isCOBOL Evolve: WebClient

Thin Client inside a web browser

Key Topics:

- Testing the product using the isCOBOL Demo program (Iscontrolset)
- Applications Monitoring and Configuration
- Known limitations and differences between WebClient and Thin Client

veryant

Introduction

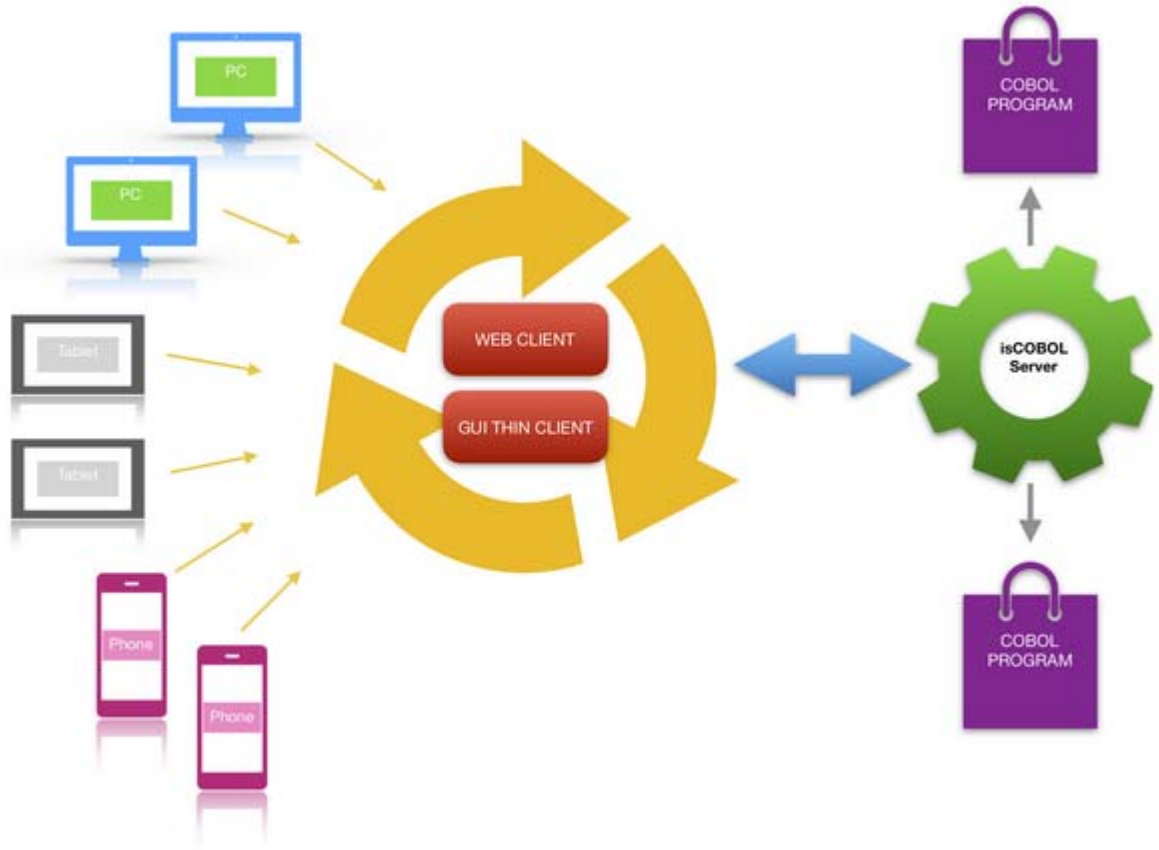
With isCOBOL WebClient your organization can leverage existing COBOL syntax to develop and deploy a universally accessible, zero client, rich Internet application (RIA) using standard COBOL screen sections and existing program procedure flow. No knowledge of object-oriented programming, JavaScript, HTML, or other Web languages is required.

isCOBOL WebClient is an HTTP server that reproduces the GUI of the isCOBOL Thin Client in a web browser.

The WebClient's HTTP server runs on a computer that can access the isCOBOL Application Server, just as the isCOBOL Thin Client does. This machine then becomes the 'web server' for users connecting to your COBOL application from a web browser.

Every time a new session of the COBOL application is launched from a web browser, a new isCOBOL Client (hence a new JVM) is instantiated on the web server.

Note - Applications running in WebClient are run by the server, and only rendered images are sent to the browser. There is no HTML involved, therefore it's not possible to use CSS to customize the layout of the applications.



This WebClient architecture brings some notable capabilities:

- User can interact with the application as if it were a regular desktop application.
- Users can re-establish their sessions after a lost connection with session persistence.
- Administrators can monitor running applications in real time, viewing important information such as memory usage, CPU usage and response times.

- Administrators can provide assistance to end users by using the built-in remote assistance feature, which mirrors the user's screen on the WebClient administrative console, and allows administrator to take control of the session and help the user accomplish a task or troubleshoot a problem.
- The WebClient includes an administration web console you can use to configure users, isCOBOL programs, and a wide variety of customizations and settings.

Installation Environment

isCOBOL WebClient is based on WebSwing technology.

isCOBOL WebClient works only with Java 8 and Java 11. Other Java versions are currently not supported.

isCOBOL WebClient is available for Windows and Linux platforms.

The product is provided and supported only for the 64 bit architecture.

In order to run on Linux, WebClient requires the X virtual framebuffer (Xvfb) or a X Window System (X11).

Getting Started

The setup of a WebClient environment requires the following steps:

1. Download and install the Java Runtime Environment (JRE)
2. Download and install isCOBOL Evolve SDK 64-bit
3. Activate the License
4. Set the Secret Key

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 the Java Runtime Environment (JRE)

JRE version 8 or 11 must be installed on your machine in order to use isCOBOL WebClient. For best results and performance, install the latest update build available for your platform.

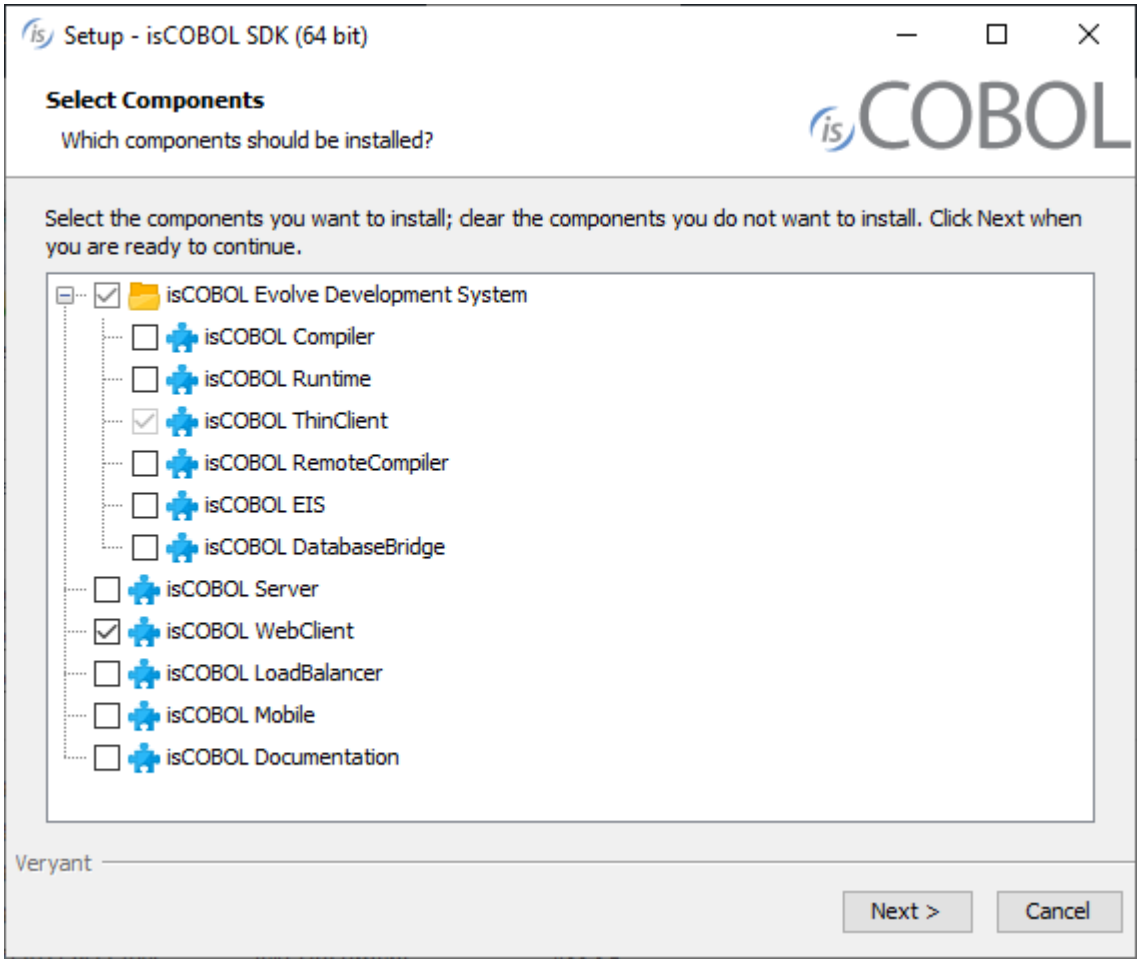
Self-extracting setups are provided for the Windows platform

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).

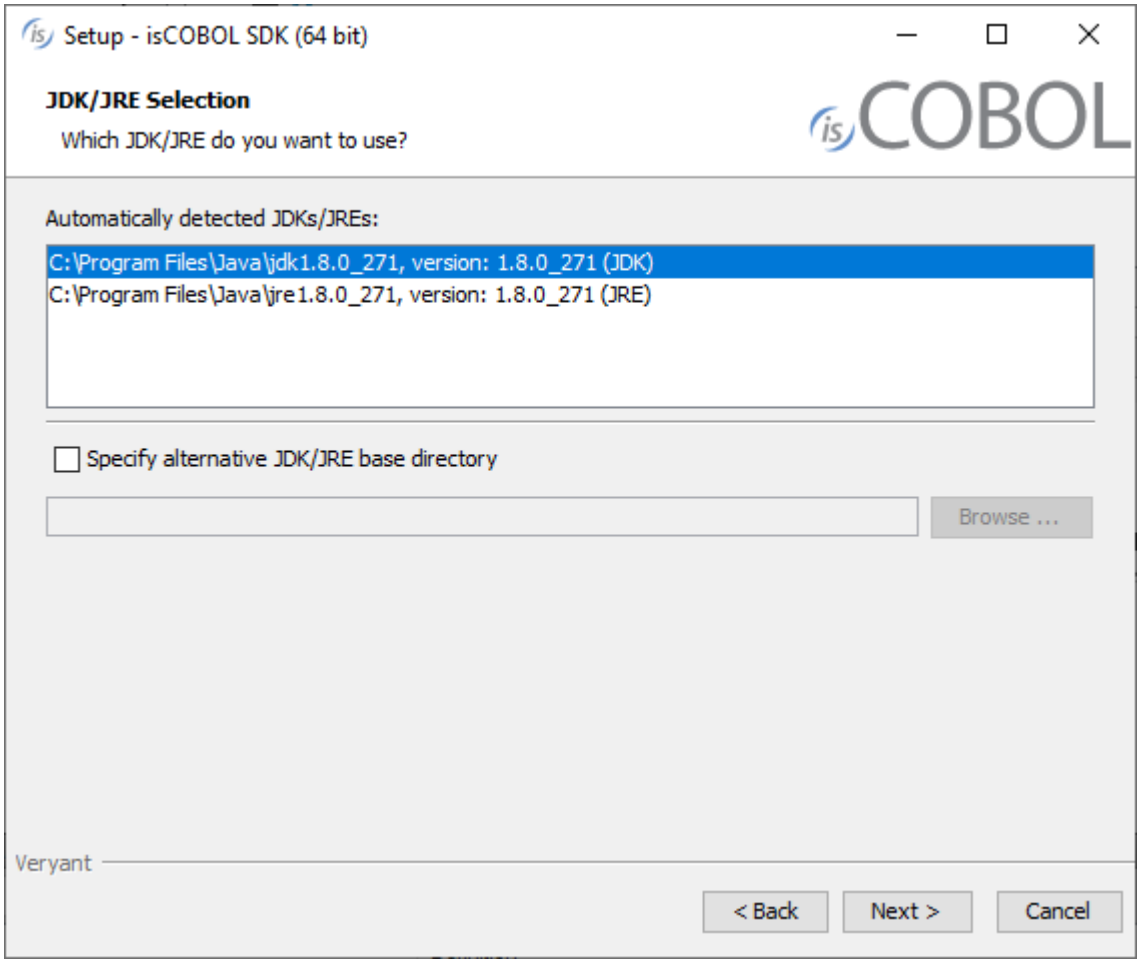
Download and install isCOBOL Evolve SDK 64-bit

Windows

1. If you haven't already done so, Download and install the Java Runtime Environment (JRE).
2. Go to "https://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. Select isCOBOL_2021_R1_Windows.64.msi, where *n* is the build number.
6. Run the downloaded installer to install the files.
7. Select the desired items from the list of products when prompted.



8. Select your JDK/JRE when prompted



9. Follow the wizard procedure to the end. In the process you will be asked to provide the installation path ("C:\Veryant" by default) and license keys. You can skip license activation and perform it later, as explained in Activate the License.

Linux

1. If you haven't already done so, Download and install the Java Runtime Environment (JRE) .
2. Go to "https://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 Linux 64 bit platform.
6. Extract all contents of the archive:

```
gunzip isCOBOL_2021_R1_*_Linux.64.x86_64.tar.gz
tar -xvf isCOBOL_2021_R1_*_Linux.64.x86_64.tar
```

7. Change to the "isCOBOL2021R1" folder and run "./setup", you will obtain the following output:

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

                          isCOBOL EVOLVE Installation
                          For isCOBOL Release 2021R1
                      Copyright (c) 2005 - 2021 Veryant

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

Install Components:

    [0] All products..................................... (no)
    [1] isCOBOL Compiler (includes [2] & [3])............ (yes)
    [2] isCOBOL Runtime (includes [3])................... (no)
    [3] isCOBOL ThinClient............................... (no)
    [4] isCOBOL RemoteCompiler........................... (no)
    [5] isCOBOL EIS...................................... (no)
    [6] isCOBOL DatabaseBridge........................... (no)
    [7] isCOBOL Server................................... (no)
    [8] isCOBOL WebClient................................ (no)
    [9] isCOBOL LoadBalancer............................. (no)
   [10] 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 "8", then press Enter to select isCOBOL WebClient.
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

The WebClient product is qualified only for the Linux platform. Despite the platform independent includes it, there's no guarantee that it will work correctly on other Unix platforms.

