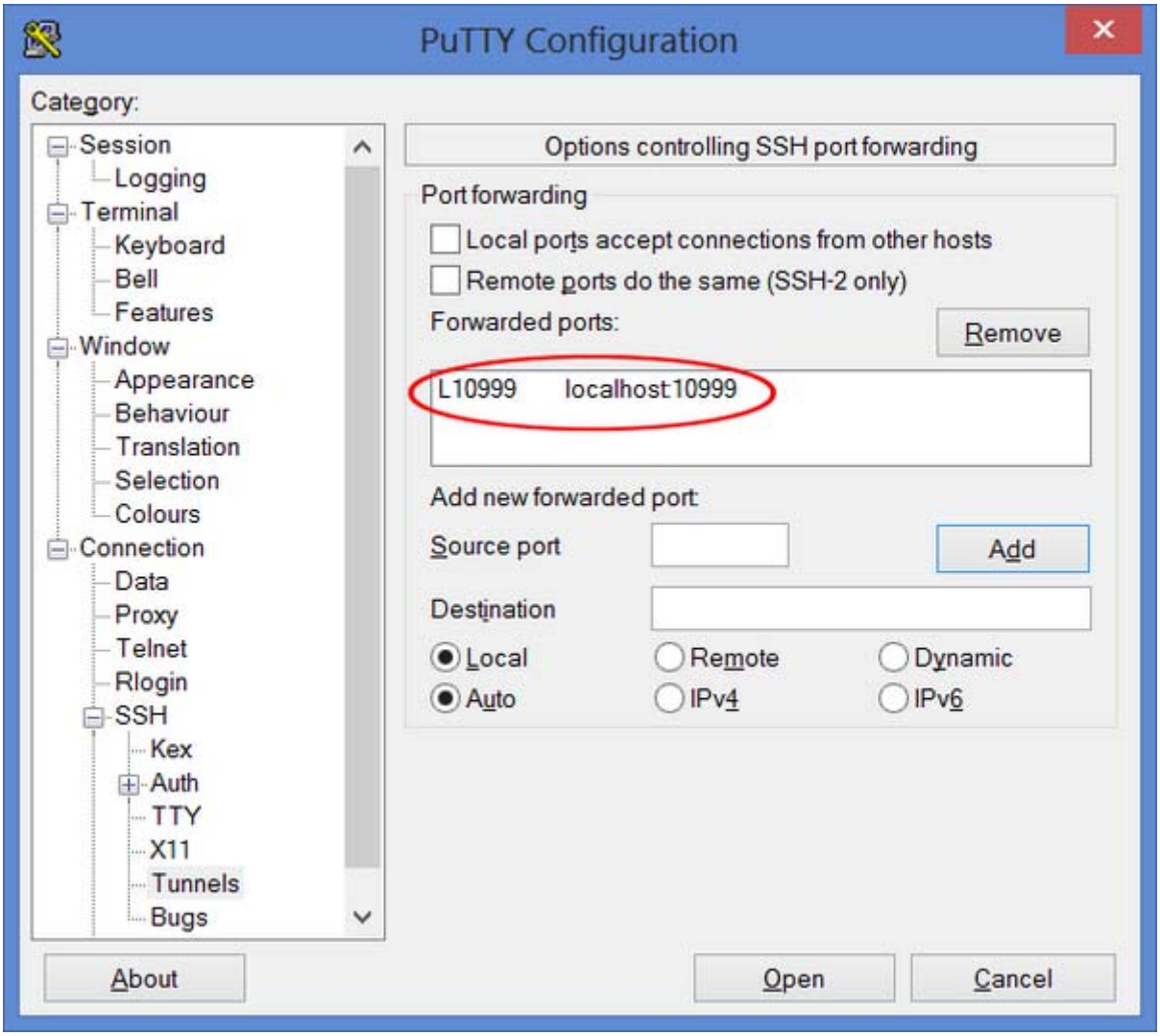


In the Tunnels configuration specify the hostname and port of the Client Listener (in the screenshot below, default values have been used).

