

isCOBOL Evolve: WebClient

Thin Client inside a web browser



Copyrights

Copyright (c) 2023 Veryant
6390 Greenwich Drive, #225, San Diego, CA 92122, USA

All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution and recompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Veryant and its licensors, if any.

Key Topics

- [Testing the product using the isCOBOL Demo program \(Iscontrolset\)](#)
- [Applications Monitoring and Configuration](#)
- [Known limitations and differences between WebClient and Thin Client](#)
- [Cluster Deployment](#)
- [Test Tool](#)

Introduction

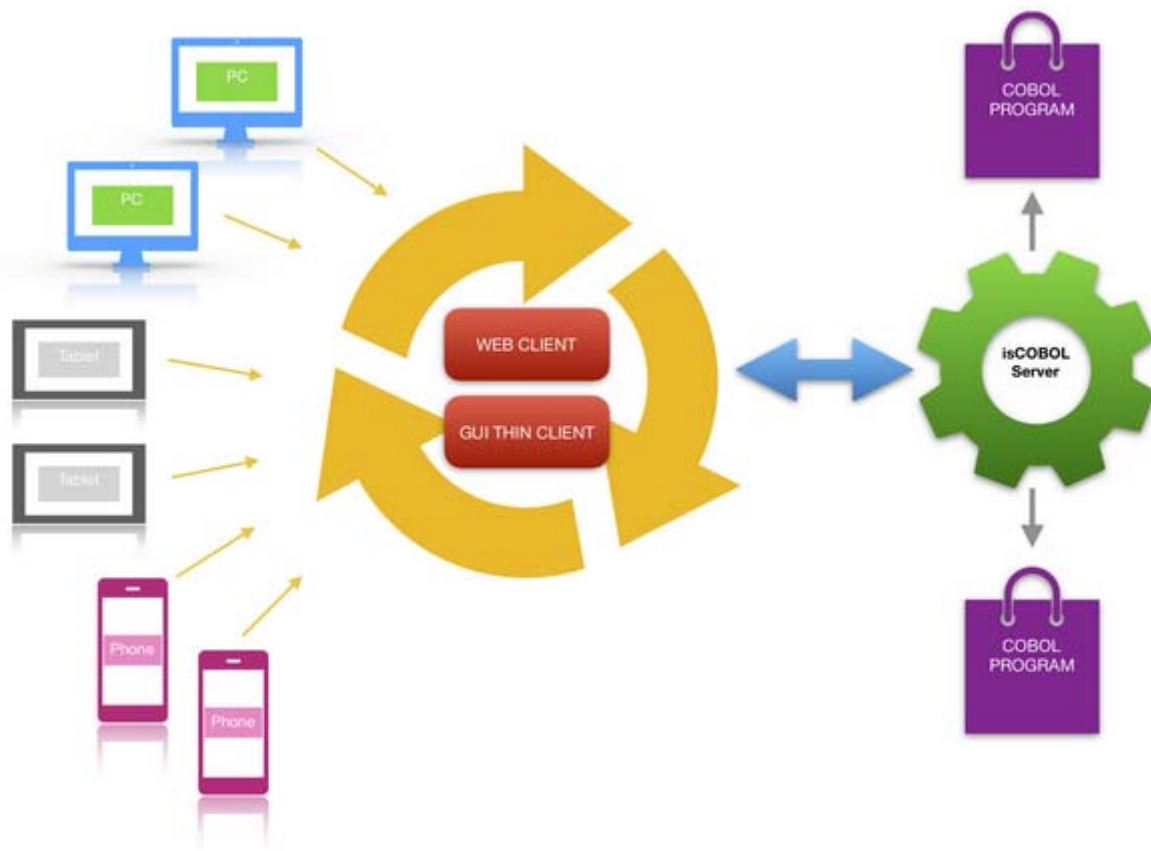
With isCOBOL WebClient your organization can leverage existing COBOL syntax to develop and deploy a universally accessible, zero client, rich Internet application (RIA) using standard COBOL screen sections and existing program procedure flow. No knowledge of object-oriented programming, JavaScript, HTML,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

Linux requirements

In order to run on Linux, WebClient requires the X virtual framebuffer (Xvfb) or a X Window System (X11). Also, be sure that the following libraries are available (names are from Ubuntu repositories - use relevant counterparts for your distribution):

- libxext6
- libxi6
- libxtst6
- libxrender1

Network requirements

The latency threshold of acceptable user experience is around 200ms. Beyond that threshold, the application might be usable but user experience goes down sharply.

The recommended latency is 100ms or less.

The bandwidth is required mainly for images. When [DirectDraw Rendering](#) is disabled, all the user interface is rendered via PNG images, so you might consider to keep [DirectDraw Rendering](#) enabled for networks with low responsiveness.

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 WebClient 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).

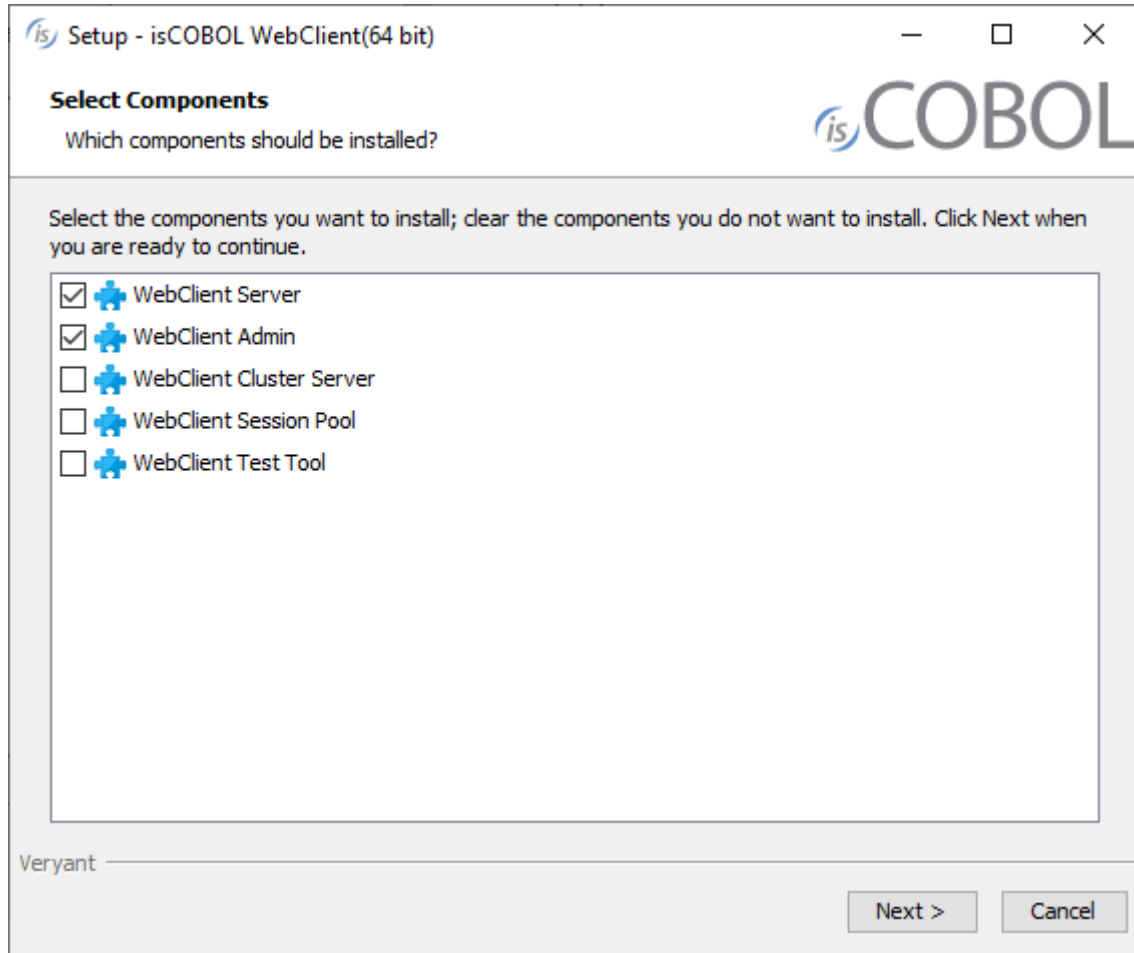
Download and install isCOBOL WebClient 64-bit