Distribution Files

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

Activate the License

WebClient doesn't require a specific license in order to start, same as the standard isCOBOL Client.

The licensing is managed by the isCOBOL Server.

Refer to Activate the License in isCOBOL Server's documentation for more information.

Set the Secret Key

WebClient and WebClient Admin Console come with a default secret key set to the text "change this in production" followed by a series of zeroes. The secret key is defined in the file *webclient/webclient.properties* for WebClient and in the file *webclient/admin/webclient-admin.properties* for the Admin Console. The default setting is:

```
webclient.connection.secret =
change_this_in_production_0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
```

You should change this value to a value of your choice. The value must be 128 characters long.

On Windows, if you install WebClient as a service, the setup will generate a random key for you.

It's important that WebClient and WebClient Admin Console use the same secret key, otherwise they will not be able to communicate.

Testing the product using the isCOBOL Demo program (Iscontrolset)

In this guide we're going to run the isCOBOL Demo program (Iscontrolset) in a web browser through isCOBOL WebClient.

Before you start the WebClient service, it's good practice to ensure that the isCOBOL Thin Client can execute the program correctly. So, ensure that this command opens the isCOBOL Demo program:

```
iscclient -hostname <yourAppServerNameOrIp> ISCONTROLSET
```

The above command assumes that there is an isCOBOL Server running on the machine identified by *yourAppServerNameOrIp* and the folder containing ISCONTROLSET.class is in the Server's Classpath or code-prefix.

See isCOBOL Evolve: Application Server for more information about how to start programs in a thin client environment.

Once the above command works correctly, you can start the WebClient services.

Note - If you wish to connect to the web application from different machines in the network in addition to your PC, then the service must be listening on the IP address of the PC instead of localhost. You can configure this setting in Jetty Configuration.

Use the following command to start the WebClient service:

- Windows

```
cd %ISCOBOL%\bin
webcclient.exe
```

- Linux

```
cd $ISCOBOL/bin
./webcclient
```

A correct startup shows a message like this at the bottom of the console output:

```
INFO:oejs.Server:main: Started @3791ms
```

Use the following command to start the WebClient Admin Console service:

- Windows

```
cd %ISCOBOL%\bin
webcclient-admin.exe
```

- Linux

```
cd $ISCOBOL/bin
./webcclient-admin
```

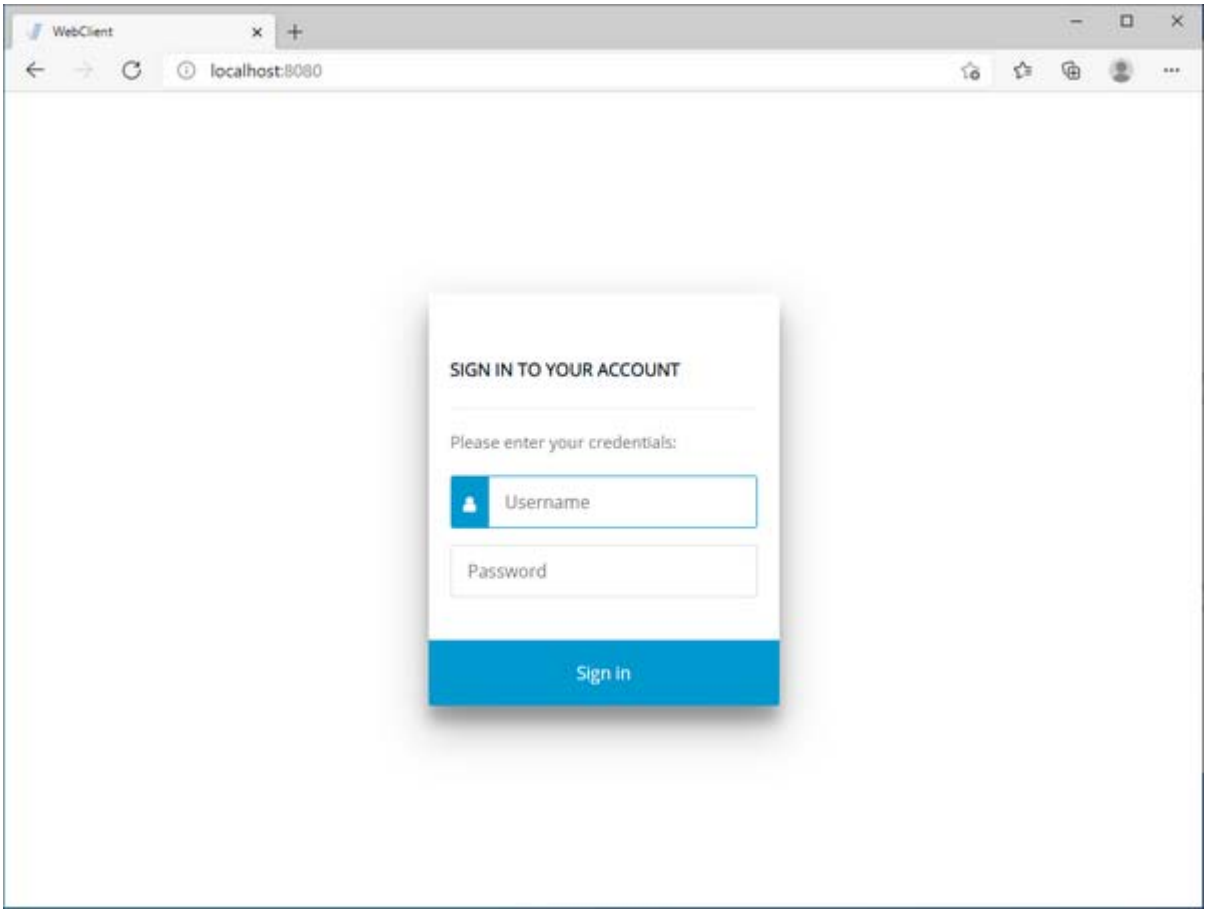
A correct startup shows a message like this at the bottom of the console output:

```
INFO:oejs.Server:main: Started @4082ms
```

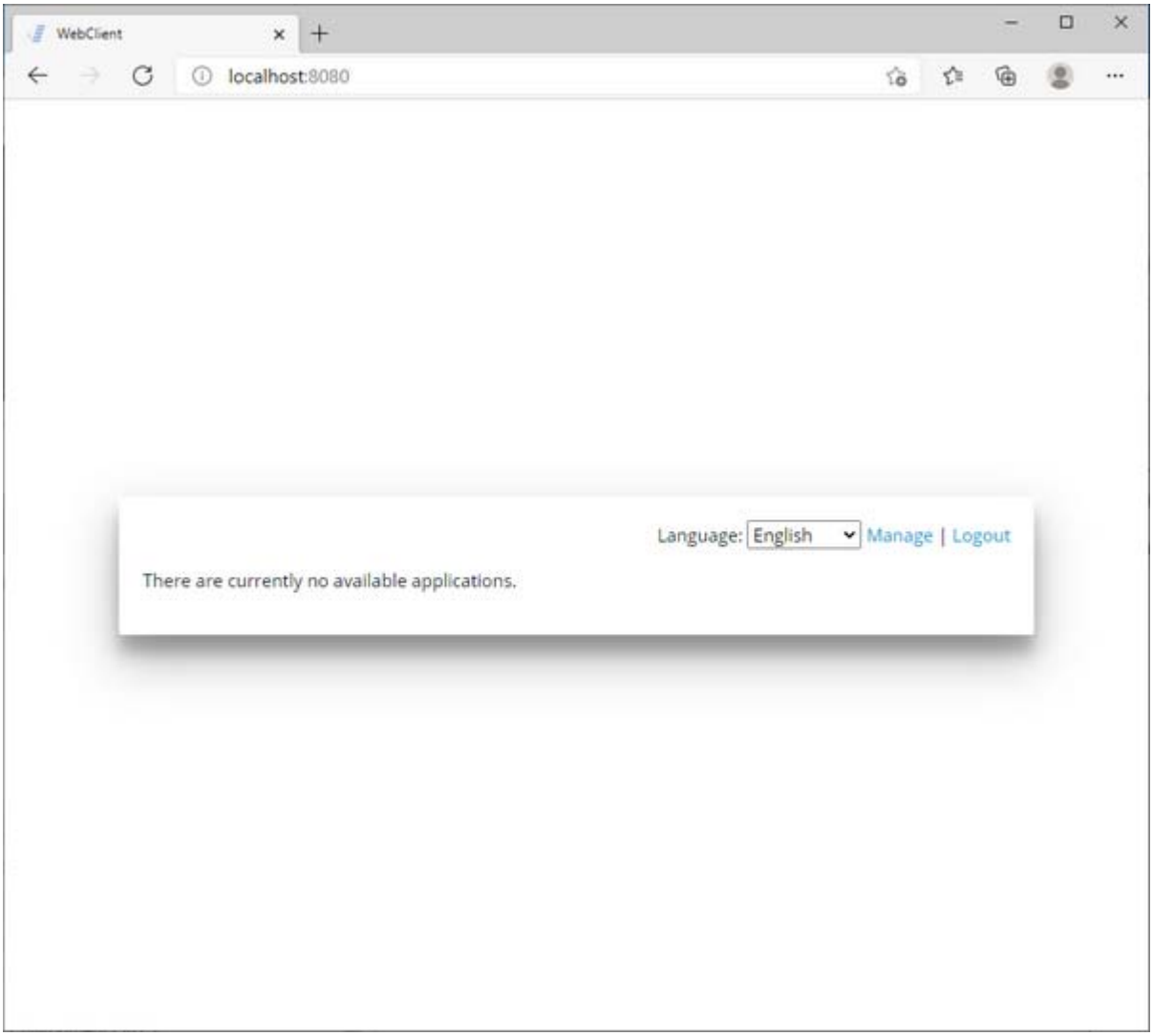
When both the WebClient service and the WebClient Admin Console service are up and running, you can navigate to http://*machine-ip*:*port* with a web browser.

Note - *machine-ip* and *port* must match the *server.host* and *server.http.port* values set Jetty Configuration.

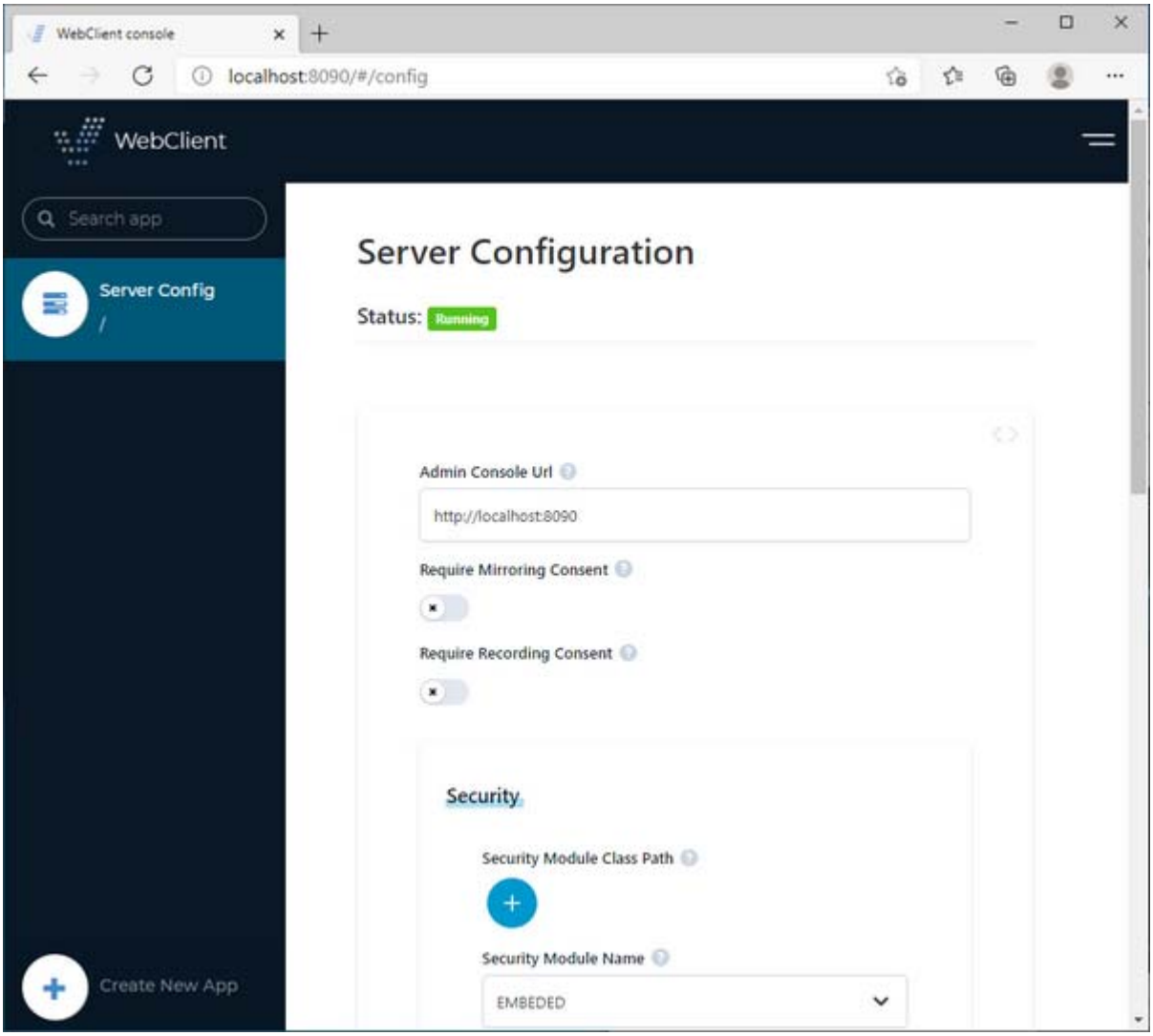
You will get this screen:



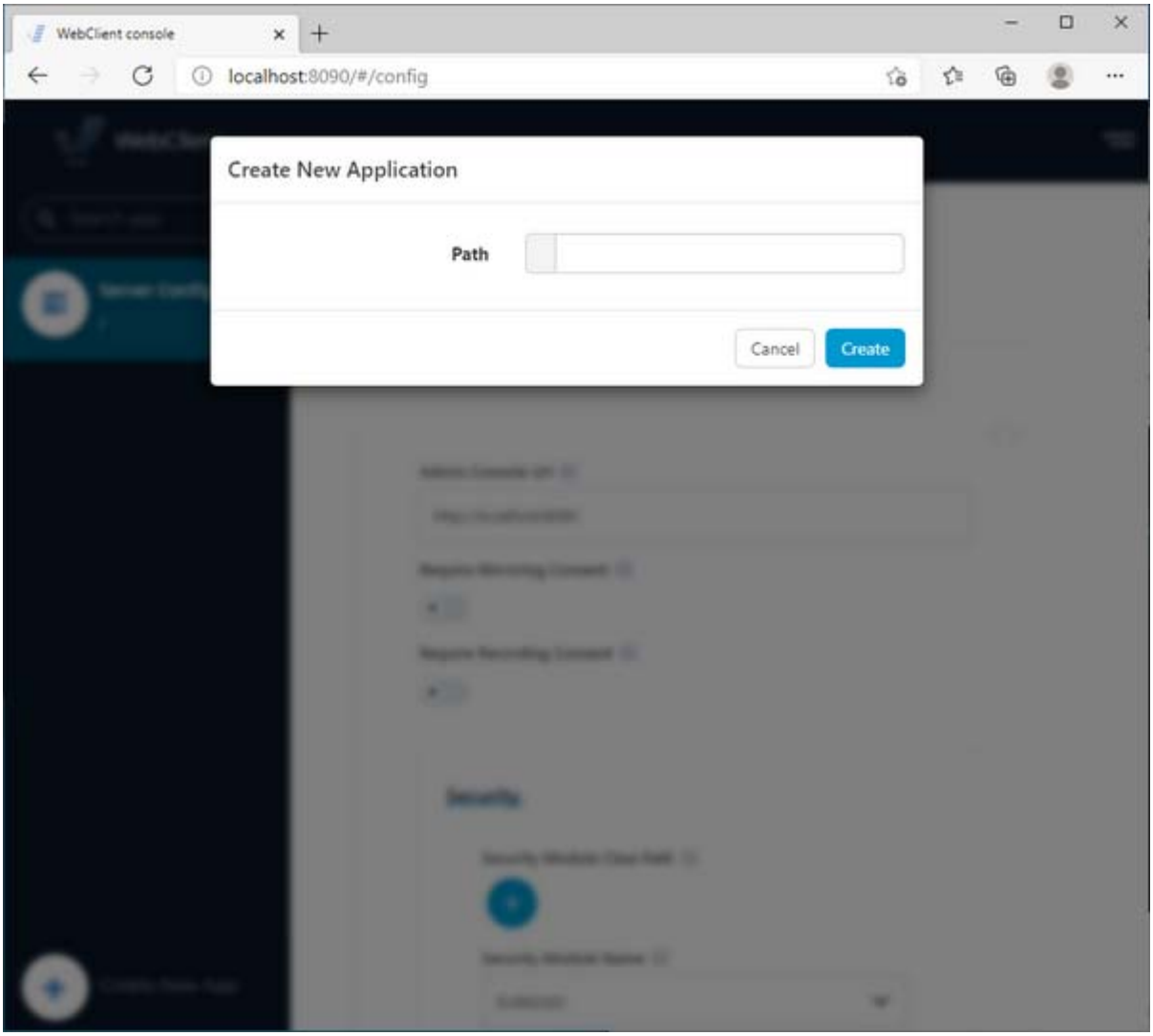
Log in as user "admin" with password "admin", you will get this screen:



Click on "Manage" and you will get this screen:

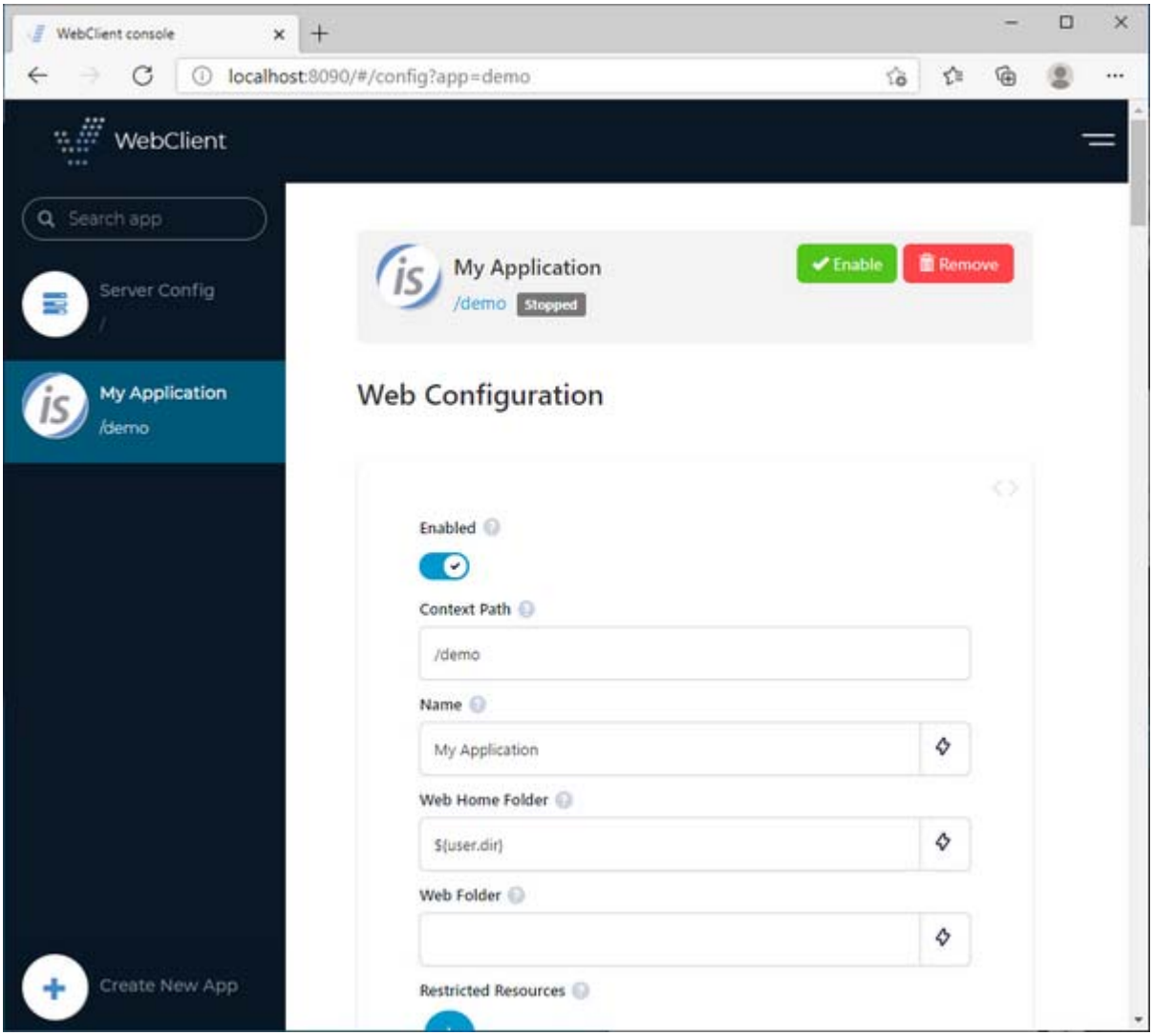


Click on "Create New App" in the bottom left corner of the page and you will get this screen:



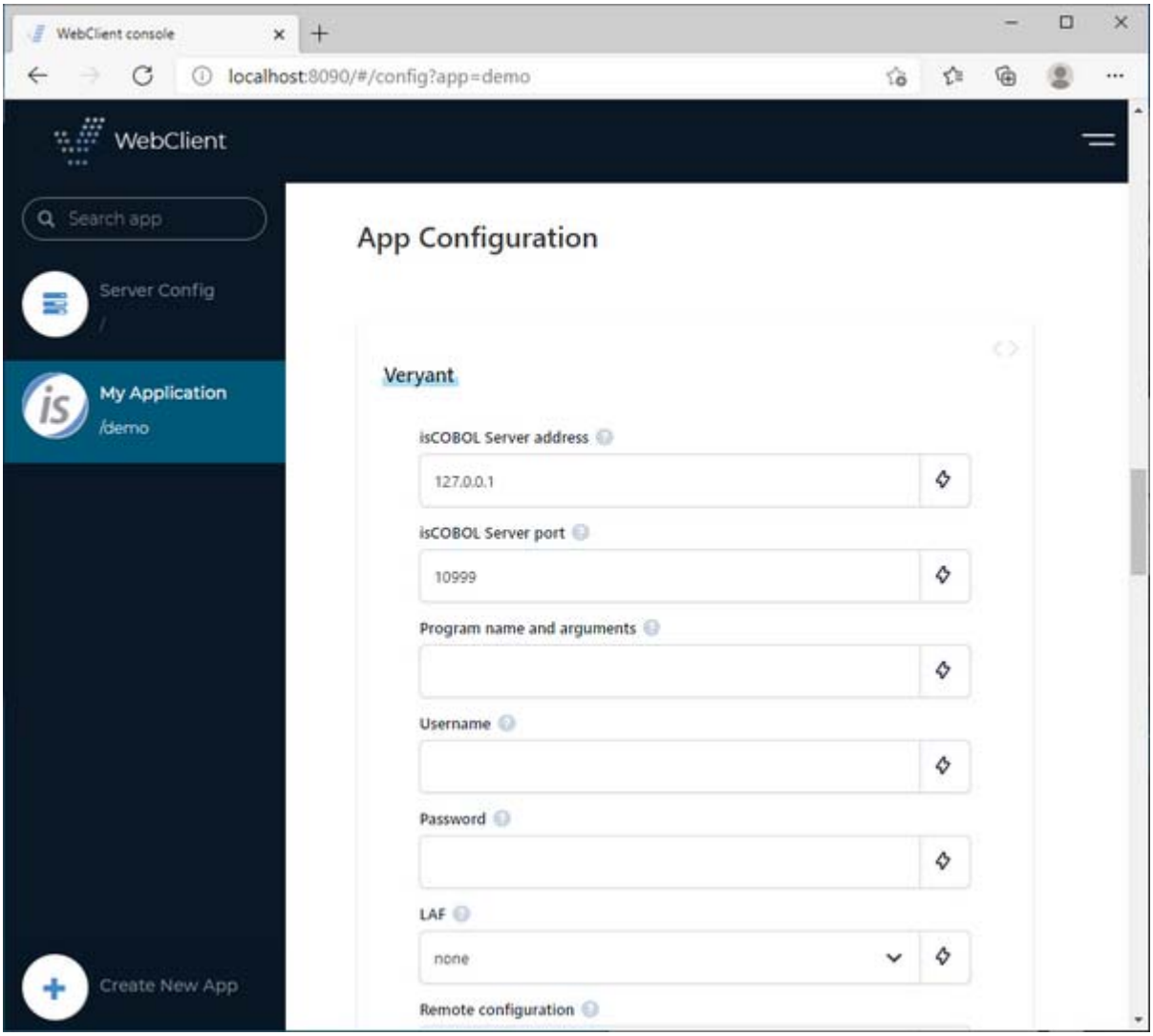
Type "demo" in the field. This is the name that will be used in the URL in order to use the COBOL application. Note that this name cannot be changed after the application is created.

Click on "Create" and you will be redirected to the configuration of the application:



Note - in this page applications are listed in alphabetical order and the first application in the list is selected when the page is loaded. You may need to click on your application name in the list on the left in order to load its configuration on the right

Click the *Enable* button in order to activate the application, then scroll down to reach the App Configuration section:



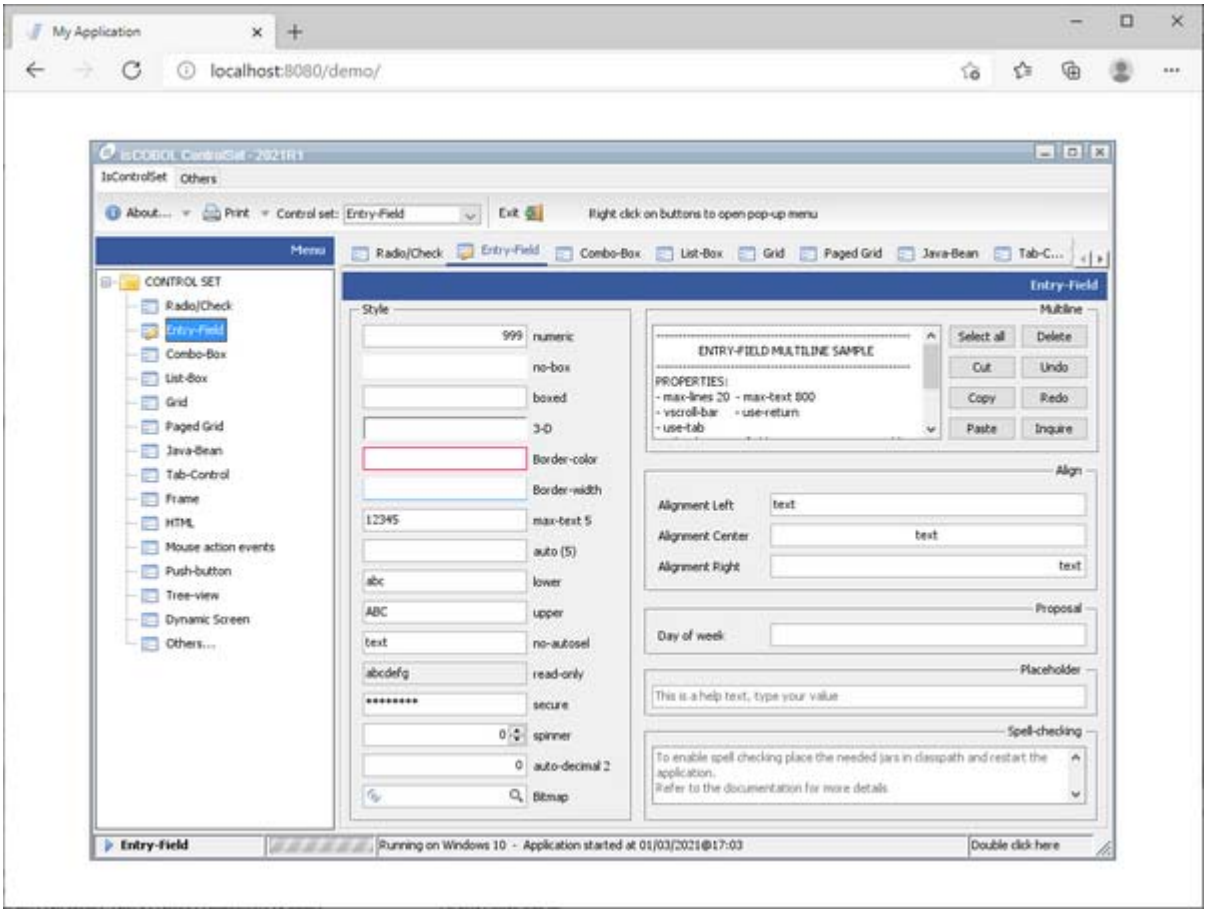
Type "ISCONTROLSET" (upper case) in the *Program name and arguments* field.