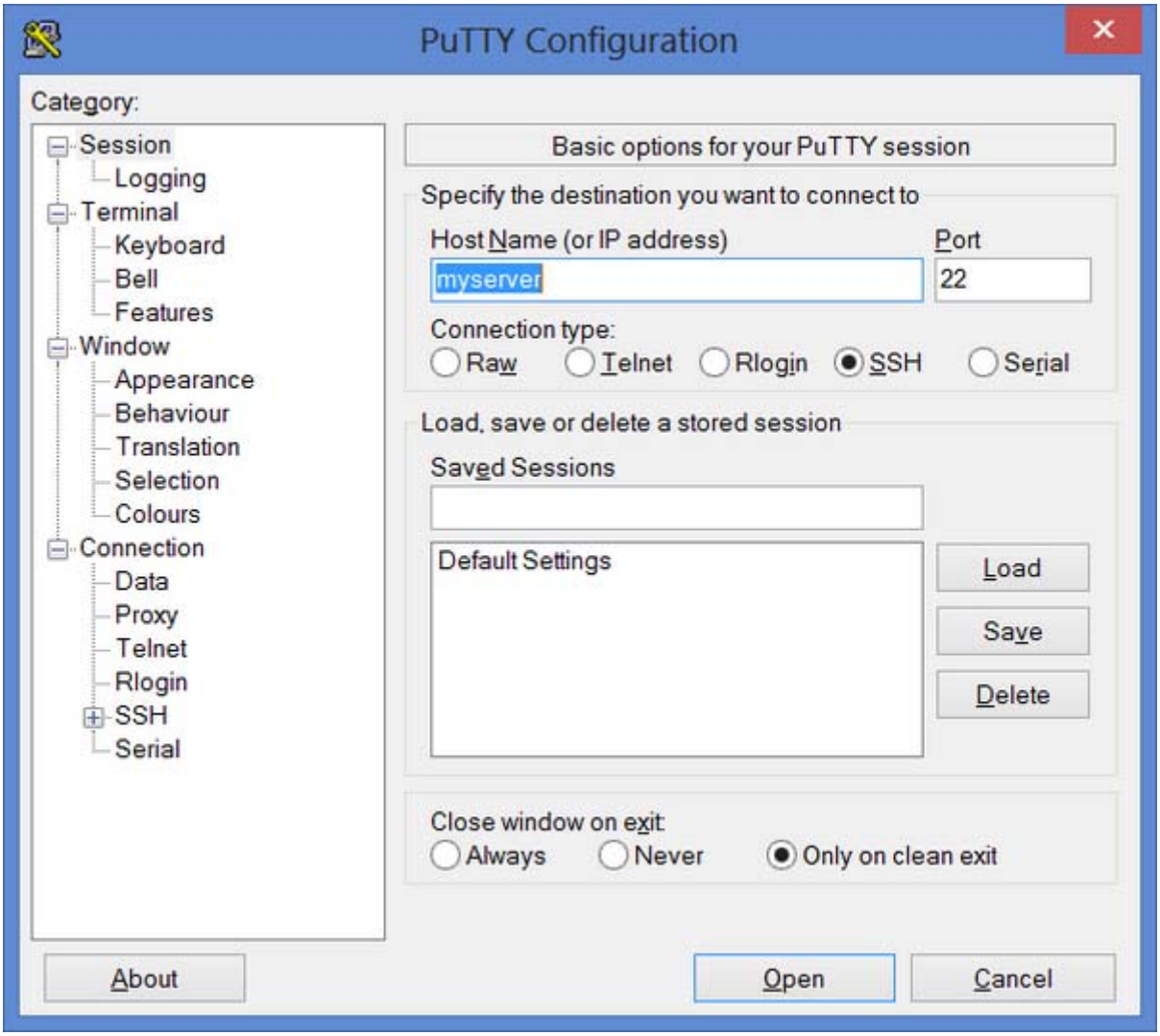
Note: In order for the above settings to work correctly, any other X11 services must be turned off. Otherwise you may experience unexpected behaviors.