Windows

1. If you haven't already done so, [Download and install the Java Runtime Environment \(JRE\)](#).
2. Go to "<https://support.veryant.com>".
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_2023_R1_n_WEBC_Windows.64.msi,

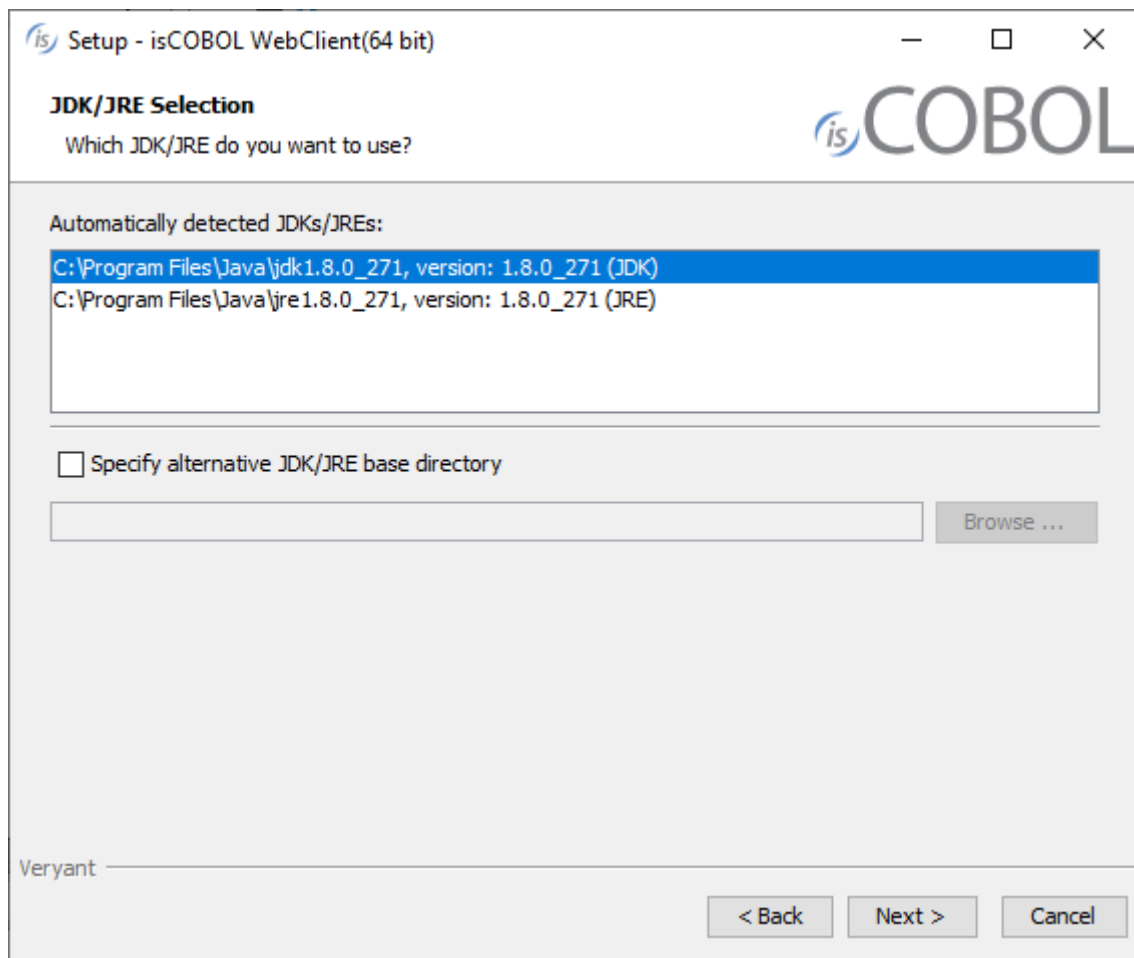
where n is the build number.

6. Run the downloaded installer to install the files.
7. Select the desired items from the list of products when prompted.



Note - By default only WebClient and WebClient Admin are installed.

8. Select your JDK/JRE when prompted



Note - WebClient Test Tool requires Java version 11 or 17. The other products work also with Java version 8. Select the proper Java version depending on the products that you installed and you plan to use.

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](#).
10. You will also be asked if you want to install the WebClient tools as system services or not. If you don't install the services during setup, you will need to start the WebClient components in the foreground or install the services from a command prompt. You will also need to set the secret key manually in the *webclient.properties* and *webclient-admin.properties* files.