See Change the application configuration for more details about the configuration of a WebClient application.

Click on "Apply" when prompted.

At this point you can test your application from any web browser from any machine in the network by navigating to: http://*machine-ip*:*port*/demo.

Note - *machine-ip* and *port* must match the *server.host* and *server.http.port* values set Jetty Configuration.



Applications Monitoring and Configuration

Note - The applications configuration is saved in the file *webclient/webclient.config* under the isCOBOL installation folder. It's good practice to make a backup copy of this file every time you change it, as it may be overwritten by the installation of an isCOBOL SDK update.

Applications created in the WebClient can be monitored and configured through the WebClient Admin Console.

By default, the Admin Console is reachable via HTTP on the port 8090 of the server where you started the webclient-admin service, i.e.

```
http://localhost:8090
```

Refer to Jetty Configuration for instructions about how to use a different port.

The Admin credentials are required in order to access this app.

There are 4 sections:

Overview

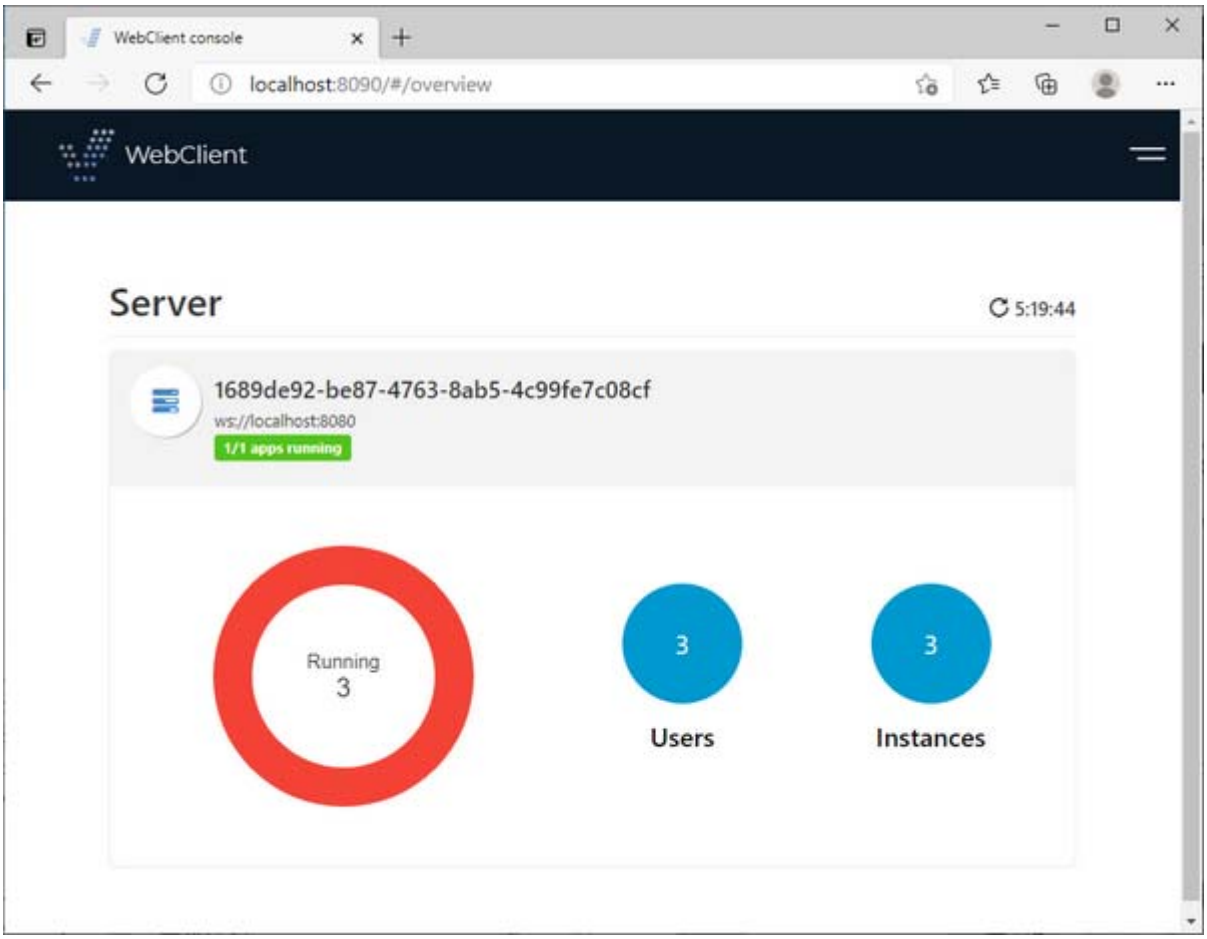
Applications

Sessions

Logs

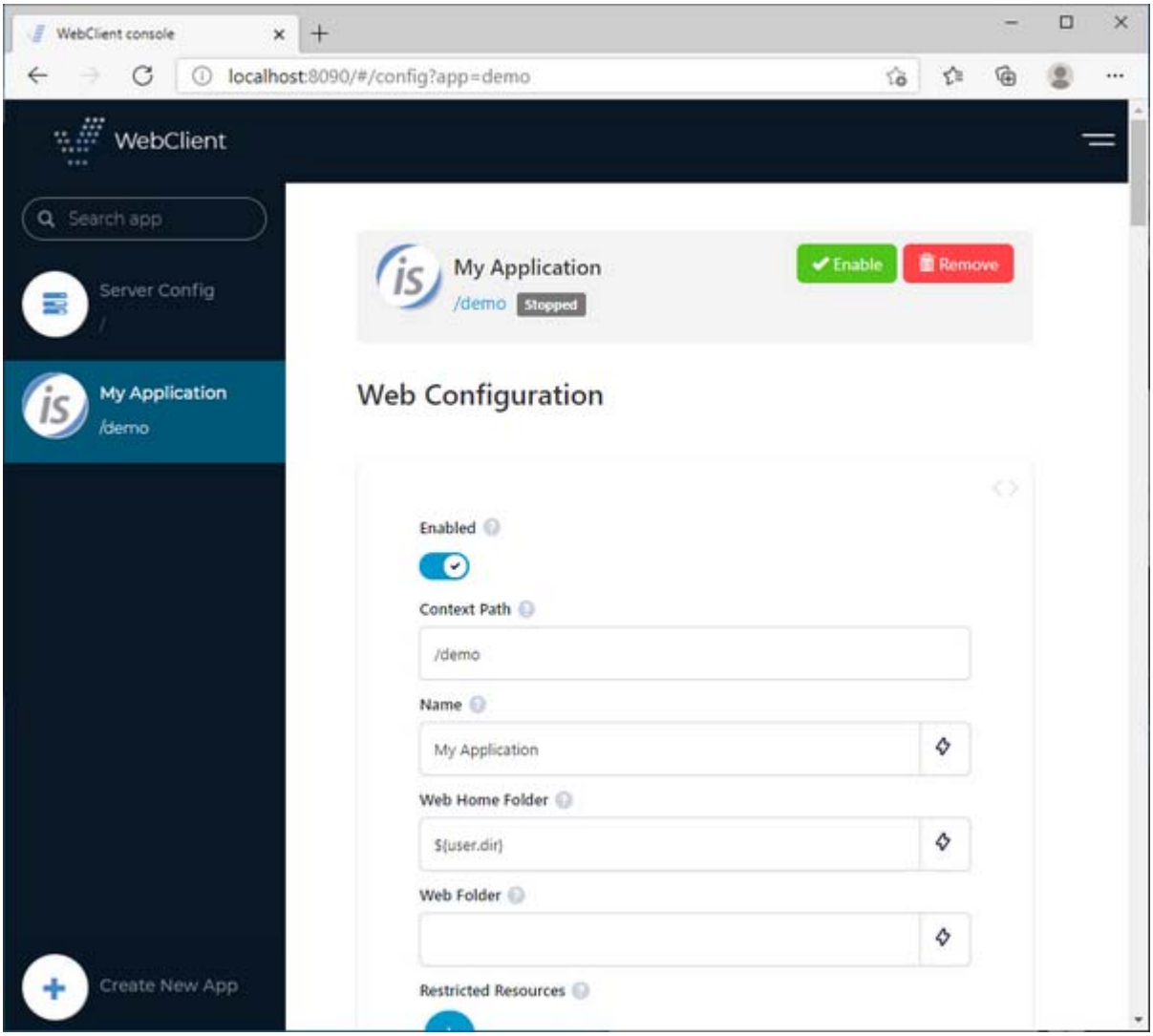
Overview

The Overview section provides an overview on the load of the servers known to the WebClient Admin Console. See Managing multiple WebClient servers from the same WebClient Admin Console for information about how to attach multiple servers with the same Admin Console.



Applications

The Applications section allows you to alter the settings of a specific application as well as create a brand new application. This is the default section where you're positioned after connecting to the Admin Console.



Here it's possible to

- Create a new application
- Enable or Disable applications
- Change the application configuration
- Remove an application

Create a new application

Click on the "Create New App" button and provide a name for your application. The new application appears in the list. Click on it to alter its configuration.

Refer to Getting Started for a step by step tutorial to create your first application.

Enable or Disable applications

Click on the "Disable" button to make the application unavailable to the users. You will be prompted to kill active connections, if any.

Disabling an application is useful during maintenance (e.g. during a update of program classes).

Click on the "Enable" button in order to make the application available to the users.

Change the application configuration

Select the desired application from the list on the left to access the list of available configuration entries.

Entry	Meaning
Enabled	**ON -** The application is Enabled by default when the service starts. **OFF -** The application must be enabled manually after the service starts.
Context Path	Url context path where the application will be deployed
Name	WebClient application name. This name will be displayed on the left panel and used to alphabetize the list of WebClient applicaitons.
Web Home Folder	Home directory for web related content. This is the base directory for every relative classpath entry.
Web Folder	Folder to be used to store customized static web files like HTML, CSS or JavaScript
Restricted Resources	List of paths that will be accessible only by authenticated users
Localization Folder	Folder where customized messages and translations are stored.
Icon	Image icon shown in the application selection dialog
CORS Origins	List of domains for which cross-origin resource sharing is allowed. This is useful if you're embedding the application in a page that resides on a different domain
Upload Size Limit	Maximum size in MB for uploaded files. A value of 0 means no limit
Max. Connections	Limit the maximum number of concurrent sessions for this application. By default, up to 10 concurrent sessions are allowed
Session Mode	Define if and how sessions can be restored. **ALWAYS_NEW_SESSION** - every time the application URL is loaded, a brand new runtime session is started. **CONTINUE_FOR_BROWSER** - every time the application URL is loaded from the same browser, the user is offered the choice of starting a new runtime or restoring the previous one. **CONTINUE_FOR_USER** - every time the application URL is loaded by the same user, the user is offered the choice of starting a new runtime or restoring the previous one. In this mode, the user can also restore the runtime session from different devices. This mode will have no effect if *Security Module Name* is not set to NONE, which would require users to perform authentication each time they open the application.
Monitor App Responsiveness	**ON -** Show a progress animation if Swing's Event Dispatch thread is not responding **OFF -** Don't advise users if Swing's Event Dispatch thread is not responding
Loading Animation delay	Delay in seconds before showing the progress animation. This setting is considered when Monitor App Responsiveness is ON. The minimum value is 2.