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Kex
Auth
TTY
X11
Tunnels
Bugs
Options controlling SSH port forwarding
Port forwarding
Local ports accept connections from other hosts
Remote ports do the same (SSH-2 only)
Forwarded ports:
Remove
Add new forwarded port:
Source port
Add
Destination
localhost:10999
Local
Remote
Dynamic
Auto
IPv4
IPv6
About
Open
Cancel

Click on “Add”.



As a final step, configure the SSH connection, specifying the server name (or IP address) and choosing SSH as the connection protocol.



Connect to the server using the configured terminal emulator.

Ensure that the $ISCOBOL_DISPLAY environment variable is set to “X”. If it is not set, you can set it now.

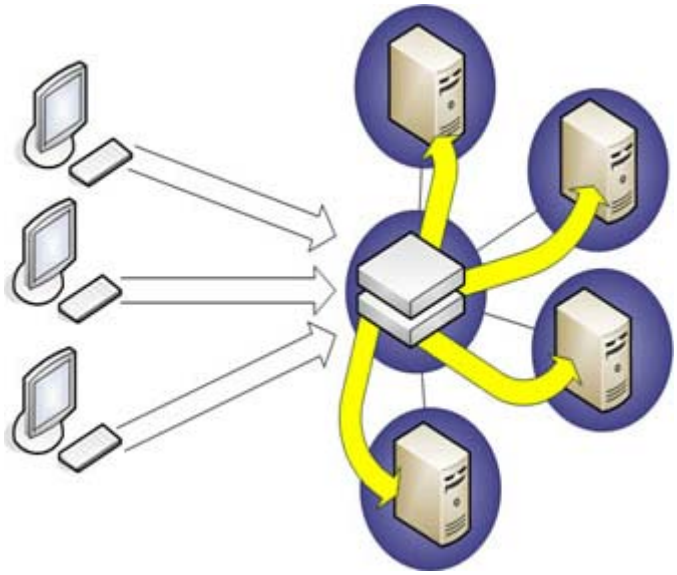
At this point you can run any kind of application using the Runtime Framework.

The iscrun command will launch a thin client instance and the program will be executed in a new window using the isCOBOL Server architecture.

When the COBOL program issues a SYSTEM call, the command will be executed in the terminal.

isCOBOL Load Balancer

isCOBOL Evolve includes a load balancing feature to distribute multiple client connections on different servers.



Once the isCOBOL Load Balancer is started, it waits for connections from isCOBOL Clients and when a connection request is performed, it evaluates the best server for satisfying the request, then it supplies the address of that server to the isCOBOL Client. From this moment, the isCOBOL Client communicates with the isCOBOL Server directly and the connection between the isCOBOL Client and the isCOBOL Load Balancer is closed, therefore shutting down the isCOBOL Load Balancer doesn't close the current connections.

The destination isCOBOL Servers are not started by the Load Balancer, they must be started separately. The startup order is not relevant. This means you can start the isCOBOL Load Balancer before or after the isCOBOL Servers.

Licensing

The isCOBOL Load Balancer is a separate product that requires its own license. The license code is specified using the following configuration property:

```
iscobol.balancer.license.2019=<license-code>
```

The property can be set exclusively in the isCOBOL Load Balancer configuration file. Refer to isCOBOL Load Balancer usage for information about how to pass a configuration file to the isCOBOL Load Balancer.

isCOBOL Load Balancer usage

The isCOBOL Load Balancer has the following usage:

```
iscbalancer [-port port] [-hostname host] [-force] configuration_file
```

While most of the command line options are optional, *configuration_file* is mandatory. This file is a Java property file whose entries are in the format *property=value*. The isCOBOL Load Balancer configuration is explained in Setting up the isCOBOL Load Balancer.

When a TCP connection is closed the connection may remain in a timeout state for a period of time after the connection is closed (typically known as the TIME_WAIT state or 2MSL wait state). For applications using a well known socket address or port it may not be possible to bind a socket to the required SocketAddress if there is a connection in the timeout state involving the socket address or port. Use the –force option to achieve it.