Service Options
Please choose options for the service

isCOBOL WebClient

☒ Install service "isCOBOL WebClient"

☐ Use a special user account for running the service

Account name:

Password:

On TCP/IP Port Number:

☐ Secret signing key

same secret must be present in admin console's webclient-admin.properties if you are using admin console it should be at least 128 characters long string

5HYKJAVYEP047ESZCCV6JVNVS MG7NPKAAKNG8D9GR6MZ77TOOOTR2UBZX9FF8
FM3J44FRUGU8QOKNB7CX9391HGRTXKBP90PL4BFIJS5Q4Z79Y4RZ3YGKNC4G1N
4UNU

Veryant

Linux

1. If you haven't already done so, [Download and install the Java Runtime Environment \(JRE\)](#) .
2. Go to "<https://support.veryant.com>".
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_2023_R1_*_WEBC_Linux.64.x86_64.tar.gz
tar -xvf isCOBOL_2023_R1_*_WEBC_Linux.64.x86_64.tar
```


7. Change to the "isCOBOL2023R1" folder and run "./setupwebc", you will obtain the following output:

```
=====
                          isCOBOL WebClient Installation
                          For isCOBOL Release 2023R1
                          Copyright (c) 2005 - 2023 Veryant
=====

Install Components:

  [0] All products..... (no)
  [1] isCOBOL WebClient..... (yes)
  [2] isCOBOL WebClient Admin..... (yes)
  [3] isCOBOL WebClient Cluster..... (no)
  [4] isCOBOL WebClient Session Pool..... (no)
  [5] isCOBOL WebClient Test Tool..... (no)

Install Path:
  [P] isCOBOL parent directory: /home/veryant/veryant

JDK Path:
  [J] JDK install directory: .

[S] Start Install      [Q] Quit

=====
Please press [ 0 1 2 3 4 5 P J S Q ]
```

8. (optional) Type "3", then press Enter to select isCOBOL WebClient Cluster.
9. (optional) Type "4", then press Enter to select isCOBOL WebClient Session Pool.
10. (optional) Type "5", then press Enter to select isCOBOL WebClient Test Tool.
11. (optional) Type "P", then press Enter to provide a custom installation path, if you don't want to keep the default one.
12. Type "S", then press Enter to start the installation.

Other Unix

The WebClient product is qualified only for the Windows and Linux 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, WebClient Admin, WebClient Cluster and WebClient Session Pool 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 following configuration files:

- webclient/webclient.properties
- webclient/admin/webclient-admin.properties
- webclient/cluster/cluster-server/webclient.properties
- webclient/cluster/setssion-pool/webclient-sessionpool.properties

The default setting is:

[illegible]

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 all WebClient tools 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:

```
iscclnt -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.

Use the following command to start the WebClient service:

- Windows

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

- Linux

```
cd $ISCOBOL/bin  
./webccclient
```

Note - the above snippets assume that the ISCOBOL environment variable is set to the isCOBOL WebClient installation directory.