Entry	Meaning
Session Stealing	**ON -** If Session Mode is 'CONTINUE_FOR_USER', users can resume the WebClient session even if the connection is open in other browser. Former browser window will be disconnected. **OFF -** Users can't resume the WebClient session if the connection is open in other browser
Auto Logout	**ON -** Users are automatically logged out after the application finished. **OFF -** Users remain logged in also after the application finished.
Security	
Security Module Class Path	Additional classpath for built-in Security module or for defining custom security module. Use the '+' button to add a new entry. Use the 'x' button to remove an entry.
Security Module Name	**INHERITED -** Inherits the configuration set while **Configuring Users**. **NONE -** No authentication is required to access this application. The authentication may be required anyway to access WebClient. **EMBEDDED -** User authentication is required to access this application. Selecting this value will display a pop-up area with the list of current users, allowing you to edit them or to define new users. The rules to configure this field are the same described in Configuring Users except that they affect the single application instead of the whole WebClient
Data Store	
Data Store Module Class Path	Reserved for future use
Data Store Module Name	Reserved for future use
Veryant	
isCOBOL Server address	IP address of the machine where isCOBOL Server is listening. WebClient will connect to the isCOBOL Server in the same way as a isCOBOL Client. This is equivalent to the -hostname option of the isCOBOL Client
isCOBOL Server port	Port where isCOBOL Server is listening. WebClient will connect to the isCOBOL Server in the same way as a isCOBOL Client. This is equivalent to the -port option of the isCOBOL Client
Program name and arguments	Name of the main program of the application optionally followed by one or more variables. See Passing command line arguments and end user info to the COBOL program for more details.
Username	User name for authenticating to the isCOBOL Server in case iscobol.as.authentication is set to "2" in isCOBOL Server's configuration
Password	Password for authenticating to the isCOBOL Server in case iscobol.as.authentication is set to "2" in isCOBOL Server's configuration
LAF	Look and feel to be used to display application's windows
Remote configuration	Remote configuration file for the application. This is equivalent to the -c option of the isCOBOL Client

Entry	Meaning
Local configuration	Local configuration file for the application. This is equivalent to the -lc option of the isCOBOL Client
Remote Debug	**ON -** Enable remote debug on the isCOBOL Server. **OFF -** Disable remote debug on the isCOBOL Server.
Remote debug port	Port number of the Remote Debugger in the isCOBOL Server
Home Folder	Working directory of the application. This is equivalent to the working directory of the isCOBOL Client
Theme	Specifies the decoration theme for the application windows. It affects mainly the title bar and the menu bar of the windows
Fonts	Customize logical font mappings and define physical fonts available to application. These fonts will be used for DirectDraw as native fonts. Use the '+' button to add a new mapping. Use the 'x' button to remove a mapping. Every mapping is composed of two items: *Key* and *Value*. *Key* is the name of the font. *Value* is the path to the font file. Only True Type (ttf) fonts are supported. If no fonts are defined, necessary fonts are loaded from the system by WebClient as the COBOL application requests them. This rule applies to *iscobol.font* configuration settings as well as calls to the W$FONT routine. If some fonts are defined, these fonts will be the only fonts available for the COBOL application, so be sure to map all the necessary fonts otherwise some calls to W$FONT may fail and some *iscobol.font* configuration settings may be ignored. Configuring fonts is the only way to change the font used by window title bar and menu bar. For example, the following settings will change the look of the title bar and menu bar: *Name*: "dialog", *Value*: "${user.dir}/fonts/Roboto-Regular.ttf" *Name*: "dialoginput", *Value*: "${user.dir}/fonts/RobotoMono-Regular.ttf" *Name*: "serif", *Value*: "${user.dir}/fonts/RobotoSlab-Regular.ttf"
DirectDraw Rendering	DirectDraw rendering mode uses canvas instructions to render the application instead of server-rendered png images. DirectDraw improves performance but is not recommended for applications with a lot of graphics content.
JavaFX Support	**ON -** Ability to use also JavaFx components. **OFF -** Only Swing and AWT components allowed.
Compositing Window Manager	Reserved for future use
Enable Debug Mode	Reserved for future use
Enable Test Mode	Reserved for future use
Java	
Working Directory	Specifies the working directory of the isCOBOL Client on the machine where WebClient is running.
JRE Executable	Path to the java executable
Java Version	Version of the java executable

Entry	Meaning
Class Path	Local Classpath. The isCOBOL's lib directory content must appear here. This is equivalent to the isCOBOL Client's Classpath
JVM Arguments	Java options like -Xmx go here. You should use the same options that you would use to start the isCOBOL Client
Launcher Type	Reserved for future use
Session	
Session Timeout	Specifies how long the application will be left running after the user closes the browser. User can reconnect in this interval and continue in last session. The value is expressed in seconds.
Timeout if Inactive	**ON -** Session Timeout will apply for user inactivity. **OFF -** Only disconnected sessions will time out.
Logging	
Session Logging	**ON -** log sessions in a separate log file. **OFF -** don't log sessions in a separate log file.
Session Log Size	Maximum size in MB for a session log file
Maximum Session Logs Size	Maximum size in MB of all session log files. When exceeded, the older log files are deleted to gain space
Statistics Logging	**ON -** log statistics for sessions. **OFF -** don't log statistics for sessions.
Features	
Isolated Filesystem	**ON -** the Open and Save dialogs of C$OPENSAVEBOX can only browse the WebClient's Upload Folder and its subfolders. **OFF -** the Open and Save dialogs of C$OPENSAVEBOX can browse every folder of the machine where WebClient is running.
Uploading Files	Enable the ability to upload files through the Open File dialog generated by the C$OPENSAVEBOX library routine.
Deleting Files	Enable the ability to delete files from the Open and Save dialogs generated by the C$OPENSAVEBOX library routine.
Downloading Files	Enable the ability to download files through the Save File dialog generated by the C$OPENSAVEBOX library routine.
Auto-Download from Save Dialog	Enable the automatic download of files to the end user's PC when the Save File dialog of C$OPENSAVEBOX is called.
Transparent Open File Dialog	**ON -** when C$OPENSAVEBOX is called to open a file, show only the client-side file browser and upload the selected file. **OFF -** when C$OPENSAVEBOX is called to open a file, show the Open dialog along with the client-side file browser and upload the selected file.
Transparent Save File Dialog	**ON -** when C$OPENSAVEBOX is called to save a file, ask only for the file name. **OFF -** when C$OPENSAVEBOX is called to save a file, show the Save dialog along with the prompt for file name.

Entry	Meaning
Upload Folder	Folder where files uploaded by the user through C$OPENSAVEBOX are stored
Clear Upload Folder	**ON -** Delete all files in the transfer folder when the application process is terminated. **OFF -** Don't delete any file when the application process is terminated.
Allow JsLink	Reserved for future use
JsLink White List	Reserved for future use
Allow Local Clipboard	**ON -** Allow access to the end user's PC clipboard. **OFF -** The user can't cut, copy or paste text in the application screen.
Allow Server Printing	**ON -** Allow access to the printers installed on the machine where WebClient is running. **OFF -** Print to PDF using the internal WebPrintService printer and send the PDF to the client browser.

After changing one or more of the above settings, you can either

- Click on "Apply" if you wish to activate the new configuration, or
- Click on "Reset" to clean your changes and restore the active configuration.

The application's configuration is saved in the file *webclient/webclient.config* under the isCOBOL installation folder.

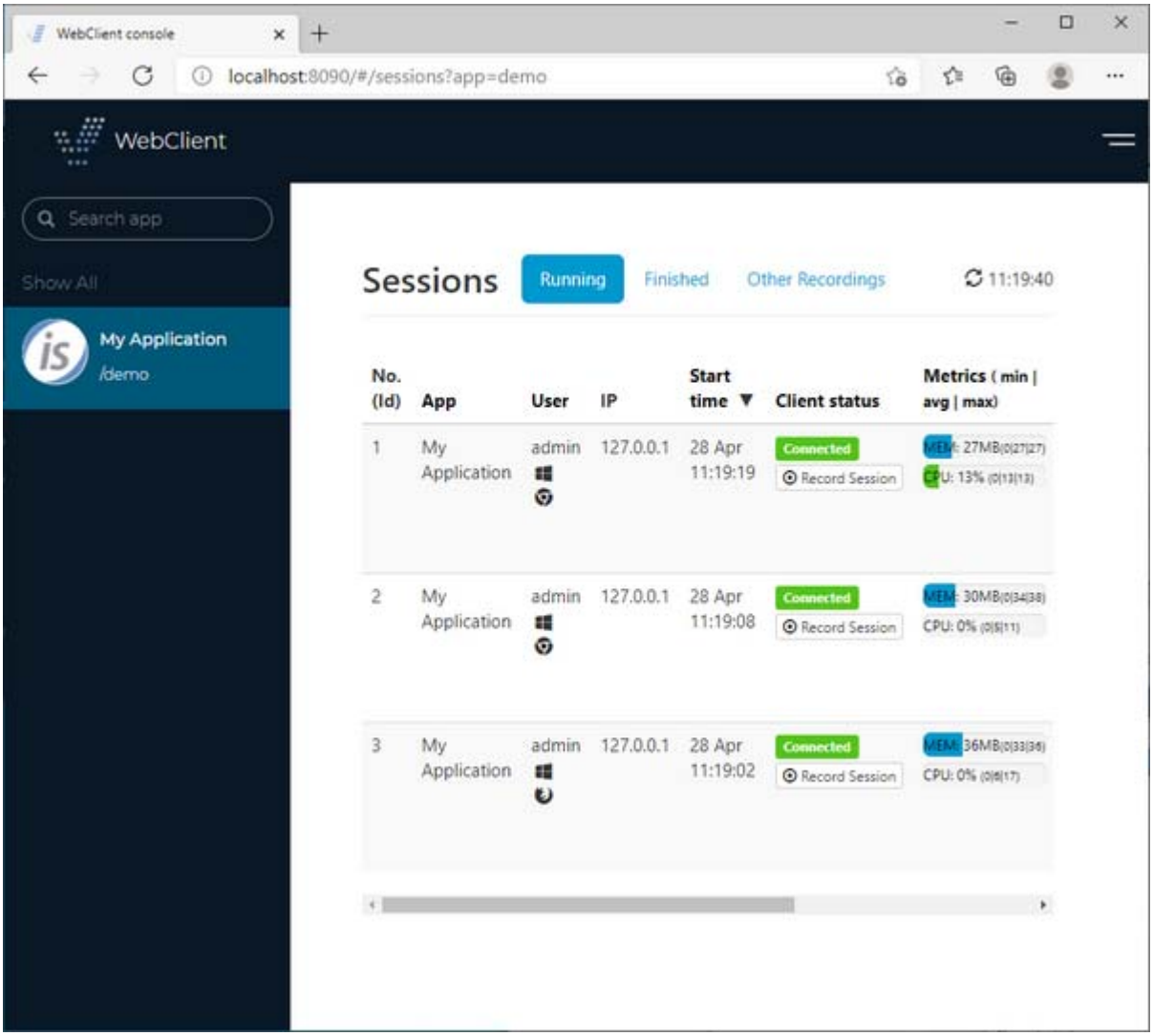
Remove an application

Click on the "Disable" button to make the application unavailable to the users. You will be prompted to kill active connections, if any.

Once the application is disabled, click on the "Remove" button to delete the application from the WebClient environment.

Sessions

The Sessions section lists the sessions of each application.



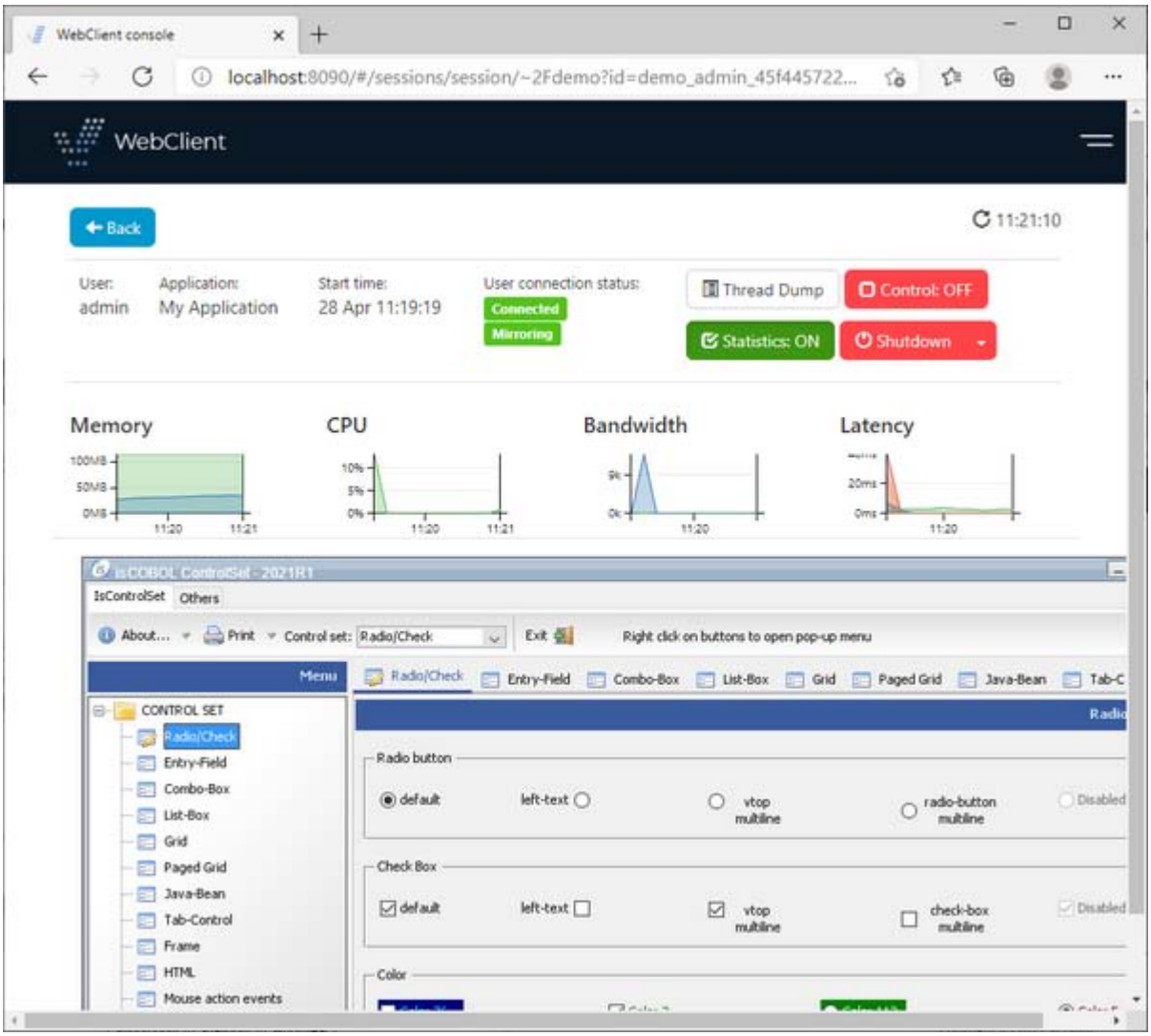
For each session, the following information is returned:

Entry	Meaning
No. (Id)	Unique ID assigned by WebClient to the session
User	Login user name, or "anonym" if no login was performed
IP	IP of the end user's PC
Start time	Date and time when the session was started
Client status	Status of the client PC
Metrics	CPU and Memory usage
Bandwitch	Bandwitch usage

Entry	Meaning
Latency	Latency

For each session, the following actions are possible:

- Click on the "Thread dump" button in order to take a thread dump of the underlying JVM. The dump can be reviewed by clicking on Warnings in the "Client status" column.
- Click on the "Shutdown" button in order to terminate the session, causing the user to be disconnected.
- Click on "Record Session" in order to record the user actions. The recording will stop when the user session terminates and the recorded video will be playable by clicking on the "Play" button in the *Finished sessions* list. If multiple recordings exist for a session, you will find multiple "Play" buttons, one for each recording.
- Click on the "View" button in order to monitor the user activity on the application.