A correct startup will produce an output similar to this:

```
Load balancer started and listening on port 10999
```

Host name and port can also be set in the configuration file, as shown below.

```
iscobol.balancer.hostname host
iscobol.balancer.port port
```

Setting up the isCOBOL Load Balancer

In order to provide a list of available servers to isCOBOL Load Balancer, you need to create different entries in the configuration file. The entries have the following format:

```
iscobol.balancer.server.<id>=<host-name>:<port>,<number-of-users>
```

Where:

id	A numeric unique value that identifies the server. Usually servers are numbered from 1 by 1.
host-name	The name or IP address of the machine where the isCOBOL Server is listening (default: localhost).
port	The port where the isCOBOL Server is listening (default: 10999).
number-of-users	The maximum number of users allowed by the server (default: 10). It should be set to a value equal to or less than the maximum number of users allowed by the isCOBOL Server license.

The following configuration, for example, describes three different isCOBOL Servers listening on three different server machines on the default port and allowing at best 100 users each:

```
iscobol.balancer.server.1=192.168.0.101,100
iscobol.balancer.server.2=192.168.0.102,100
iscobol.balancer.server.3=192.168.0.103,100
```

When the isCOBOL Load Balancer is required to provide a server address to a new client, it chooses between the listed servers using the one with the lowest rate (currently-connected-users / maximum-number-of-users).

Automatic Server Checking

The isCOBOL Load Balancer checks the connections to the configured servers at regular intervals. The interval is 60 seconds by default but the number of seconds can be configured through the iscobol.balancer.update.interval property. Consider that the interval is applied to each single server, so, for example, if we have three servers listed in the configuration and we leave the interval at its default value (60 seconds), then 3 minutes will be necessary to check the connectivity to all the servers.

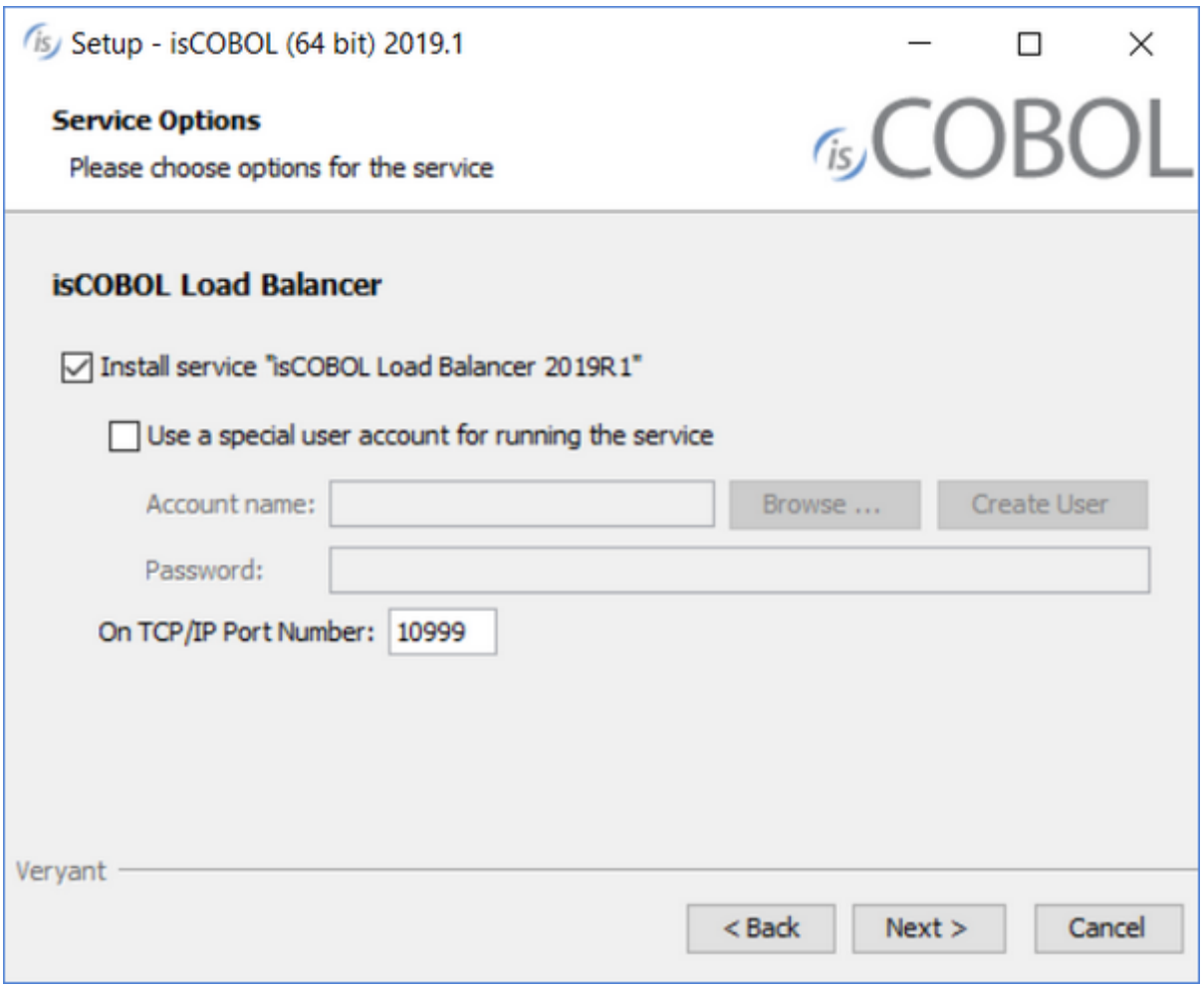
When a server is checked, there is a default connection timeout of 60 seconds. If there is no response before the timeout, the server is considered unavailable. The amount of seconds for a timeout can be configured through the iscobol.balancer.update.timeout property.