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  
webccclient-admin.exe
```

- Linux

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

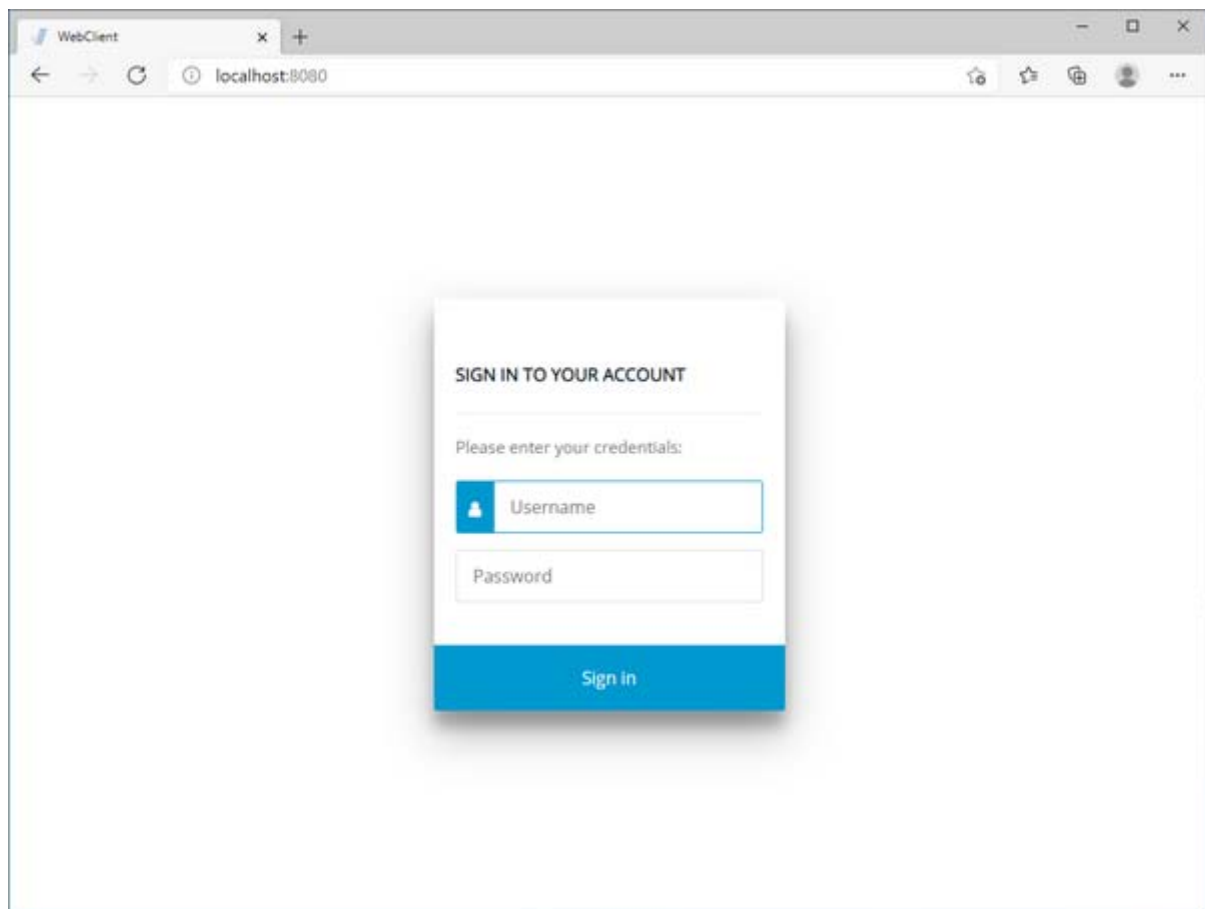
Note - the above snippets assume that the ISCOBOL environment variable is set to the isCOBOL WebClient installation directory.

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

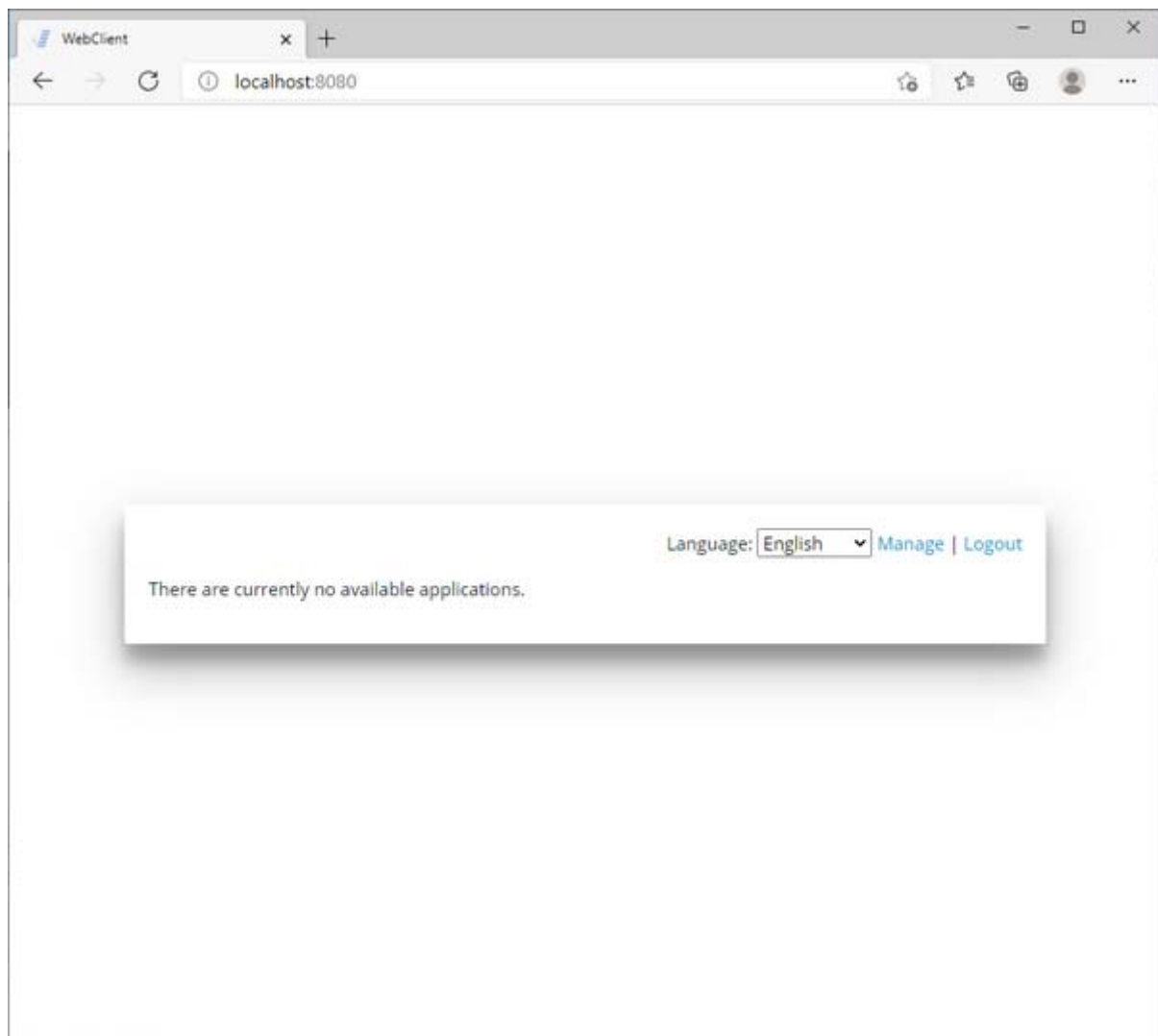
```
INFO:oejs.Server:main: Started @4082ms
```

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.

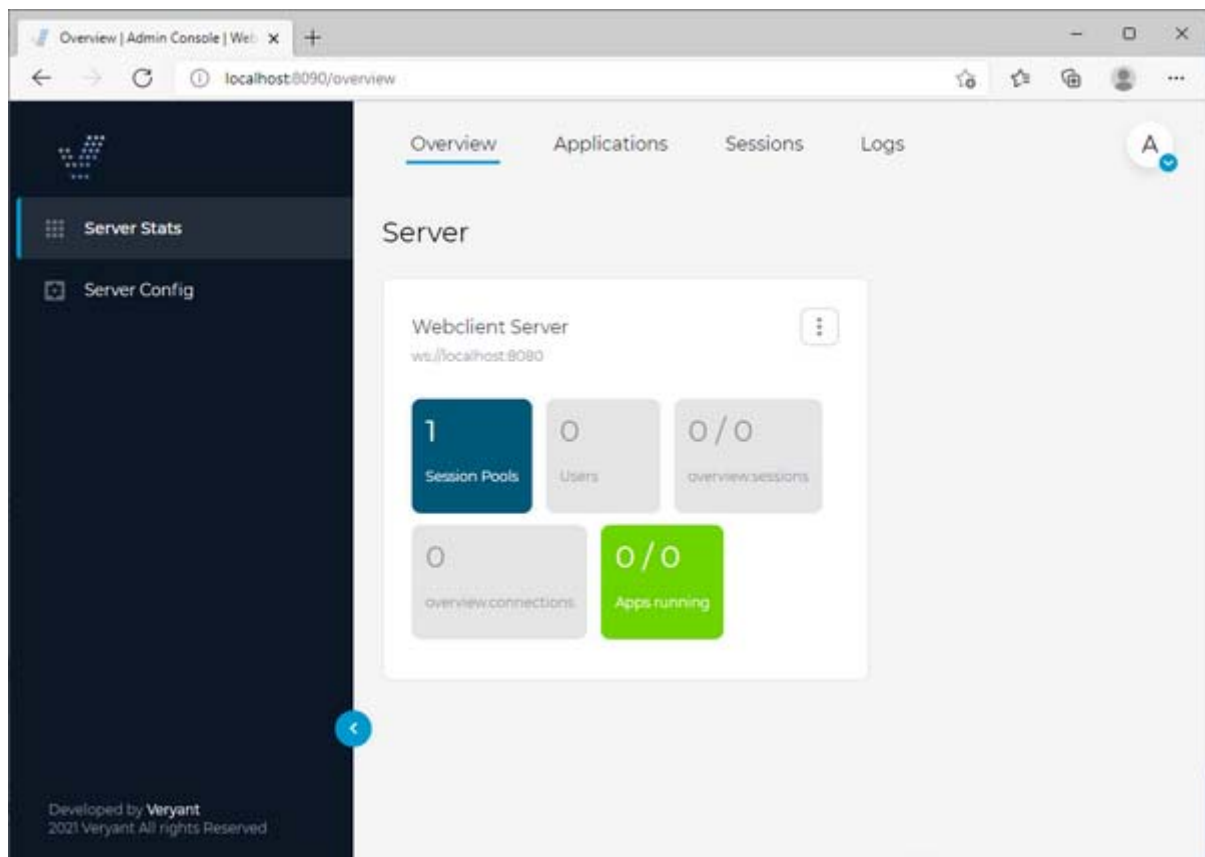