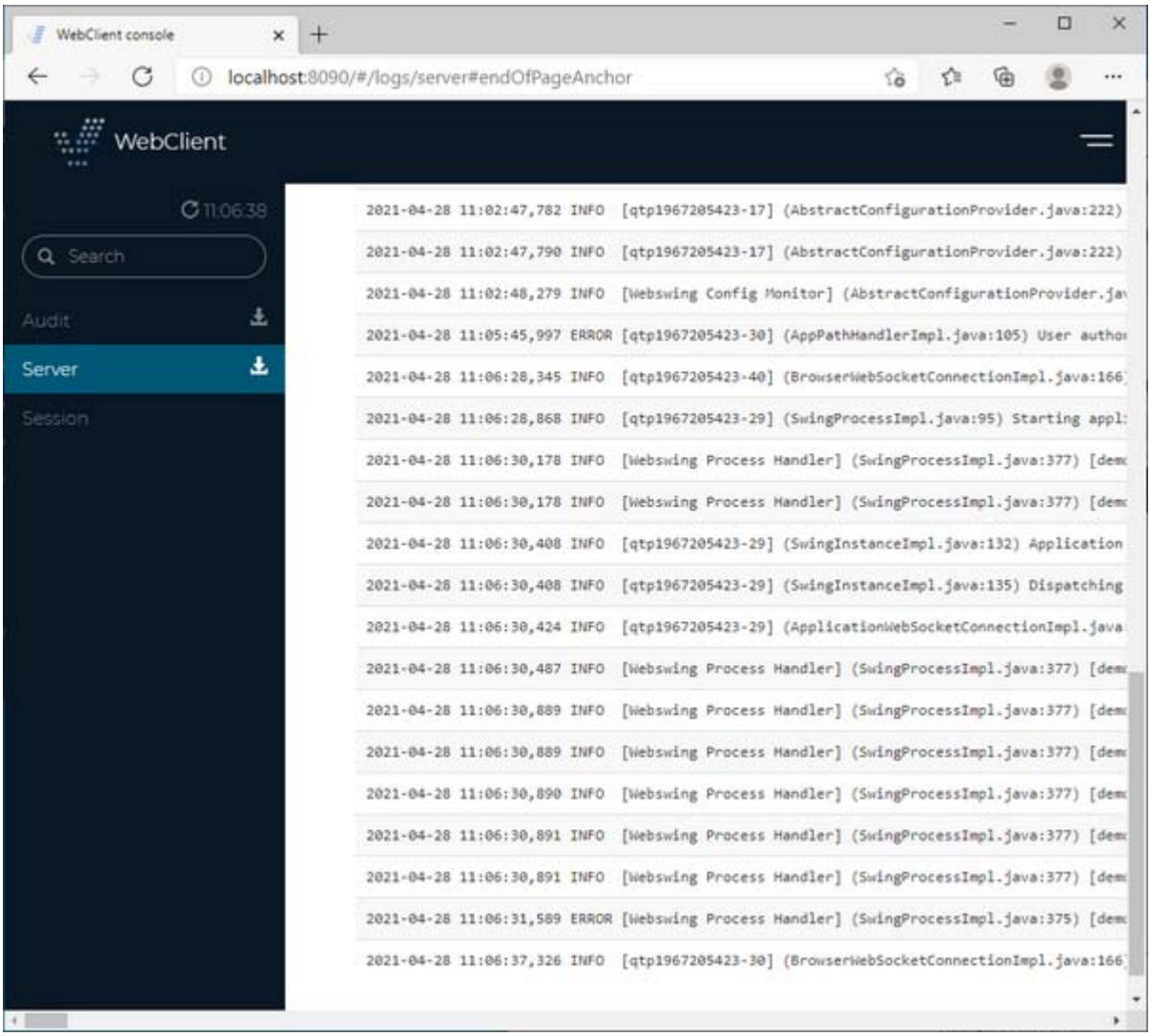
From this dialog it's possible to:

- Click on the "Thread dump" button in order to take a thread dump of the underlying JVM. The dump can be reviewed by clicking on Warnings in the "Client status" column.

- Click on "Control:OFF" changing it to "Control:ON" and take control of the application. This is useful, for example, in order to provide remote support.
- Click on "Statistics:ON" changing it to "Statistics:OFF" to stop monitoring CPU, memory and bandwitch.
- Click on the "Shutdown" button in order to terminate the session, causing the user to be disconnected.

Logs

The Logs section shows the content of WebClient's log files.



Configuring Users

By default only the Admin user exists and a login is required only for administration operations like creating and configuring applications.

It's possible to create additional users and configure the applications to ask for user credentials when the session starts.

Users can be configured through the WebClient Admin Console.

By default, the Admin Console is reachable via HTTP on the port 8090 of the server where you started the webclient-admin service, i.e.

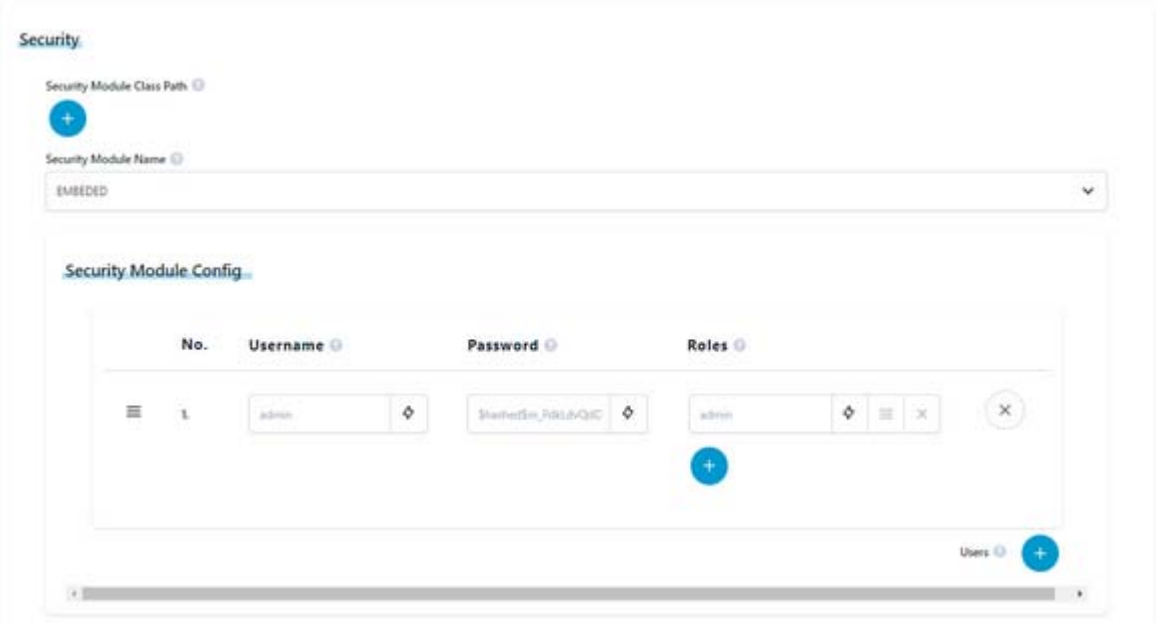
```
http://localhost:8090
```

Refer to Jetty Configuration for instructions about how to use a different port.

The Admin credentials are required in order to access the Configuration page.

By default, users are defined via the WebClient interface as follows:

1. set the *Security Module Name* field to EMBEDDED to make the list of current defined users appear



2. click on the "+" button on the bottom right for a new row to appear in the list
   a. fill *Username* and *Password* fields with the new user credentials
   b. optionally assign a role to the user (see Roles below for more information)
   c. click on the "Apply" button

Roles

WebClient users can be assigned the following roles:

Role	Permissions
admin	create new users create new applications change the configuration of an application monitor the activity of connected users run an application

Role	Permissions
support	view the configuration of an application monitor the activity of connected users run an application
<none>	run an application

Only admin and support users have access to the Dashboard.

Setting *Security Module Name* field to NONE allows all users to access WebClient without authentication. This is not good practice.

The users configured here are available in the whole WebClient environment. It's also possible to define different users for the single applications. See Applications Monitoring and Configuration for details.

It's possible to define users also through a property file or a database. In order to enable these features:

1. click on the "+" below *Security Module Class Path*
2. select the value "${webclient.rootDir}/security/database+property/*" from the list
3. click on the "Apply" button

After it, the list of options under *Security Module* changes from

- NONE
- EMBEDDED

to

- NONE
- EMBEDDED
- org.webswing.security.modules.database.DatabaseSecurityModule
- org.webswing.security.modules.property.PropertySecurityModule

For more information about configuring users via property file or database, refer to the next chapters:

- Reading users from a property file
- Reading Users from a JDBC data source

Note - regardless of the method that you choose for configuring users, ensure to have at least one user with role "admin", otherwise it will not be possible to alter the WebClient configuration.

Reading users from a property file

In order to configure users via property file, set the *Security Module Name* field to "org.webswing.security.modules.property.PropertySecurityModule".

A new section named *Security Module Config* appears. This section includes only one field named *File*, that allows you to provide the location of the property file (by default a file named *user.properties* is searched in the WebClient working directory).

Each line in that file defines a user. The syntax is:

```
user.<username>=<password>[,role1][,role2]
```

For example:

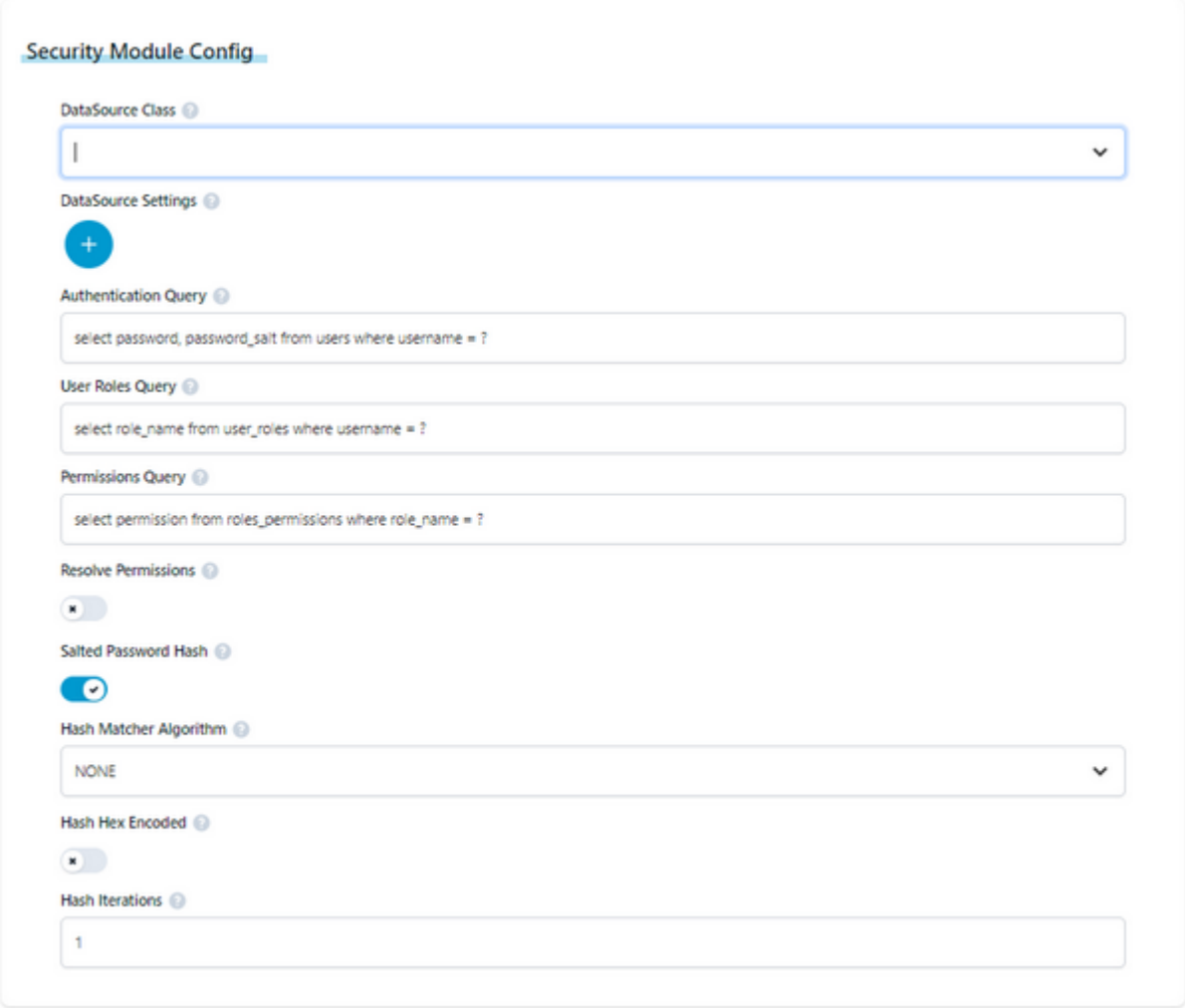
```
user.admin=admin,admin
user.support=support,support
user.user=user
```

Reading Users from a JDBC data source

Users can be registered in a database that WebClient will query via JDBC.

Every database that allows JDBC connections is suitable, including the c-treeRTG SQL engine.

In order to make WebClient look for users in a JDBC data source, set the *Security Module Name* field to "org.webswing.security.modules.database.DatabaseSecurityModule".



Use the *Security Module Class Path* to provide the full path of the jar libraries of the JDBC drivers you wish to use. Use the "+" button to add a new driver. Use the "x" button to remove a driver.

Set the *DataSource Class* field to the name of the data source class. The class must implement the DataSource interface. The field includes a list of known data source classes. If the class that you wish to use doesn't appear in this list, type the class name in the field.

Examples:

	Security Module Class Path	DataSource Class
c-treeSQL	/path/to/ctreeJDBC.jar	ctree.jdbcx.CtreeDataSource
Oracle	/path/to/ojdbc7.jar	oracle.jdbc.pool.OracleDataSource
MySQL	/path/to/mysql-connector-java-bin.jar	com.mysql.jdbc.jdbc2.optional.MysqlDataSource
PostgreSQL	/path/to/postgresql.jdbc4.jar	org.postgresql.ds.PGSimpleDataSource

When the *DataSource Class* has been selected, the *DataSource Settings* will provide possible parameters. Use the "+" button to add a new setting. Use the "x" button to remove a setting.