Windows Service and Unix daemon

Windows Service

On Windows it's possible to install isCOBOL Load Balancer as a Windows Service.

The isCOBOL Load Balancer service can be installed during the setup process:



When isCOBOL has been installed, the service can be installed, removed and managed through isbalancer.exe command line utility.

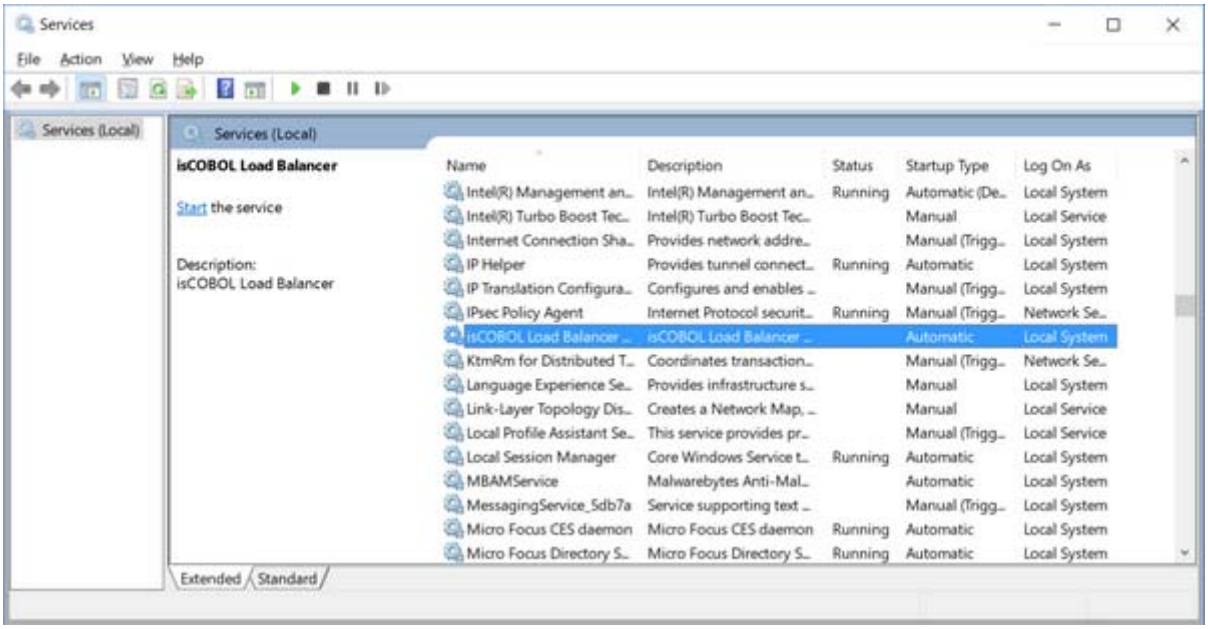
isbalancer.exe Usage

The service maintenance is done through isbalancer.exe.

To install the service, use the command:

```
isbalancer -install
```

If the operation is successful, there will be a new entry in the Windows service manager.



The service is installed in auto mode, which means the service will automatically start along with the system.

To install the service in demand mode, use the command:

```
isbalancer -install-demand
```

In this mode, the service should be manually started by the user in the Windows service manager.

To retrieve the service status, use the command:

```
isbalancer -status
```

The exit code of this command is 0 when the service is running, 3 when it is not running and 1 when the state cannot be determined.

To start the service, use the command:

```
isbalancer -start
```

To stop the service, use the command:

```
isbalancer -stop
```

To uninstall the service, use the command:

```
isbalancer -uninstall
```

If the command is successful, the isCOBOL Load Balancer service will disappear from the Windows service manager.

In some situations, you might want to install a Windows service as a non-interactive service so that the service does not have any possibility to access the GUI subsystem. In order to do that, add non-interactive after the -install parameter. A custom service name can still be specified after the non-interactive parameter:

```
isbalancer -install non-interactive
```

It's also possible to specify a custom name for the service. This name should be added as the last parameter of the isbalancer.exe command line for all the options. For example, the following list of commands manges an isCOBOL Load Balancer service named "myservice":

```
isbalancer -install myservice
isbalancer -start myservice