You will get this screen:



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



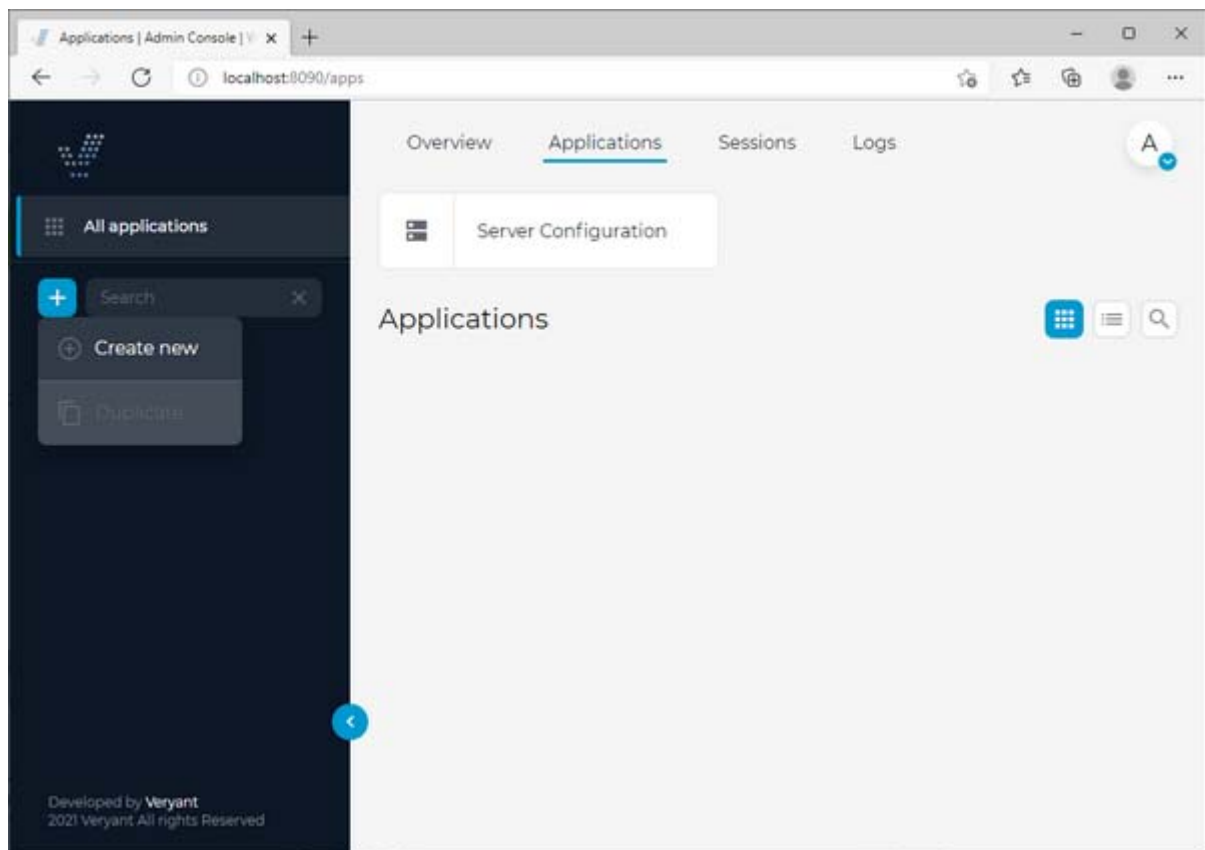
Click on "Manage" and you will get this screen:



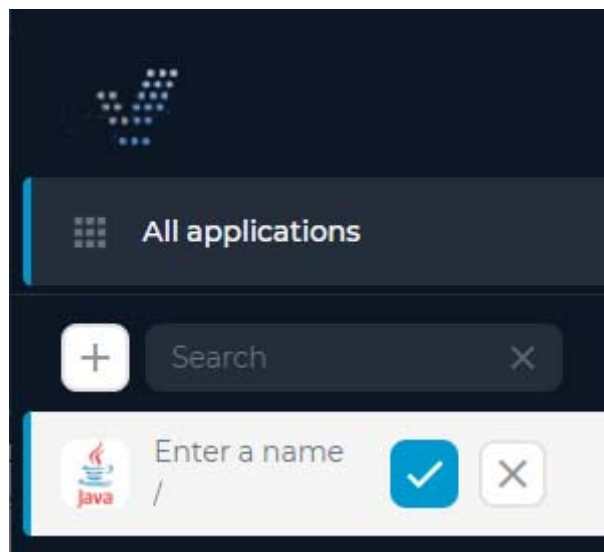
Switch to the *Applications* tab.

The menu on the left shows the list of available applications. This list is currently empty.

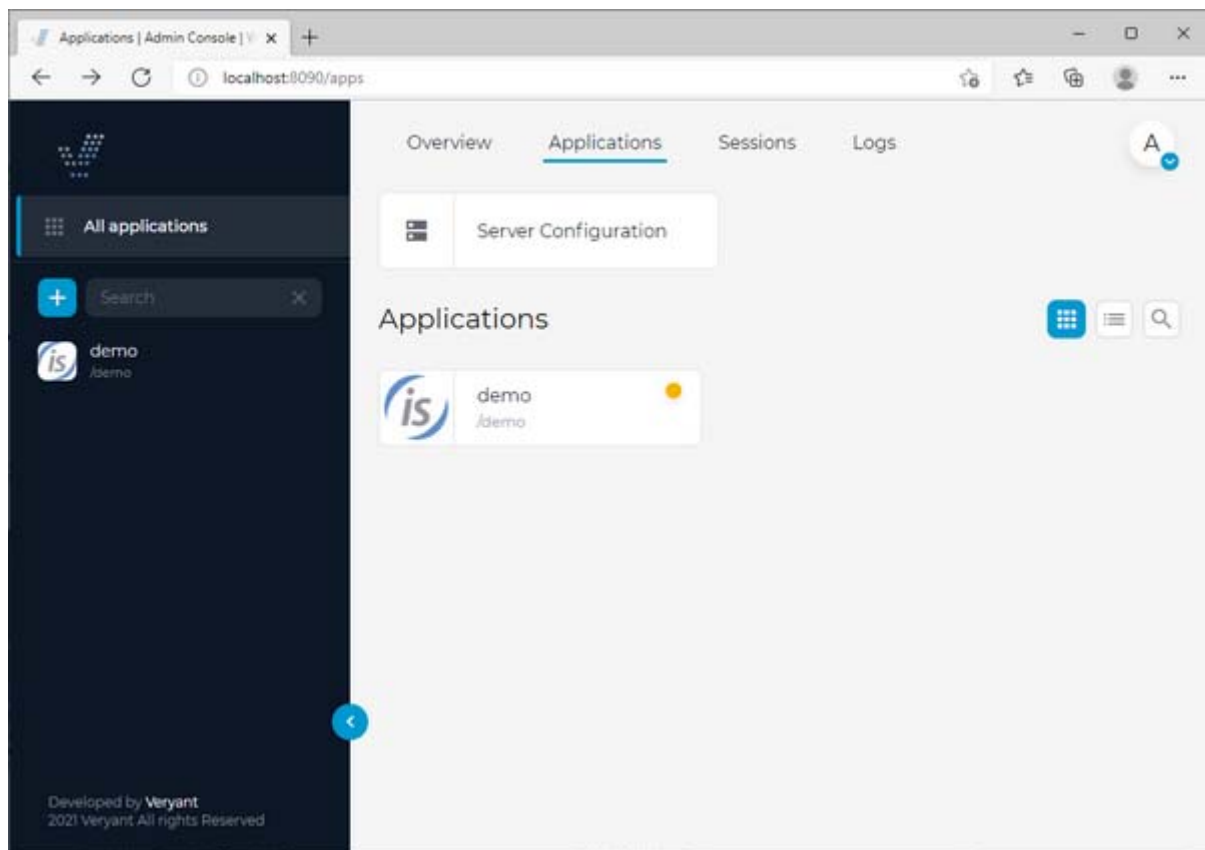
Click the plus button before the search field to show the "Create new" option.



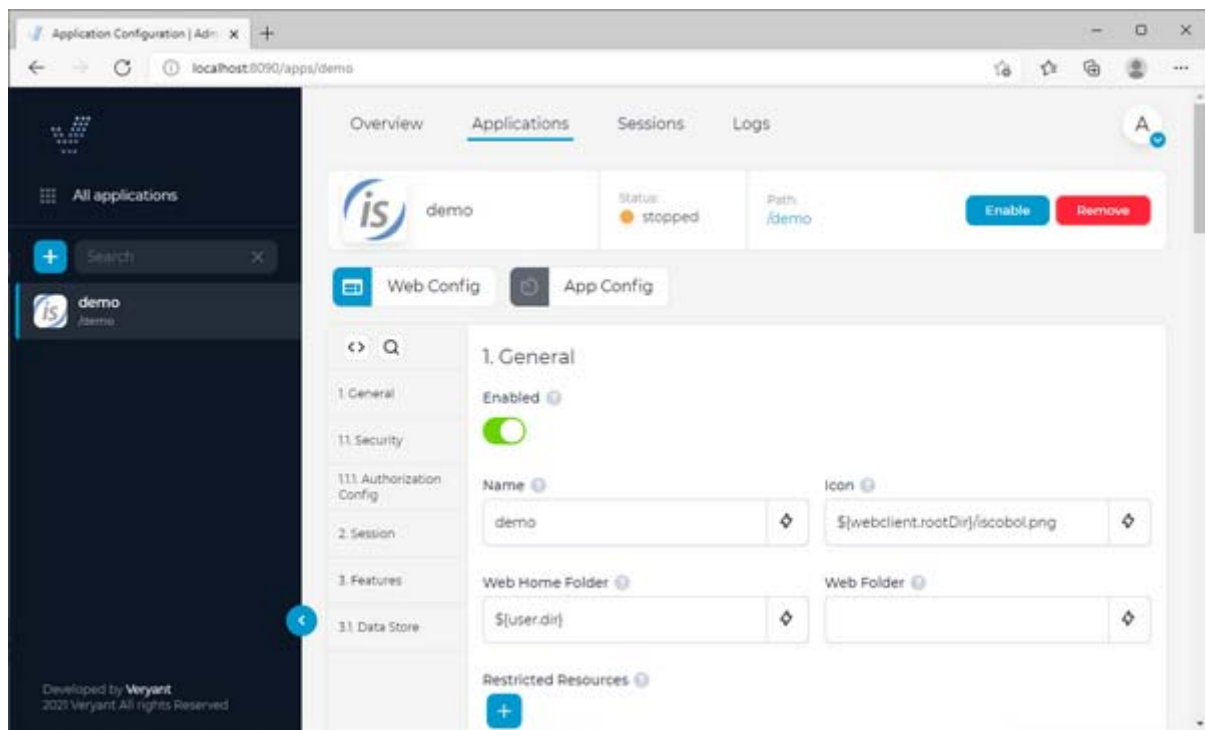