Parameter name	Parameter value
serverName	IP or name of database server
databaseName	Name of database with user configuration tables
user	Username to connect to the database
password	Password to connect to the database
portNumber	TCP port number used by the database server

The fields *Authentication Query*, *User Roles Query* and *Permissions Query* show the queries that will be performed by WebClient in order to retrieve the desired data. Ensure that your database includes the required tables and fields. Fields must be of type VARCHAR.

The minimum database schema to support the DATABASE authentication needs the following tables:

Table "users"	
username	varchar()
password	varchar()
password_salt	varchar()
Table "user_roles"	
username	varchar()
role_name	varchar()
Table "roles_permissions"	
role_name	varchar()
permission	varchar()

If your tables have different names, different field names or different field type, then you should adapt the queries in the *Authentication Query*, *User Roles Query* and *Permissions Query* fields. For example, if you're using a c-treeRTG database whose tables are ISAM files that were sqlized, then the field type is CHAR instead of VARCHAR, so the queries should be changed from:

```
select password, password_salt from users where username = ?
select role_name from user_roles where username = ?
select permission from roles_permissions where role_name = ?
```

to:

```
select trim(password), trim(password_salt) from users where trim(username) = ?
select trim(role_name) from user_roles where trim(username) = ?
select trim(permission) from roles_permissions where trim(role_name) = ?
```

If you wish to store password as clear text, set the *Hash Matcher Algorithm* field to NONE. If you wish to store password encoded, select the appropriate encoding in the *Hash Matcher Algorithm* field. For example, if passwords are stored as MD5 hash, set the *Hash Matcher Algorithm* to MD5.

JETTY and JMS Configuration

The WebClient services are based on Eclipse Jetty, a Java HTTP (Web) server and Java Servlet container.

The WebClient services use the Java Message Service (JMS) to communicate with the underlying COBOL applications.

Jetty Configuration

By default the WebClient service starts on localhost on port 8080 while the WebClient Admin Console starts on localhost on port 8090.

You can change these settings by editing the files *webclient/jetty.properties* and *webclient/admin/jetty.properties* under the isCOBOL installation folder. It's good practice to make a backup copy of these files every time you change them, as they may be overwritten by the installation of an isCOBOL SDK update.

Entry	Meaning
jetty.webclient.server.host	IP address or machine name where the service listens for connections. Replace 'localhost' by the IP of the current PC if you wish to allow connections from other machines in the network.
jetty.webclient.server.http	Enable or disable listening on the HTTP protocol
jetty.webclient.server.http.port	Port number the server listens to for HTTP connections

Note - If you wish to disable the HTTP protocol, don't comment *jetty.webclient.server.http*, but set it to "false". If the entry is commented, the service may not start.

The file jetty.properties contains also entries to enable secure HTTP (HTTPS).

Entry	Meaning
jetty.webclient.server.https	Enable or disable listening on the HTTPS protocol
jetty.webclient.server.https.port	Port number the server listens to for HTTPS connections
jetty.webclient.server.https.truststore	Location of the truststore file
jetty.webclient.server.https.truststore.password	Truststore password
jetty.webclient.server.https.keystore	Location of the keystore file
jetty.webclient.server.https.keystore.password	Keystore password

Note - These entries point to a self-signed certificate. You should replace them with a valid certificate released by a certificate authority if you wish to use HTTPS.

JMS Configuration

By default JMS uses the port 34455. You can change this port via the *webclient.jmsUrl* property on the command line.

The value of this property must be specified in the form "nio://<serverNameOrIp>:<port>".

For example, in order to start WebClient on localhost using the port 12345 for JMS, use this command:

- Windows

```
cd %ISCOBOL%\bin
webcclient.exe -J-Dwebclient.jmsUrl=nio://127.0.0.1:12345
```

- Linux

```
cd $ISCOBOL/bin
./webcclient -J-Dwebclient.jmsUrl=nio://127.0.0.1:12345
```

Logging

WebClient generates and updates the following log files in the working directory:

audit.log	This log traces the access to the configuration. It is useful if more than one user can access to the configuration.
stats.log	This log stores the statistics of applications usage. Only the activity of connections coming from foreign machines is traced, the activity on localhost is not traced. This information is reflected by the charts shown in the Dashboard.
webclient.log	This log traces the startup of the WebClient service and the COBOL applications. Java exceptions, if any, are stored in this log, so this is the first thing to check if you experience odd behaviors.

When WebClient is restarted, the above files are not initialized, the new log content will be appended to them.

The logging feature is implemented via Log4j with the *log4j.properties* configuration file stored in the *WEB-INF/classes* directory of *webclient/webclient-server.war*. The settings in *log4j.properties* are suitable for most environments, but you may want to review them in some cases.

Passing command line arguments and end user info to the COBOL program

In the "Program name and arguments" field, the following variables can be used:

${webclient.appPath}	Context path of the application
${webclient.homeFolder}	Installation folder of WebClient
${webclient.rootDir}	Root directory used to resolve relative paths
${webclient.server.host}	IP address of the server where WebClient is running
${webclient.server.port}	Port where WebClient is listening
${user}	WebClient specific logged in user name
${clientId}	WebClient specific unique browser identifier
${clientIp}	IP address of browser that started this application
${clientLocale}	Locale of browser that started this application
${customArgs}	Parameters specified via the "args" parameter in the URL. The value of "args" matches the parameters that you would pass on the command line when running the isCOBOL Client from a command prompt. Multiple parameters must be separated by "%20" that matches the space you would use on the command line. For example, a URL like: *http://.../?args=ABC%20123* matches a command like: *iscclient PROG ABC 123*

In addition to the above variables, you can reference every operating system environment variable and every Java property by decorating their name with "${}". For example ${path} will contain the value of the Path environment variable (%PATH% on Windows, $PATH on Linux/Unix), while ${java.version} will contain the value of the Java version number as returned by the statement java.lang.System.getProperty("java.version").

These variables are received by the COBOL program as chaining parameters in the order they appear in the field "Program name and arguments". Since *customArgs* generates a variable number of chaining parameters for the COBOL program, it's good practice to put it at the end of the list, if used.

For example, if you set

Program name and arguments	PROG ${clientIp} ${customArgs}

Use a URL like this to pass 'ABC' and '123' as command line arguments:

```
http://yourwebsite/yourapp/?args=ABC%20123
```

Use the following COBOL code to receive the IP address of the end user and the two command line parameters:

```
program-id. prog.
...
working-storage section.
...
77  wrk-clientIp pic x any length.
77  wrk-param-1  pic x any length.
77  wrk-param-2  pic x any length.
...
procedure division chaining wrk-clientIp
                            wrk-param-1
                            wrk-param-2
                            .
```

Managing multiple WebClient servers from the same WebClient Admin Console

The WebClient Admin Console can connect more than one WebClient server and allow you to monitor and configure these servers in one single place. When this approach is used, the configuration is synchronized between the various WebClient servers (every WebClient server will include the same apps and the same users). This is useful for load balancing purposes.

In order to attach multiple WebClient servers

- the Admin Console and the various WebClient servers must use the same secret key. The secret key is defined in the file *webclient/webclient.properties* for WebClient and in the file *webclient/admin/webclient-admin.properties* for the Admin Console. The default setting is:

```
webclient.connection.secret =
change_this_in_production_0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000
```

- the WebSocket URL of the various WebClient servers must be listed in the file *webclient/admin/webclient-admin.properties* at the *webclient.server.websocketUrl* setting. By default this setting points to a WebClient on the localhost, e.g.

```
webclient.server.websocketUrl = ws://localhost:8080
```

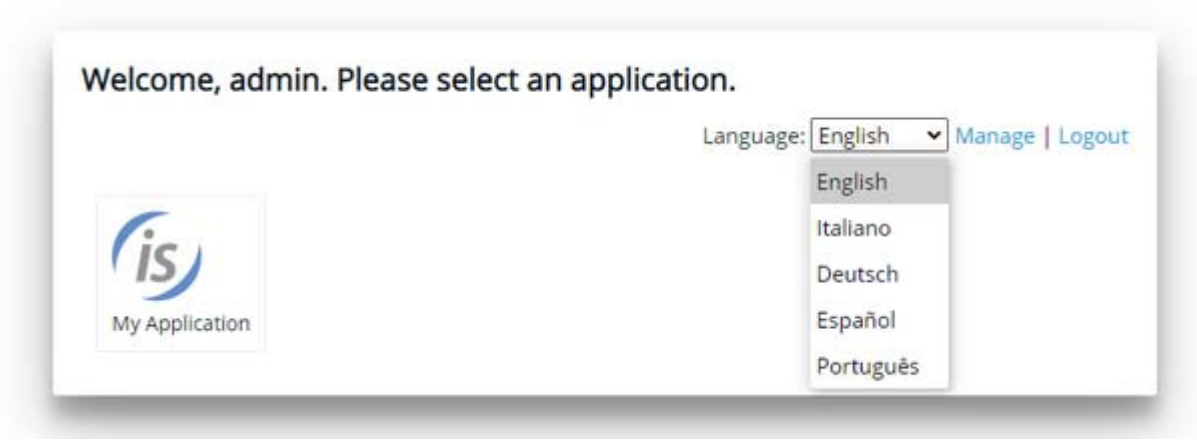
  You can replace this value or add new values to it. Multiple values must be comma separated. For example, assuming that we have another WebClient listening on server2 on the default port 8080, we will change the setting as follows:

```
webclient.server.websocketUrl = ws://localhost:8080,ws://server2:8080
```

Changing the langauge in WebClient's user interface

WebClient includes a series of messages and dialogs that are shown to the user. These messages and dialogs are in English by default, however it's possible to have them translated in multiple languages. The isCOBOL SDK includes a series of language packs that are ready to use. You can customize these language packs as well as add brand new ones.

The available languages are shown in the home page of WebClient, in a combo-box near the list of applications:



By changing the value of this combo-box, the WebClient's user interface is translated to the selected language.

Language packs are installed in the lang folder of WebClient. This folder includes a sub folder for every language and a select.json file where available language are listed. Inside each language folder you find a file named msg.json that includes the translation of each WebClient message label.

Language Customization

To customize a language translation, just edit the corresponding msg.json file.

To create a new language:

1. Copy an existing one, e.g. copy lang/en-US to lang/mylanguage
2. Edit the file lang/mylanguage/msg.json replacing JSON field values with your custom translations
3. Edit the file lang/select.json adding your custom language to the list

Note - a basic knowledge of the JSON format is required to customize WebClient's languages.

Making an application start with a specific language

As explained above, the users can select the desired language in the combo-box shown in the home page of WebClient. However, in most cases the users will go directly to the web application instead of landing on this home page (e.g. they browse to "http://serverip:port/appname" not to "http://servip:port").

In order to make users see a specific language in the WebClient's user interface when they browse directly to the application, proceed as follows:

1. Create a folder that will host your custom HTML files

2. Create a file named index.html in that folder and put the following content into the file

```
<!DOCTYPE html>
<html class="ws-fullscreen" lang="en">

<head>
    <title>WebClient</title>
    <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
    <meta http-equiv="Content-Security-Policy" content="connect-
src 'self' ws: wss: data:">
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, user-scalable=no, initial-
scale=1.0">
    <link rel="stylesheet" href="css/style.css"/>
    <link rel="icon" href="favicon.ico"/>
    <link rel="manifest" href="manifest.json" />
</head>

<body>
    <div class="webswing-element" data-webswing-instance="webswingInstance0">
        <div id="loading" class="ws-modal-container">
            <div class="ws-login">
                <div  class="ws-login-content"><div class="ws-
spinner"><div class="ws-spinner-dot-1"></div> <div class="ws-spinner-dot-2"></div></
div></div>
            </div>
        </div>
    </div>

<script>
    var webswingInstance0 = {
        options: {
            autoStart: true,
            args: getParam('args'),
            recording: getParam('recording'),
            debugPort: getParam('debugPort'),
            recordingPlayback: getParam('recordingPlayback'),
            securityToken: getParam('securityToken'),
            realm: getParam('realm'),
            debugLog: getParam('debugLog')
        }
    }
    function getParam(name) {
        name = name.replace(/[\[]/, "\\[").replace(/[\]]/, "\\]");
        var results = new RegExp("[\\?&]" + name + "=([^&#]*)").exec(location.href);
        return results == null ? null : decodeURIComponent(results[1]);
    }
</script>
```

```
<script data-webswing-global-var="webswing">

    function getAppName() {
        var xmlhttp1 = new XMLHttpRequest();
        xmlhttp1.onreadystatechange = function() {
            if (xmlhttp1.readyState == XMLHttpRequest.DONE ) {
                var appName = xmlhttp1.status == 200 ? xmlhttp1.responseText : "";
                if (appName=='')
                    setTimeout(function() {
                        getAppName();
                    },4000);
                else
                    document.title = appName;
            }
        };
        var path = document.location.toString();
        if (path.indexOf('?')>0)
            path = path.substring(0, path.indexOf('?'));
        xmlhttp1.open("GET", path + "rest/info", true);
        xmlhttp1.send();
    }
    (function (window, document) {
        var loader = function () {
            if (!window.location.origin) {
                window.location.origin = window.location.protocol + "//
" + window.location.hostname + (window.location.port ? ':' + window.location.port :
'');
            }
            var baseUrl = document.location.origin + document.location.pathname;
            baseUrl = baseUrl.indexOf("/", baseUrl.length - 1) !== -
1 ? baseUrl : (baseUrl + "/");
            var xmlhttp = new XMLHttpRequest();
            xmlhttp.onreadystatechange = function () {
                if (xmlhttp.readyState == XMLHttpRequest.DONE) {
                    var version = xmlhttp.status == 200 ? xmlhttp.responseText : "un
defined";
                    var script = document.createElement("script"),
                        tag = document.getElementsByTagName("script")[0];
                    script.src = baseUrl + "javascript/webswing-
embed.js?version=" + version;
                    tag.parentNode.insertBefore(script, tag);
                }
            };
            xmlhttp.open("GET", baseUrl + "rest/version", true);
            xmlhttp.send();

            getAppName();
```

```
        };
        window.addEventListener ? window.addEventListener("load", loader, false) : w
indow.attachEvent("onload", loader);
    })(window, document);
</script>
</body>

</html>
```

Note - this is the default index page generated by WebClient when you don't provide any custom html file for your application.