isbalancer -status myservice
isbalancer -stop myservice
isbalancer -uninstall myservice
```

Load Balancer configuration

The isCOBOL Load Balancer service reads a configuration file named *isbalancer.properties* stored in the bin directory of isCOBOL. Ensure that a valid license code and at least one server description are present in this file before starting the service.

Output redirection

The isCOBOL Load Balancer service redirects all the console output (stderr and stdout) to two files named *isbalancer_err.log* and *isbalancer_out.log*. These files are located in the isCOBOL bin directory, which is the default directory of the service.

Service configuration

Java options should be put in the *isbalancer.vmoptions* file, located in the isCOBOL bin directory, which is the default directory of the service. In this file comments are prefixed by a hash and each option is on a separate line.

Setting the Classpath in the *isbalancer.vmoptions* has no effect. Every occurrence of -cp and -classpath in that file is discarded. The isCOBOL Load Balancer service inherits the Classpath from the system and adds all jar libraries from the isCOBOL lib directory to it.

Note: On some Windows distributions it's necessary to reboot the system in order to make services aware of modifications to the system environment.

Unix daemon

On Unix systems, the isCOBOL Load Balancer can be installed as a daemon process and maintained using the isbalancer command.

isbalancer usage

The isbalancer command has the following options:

start	Run the isCOBOL Load Balancer service without keeping the console busy
stop	Stop the isCOBOL Load Balancer service
restart	Restart the isCOBOL Load Balancer service
status	Show the status of the isCOBOL Load Balancer service

You need to be root in order to use this command.

Daemon configuration

The isbalancer command looks for the file *default_java.conf* that is located in the isCOBOL bin directory.

This file is generated by the setup process and it includes the location of the isCOBOL SDK and the associated Java.

In this file comments are prefixed by a hash and each option is on a separate line.