By clicking "Create new" you will be prompted for the application name:



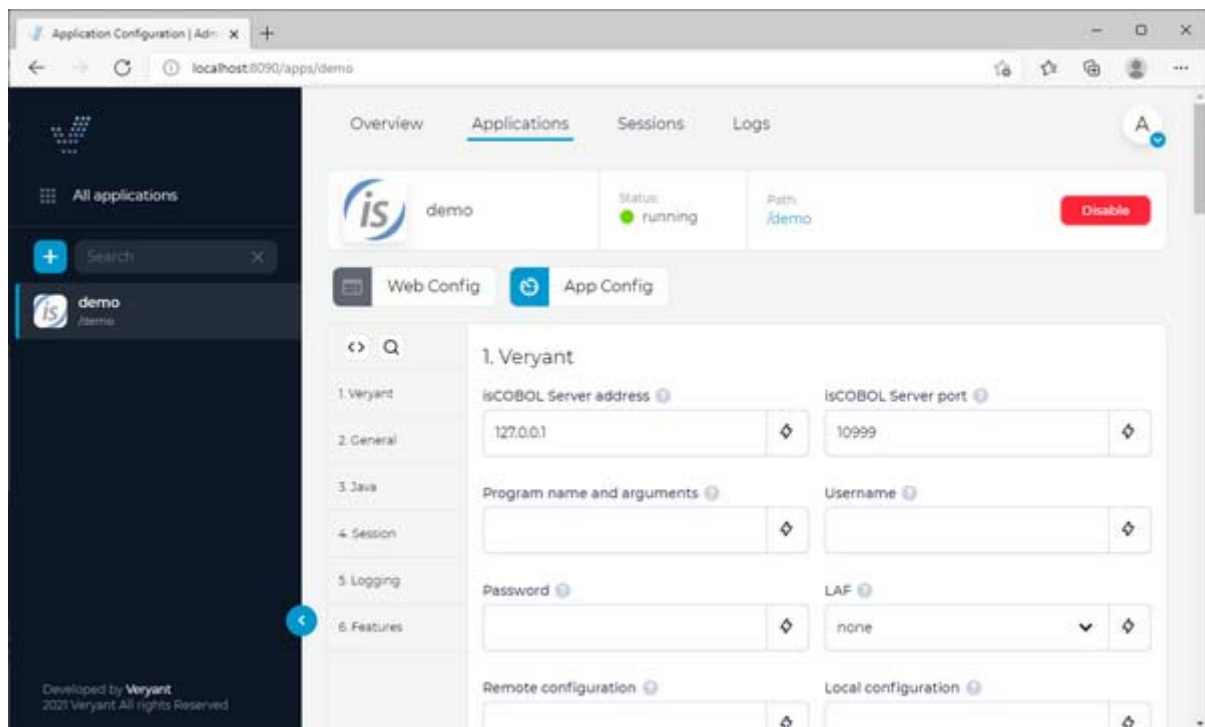
Type "demo" in the field. This is the name that will be used in the URL in order to use the COBOL application.
Click the confirmation button to make the new application appear in the list:



Click on the demo button to reach the application configuration.



Click the *Enable* button in order to activate the application, then switch to the *App Config* section:



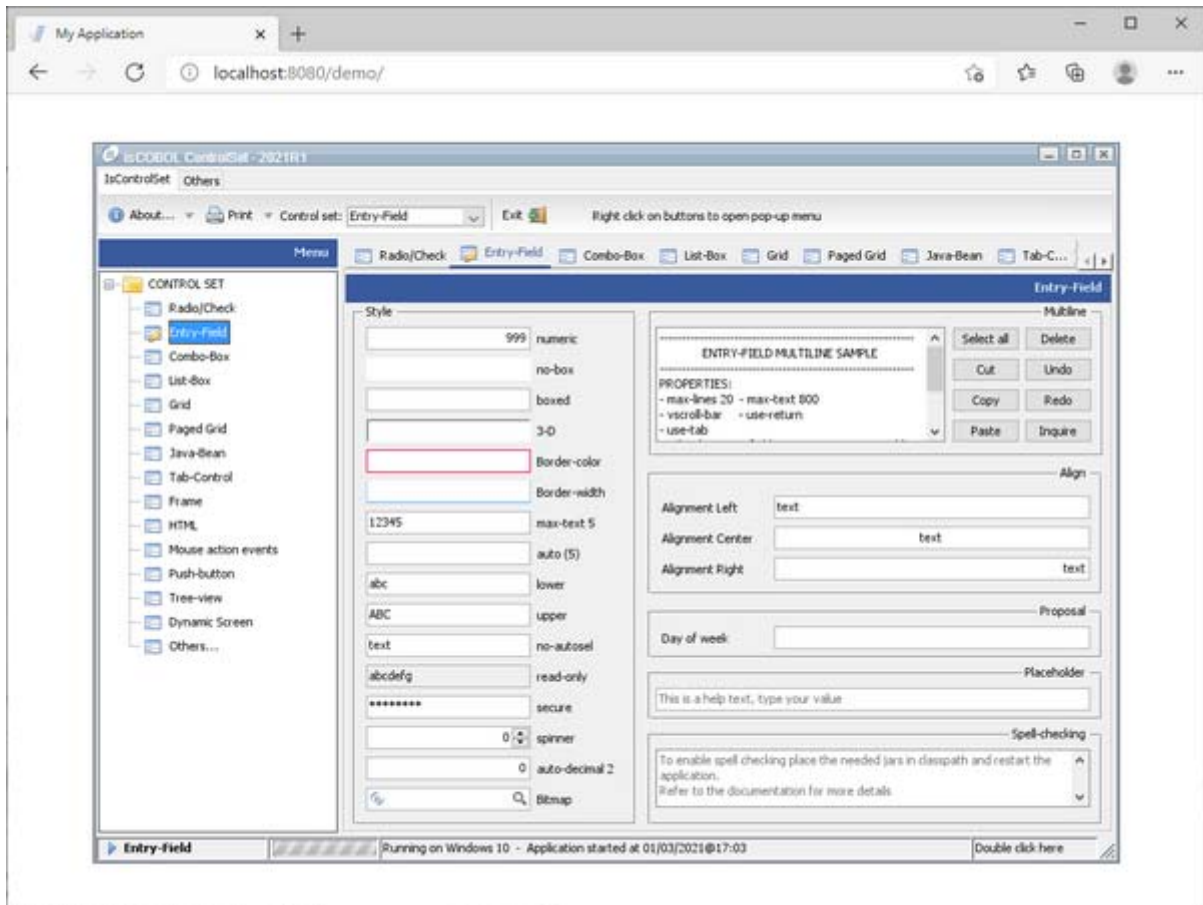
Type "ISCONTROLSET" (upper case) in the *Program name and arguments* field.

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](#).



WebClient commands

The WebClient SDK includes the following commands:

- `webcclient-admin`
- `webcclient-and-admin`
- `webcclient-cluster`
- `webcclient-session`
- `webcclient-testtool`
- `webcclient`
- `webclient-admin`
- `webclient-and-admin`
- `webclient-cluster`
- `webclient-session`
- `webclient-testtool`
- `webclient`

webcclient-admin

The `webcclient-admin` command starts the WebClient Admin Console in foreground mode.

Usage