3. In the index page, at the very beginning of the first script tag, set the global variable webclientLang to the desired language id using the localStorage.setItem() function. For example, in order to use Spanish language:

```
...
<script>
    localStorage.setItem('webclientLang', 'es-ES');
    var webswingInstance0 = {
        options: {
...
```

Note - the language id must match with the name of one of the folders installed under the lang directory of WebClient.

4. In the app configuration, set Web Folder to the path of folder you created at step 1.

Deploying in Tomcat

Even though WebClient comes with an embedded Jetty server, it is also possible to deploy it in an external servlet container like Tomcat. Other J2EE servers can work as well, as long as they support the Servlet 3.0 spec.

WebClient

To deploy WebClient to Tomcat follow these steps:

1. Make a copy of *webclient-server.war* from the isCOBOL SDK to Tomcat's *webapps* folder
2. In *webclient.properties* set the property webclient.server.websocketUrl to

```
ws://localhost:<port>
```

   with the port that Tomcat is running on

3. In *catalina.properties* add the following properties:

```
webclient.warLocation=webapps/webclient-server.war
webclient.configFile=<path to WebClient's webclient.config file>
webclient.tempDirBase=<path to WebClient's tmp folder>
webclient.rootDir=<path to WebClient root>
```

Note - Tomcat should be executed from its root folder, otherwise the path to *webclient.war* needs to be adjusted.

If Tomcat is running on headless Linux/Unix, then it must be started along with the Xvfb framework, with acommand like:

```
#!/bin/bash
/etc/service/xvfb/run &
catalina.sh run
```

WebClient Admin Console

The approach described above is applicable also the the WebClient Admin Console.

1. Make a copy of *webclient-admin-server.war* from the isCOBOL SDK to Tomcat's *webapps* folder
2. In *webclient-admin.properties*
   a. set the property webclient.server.websocketUrl to

```
ws://localhost:<port>
```

   with the port that Tomcat is running on
   b. set the *webclient.connection.secret* property to match the setting in *webclient.properties*
   c. set *webclient.server.websocketUrl* to the WebClient server's websocket URL (i.e. ws://localhost:8080/webclient-server)
3. Configure the Admin Console URL in *webclient.config*, i.e.:

```
"/": {
      ....,
      "adminConsoleUrl" : "http://localhost:8080/webclient-admin-server"
 }
```

Known limitations and differences between WebClient and Thin Client

This chapter lists the features that are currently not supported by WebClient as well as behaviors that are different between running as a standard COBOL application and running as a web application.

The list is updated to the date this document has been written.

Most of these differences and limitations are related to the more complex architecture required for the WebClient. In a Thin Client environment, there are only two machines involved: the user's PC and the application server, both using isCOBOL products. But in a WebClient environment, there are three machines involved, the web server, the web client, and the application server. The machine previously known as the user's PC becomes a web server, with no isCOBOL products installed. Instead the PC uses a web browser to interact with the web server. This means that when the COBOL application looks for client resources, it will find the web server’s resources, not the resources on the end user’s PC.

Printing

By default there is only one printer available, its name is “WebPrintService” and it’s a PDF printer. When a print operation is performed on WebPrintService, the browser automatically opens the resulting PDF in a new tab at the end of the print job. This is the suggested way of dealing with print jobs in WebClient environment. In the rare case your application needs to interact with the printers installed on the web server, enable Allow Server Printing in the configuration of the application.

The WebPrintService printer is not recognized as default printer by the Win$Printer functions that return printer information (e.g. WINPRINT-GET-CURRENT-INFO).

Library Routines

Unless differently specified in the library routine documentation, every routine that access client resources in a WebClient environment works on the server where the WebClient service is running and not on the end user PC where the web browser is running. This rule applies to routines called via CALL CLIENT as well as routine functions that access to the client machine (e.g. C$COPY when one of the parameters start with "@[DISPLAY]:").

The C$DESKTOP and C$EASYOPEN routines trigger the download of the file instead of opening it with the associated application.

The J$GETFROMLAF routine is not supported. Calling it will return unpredictable results.

The W$MENU routine is not able to manage the tray icon.

The $WINHELP routine is not supported. Calling it may cause a crash of the application.

The C$OPENSAVEBOX routine behavior is affected by the following configuration entries: Isolated Filesystem, Uploading Files, Deleting Files, Downloading Files and Auto-Download from Save Dialog.

The W$CAPTURE routine is not supported. Calling it will return unpredictable results.

The WIN$PLAYSOUND routine plays the sound on the web server machine where WebClient is running.

User Interface

The windows decoration is driven by an internal theme and differs from the decoration of your current Java Swing Look & Feel.

The default Web-Browser implementation (DJBrowser) doesn't work. Use the JavaFx implementation by setting *iscobol.gui.webbrowser.class=com.iscobol.fx.JFXWebBrowser* in the COBOL configuration.

The copy and paste of text on character-based screens is not supported.

Function Keys

Function keys are caught by both browser and COBOL application.

If the F5 key is caught by the COBOL program, then the browser will not refresh the page.

Debug

In order to debug a program running under WebClient, the Remote Debugger should be used.

Set *iscobol.rundebug* to "1" or "2" in the COBOL configuration and ensure that the classes loaded by the isCOBOL Server are compiled in debug mode.

Start the application in your web browser.

Launch the Debugger on your PC, the same where you're executing the browser and connect it to the port where the Remote Debugger is listening (usually 9999) on the machine where isCOBOL Server is running.

For more information about remote debugging, see Remote Debugging.

Using the application from mobile devices

Mobile devices like smartphones and tablets include web browser applications and therefore are suitable to use a COBOL application via WebClient.

When developing applications that could run on mobile devices, developers should keep in mind the restrictions on display size. This means that windows may not fit in the available space, and windows cannot be dragged on mobile browsers.

A suggestion could be do open windows maximized, and eventually using the LM-RESPONSIVE layout manager.

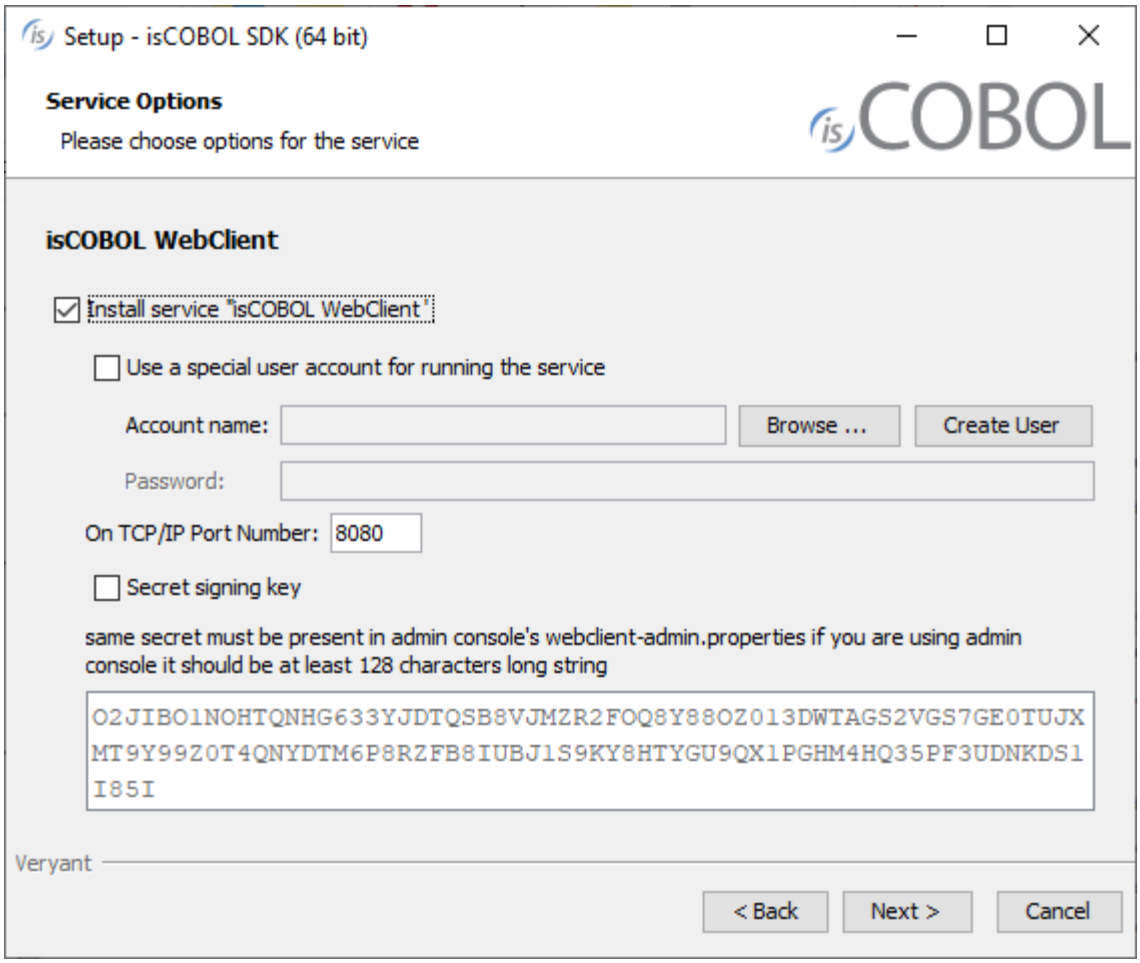
When the virtual keyborad appears over a maximized window, the NTF-RESIZED event is fired.

Windows service and Unix daemon

Windows service

On Windows it's possible to install isCOBOL WebClient and his Admin Console as Windows services.

These services can be installed during the setup process:



When isCOBOL has been installed, the service can be installed, removed and managed through the webclient.exe and webclient-admin.exe command line utilities.

webclient.exe and webclient-admin.exe usage

The service maintenance is done through webclient.exe and webclient-admin.exe.

You must have administrator privileges in order to run these commands.

Install

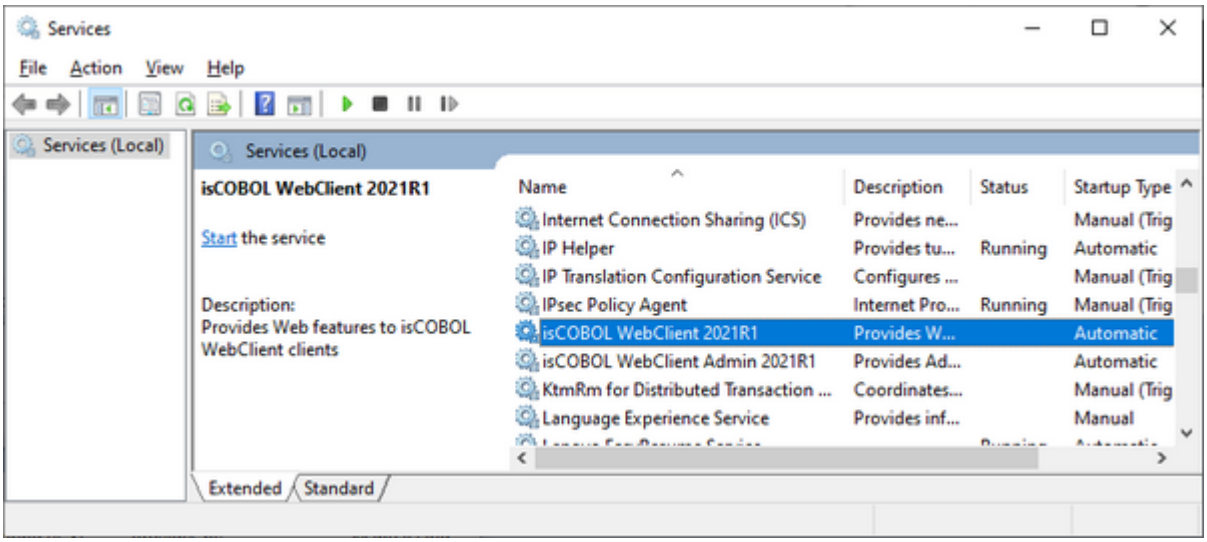
To install the WebClient service, use the command:

```
webclient -install
```

To install the WebClient Admin Console service, use the command:

```
webclient-admin -install
```

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



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

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

```
webclient -install-demand
```

To install the WebClient Admin Console service in demand mode, use the command:

```
webclient-admin -install-demand
```

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

Status

To retrieve the WebClient service status, use the command:

```
webclient -status
```

To retrieve the WebClient Admin Console service status, use the command:

```
webclient-admin -status
```

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

Start

To start the WebClient service, use the command:

```
webclient -start
```

To start the WebClient Admin Console service, use the command:

```
webclient-admin -start
```

Stop

To stop the WebClient service, use the command:

```
webclient -stop
```

To stop the WebClient Admin Console service, use the command:

```
webclient-admin -stop
```

Uninstall

To uninstall the WebClient service, use the command:

```
webclient -uninstall
```

To uninstall the WebClient Admin Console service, use the command:

```
webclient-admin -uninstall
```

If these command are successful, the services will disappear from the Windows service manager.

Custom name

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:

```
webclient -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 WebClient service named "myservice":

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

Output redirection

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

The isCOBOL WebClient Admin Console service redirects all the console output (stderr and stdout) to two files named *webclient-admin_err.log* and *webclient-admin_out.log*. These files are located in the isCOBOL bin directory, which is the default directory of the service.

Service configuration

WebClient's Java options must be put in the *webclient.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 WebClient service:

```
#memory settings
-Xmx256m
-Xms128m

#configuration
-Discobol.conf=/myapp/myconf

#classpath
-classpath/p .
-classpath/a C:\dev\myclasses.jar
```

The isCOBOL WebClient 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 *webclient.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

The same rules are applicable to the WebClient Admin Console service, whose Java options are specified in the file *webclient-admin.vmoptions* located in the isCOBOL bin directory, which is the default directory of the service.

Unix daemon

On Unix systems, WebClient and WebClient Admin Console can be installed as daemon processes and maintained using the webclient command and the webclient-admin command.

webclient and webclient-admin usage

The webclient command and the webclient-admin command have the following options:

run	Run the service keeping the console busy
start	Run the service without taking the console busy
stop	Stop the service
restart	Restart the service
status	Show the status of the service

You have to be root in order to use these commands.

Daemon configuration

The webclient command and the webclient-admin command look 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.