```
webcclient-admin [options]
```

See [Command-line startup options](#) for the list of available options.

Edit the `webclient/admin/webclient-admin.properties` configuration file to configure the WebClient Admin Console. In particular, you might need to change the `webclient.server.publicUrl` property and the `webclient.server.websocketUrl` property if the WebClient server that you wish to administer is not running on the same machine or is running on a port other than 8080.

webcclient-and-admin

The `webcclient-and-admin` command allows you to start WebClient server and WebClient Admin Console as a single process. This command is particularly suitable for development environments.

Usage

```
webcclient-and-admin [options]
```

See [Command-line startup options](#) for the list of available options.

Before running the command, edit the *webclient/webclient.config* file and replace:

```
"adminConsoleUrl" : "http://localhost:8090"
```

with:

```
"adminConsoleUrl" : "http://localhost:8080/admin"
```

If you need to configure anything related to the admin application (i.e. a different secret key or a public URL different than localhost), edit the *webclient/admin/webclient-admin.properties* file.

webcclient-cluster

The *webcclient-cluster* command starts the WebClient Cluster Server in foreground mode.

Usage

```
webcclient-cluster [options]
```

See [Command-line startup options](#) for the list of available options.

See [Cluster Deployment](#) for more information on this argument.

webcclient-session

The *webcclient-session* command starts the WebClient Session Pool in foreground mode.

Usage

```
webcclient-session [options]
```

See [Command-line startup options](#) for the list of available options.

Edit the *webclient/cluster/session-pool/webclient-sessionpool.properties* configuration file to configure the WebClient Admin Console. In particular, you might need to change the *webclient.server.websocketUrl* property if the WebClient Cluster Server that you wish to attach is not running on the same machine or is running on a port other than 8080.

See [Cluster Deployment](#) for more information on this argument.

webcclient-testtool

The *webcclient-testtool* command starts the WebClient Test Tool in foreground mode.

Usage

```
webcclient-testtool [options]
```

See [Command-line startup options](#) for the list of available options.

webcclient

The *webcclient* command starts the WebClient server in foreground mode.

Usage

```
webcclient [options]
```

See [Command-line startup options](#) for the list of available options.

webclient-admin

The webclient-admin command allows you to install and manage the WebClient Admin Console as a service on Windows and as a daemon on Linux.

Usage

```
webclient-admin [options]
```

For more information see [Windows services and Unix daemons](#).

webclient-and-admin

The webclient-and-admin command allows you to install and manage the Web Client and WebClient Admin Console as a single service on Windows and as a single daemon on Linux.

Usage

```
webclient-and-admin [options]
```

For more information see [Windows services and Unix daemons](#).

webclient-cluster

The webclient-cluster command allows you to install and manage the WebClient Cluster Server as a service on Windows and as a daemon on Linux.

Usage

```
webclient-cluster [options]
```

For more information see [Windows services and Unix daemons](#).

webclient-session

The webclient-session command allows you to install and manage the WebClient Session Pool as a service on Windows and as a daemon on Linux.

Usage

```
webclient-session [options]
```

For more information see [Windows services and Unix daemons](#).

webclient-testtool

The webclient-testtool command allows you to install and manage the WebClient Test Tool as a service on Windows and as a daemon on Linux.

Usage

```
webclient-testtool [options]
```

For more information see [Windows services and Unix daemons](#).

webclient

The webclient command allows you to install and manage the WebClient server as a service on Windows and as a daemon on Linux.

Usage

```
webclient [options]
```

For more information see [Windows services and Unix daemons](#).

Command-line startup options

The WebClient commands support the following command line options:

Option	Description	Default value
-c <arg>	Configuration file name. A relative path is resolved from the WebClient's bin directory	webclient/webclient.config in the WebClient installation directory
-d <arg>	Create a new temp folder for every instance	false
-h <arg>	Local interface address where the web server will listen	0.0.0.0
-kp <arg>	Keystore password	
-ks <arg>	Keystore file location for SSL configuration	
-p <arg>	HTTP port where the web server will listen. If 0 HTTP is disabled.	8080
-s <arg>	HTTPS port where the web server will listen. If 0 HTTP is disabled.	
-t <arg>	The temp folder will be created for the WebClient server. A relative path is resolved from the WebClient's bin directory	webclient/tmp in the WebClient installation directory
-tc <arg>	Clean the temp folder before WebClient start	true
-tp <arg>	Truststore password	
-ts <arg>	Truststore file location for SSL configuration	

Option	Description	Default value
-id <arg>	Server id (visible in admin console)	
-ctx <arg>	Context path where WebClient is deployed.	Random uuid
-pf <arg>	Properties file name. A relative path is resolved from the WebClient's bin directory	webclient/webclient.properties in the WebClient installation directory

Note - hostname, ports and SSL configuration defaults are inherited from the jetty.properties configuration file and can be overridden by these command-line options. The values listed in this table reflect the content of the default jetty.properties file installed with isCOBOL and might be different if you edited that file.

Example

In order to start WebClient on the HTTP port 12345 using C:\Develop\mywebclient.config as configuration file, use the command:

```
webcclient -p 12345 -c C:\Develop\mywebclient.config
```

Startup configuration properties

The following properties can be specified on the command-line to customize the pathname of the logs directory and the pathname of the tmp directory:

Property	Description
webclient.logsDir	<p>Specifies the directory where log files will be created. See Logging for the list of log files generated by WebClient commands.</p> <p>This property takes a pathname in the form of a string and requires the file separator character at the end.</p> <p>Example of valid setting:</p> <pre>webclient.logsDir=/tmp/logs/</pre> <p>If the directory doesn't exist, it's automatically created.</p>
webclient.tempDirPath	<p>Specifies the directory where temp files will be created.</p> <p>This property takes a pathname in the form of a URI and requires the file separator character at the end.</p> <p>Example of valid setting:</p> <pre>webclient.tempDirPath=file:/tmp/</pre> <p>If the directory doesn't exist, an error occurs.</p>

You can pass the options directly on the command-line by prefixing them with -J-D.

Example

In order to start WebClient having log files created under C:\Temp\Logs, use the command:

```
webcclient -J-Dwebclient.logDir=C:\Temp\Logs
```

Alternatively, only on Windows, you can add these options in the vmoptions file of the desired WebClient command.

Example

Content of *webclient.vmoptions* to have log files created under C:\Temp\Logs:

```
# isCOBOL WebClient setting added from isCOBOL WEBC2023.1 setup
# Comments in .vmoptions files are prefixed by a hash, each VM option is on
# a separate line
# -Discobol.conf=myconf.properties
#-Xmx256m
#-Xms128m
-classpath/p .
-XX:+HeapDumpOnOutOfMemoryError
-Dwebclient.logDir=C:/Temp/Logs/
```

Windows services and Unix daemons

Windows service

On Windows it's possible to install WebClient SDK tools as Windows services.

These services can be installed during the setup process:

When WebClient has been installed, the services can be installed, removed and managed through the following commands: [webclient-admin](#), [webclient-and-admin](#), [webclient-cluster](#), [webclient-session](#) and [webclient](#).

Each command provides the following options:

Option	Action
-install	Installs the service in auto mode. The service will start automatically along with the operating system.
-install-demand	Installs the service in demand mode. The service must be started by the user.
-start	Starts the service.
-status	Returns the status of the service. The possible exit codes are: 0 = the service is running 1 = the status cannot be determined 3 = the service is not running

Option	Action
-stop	Stops the service.
-uninstall	Removes the service.

Example

To install the WebClient service, start it and check if it's running, use these commands:

```
webclient -install
webclient -start
webclient -status
```

Note - be sure to have administrator privileges, or these commands will fail.

Non-interactive mode and 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 the command line for all the options. For example, the following list of commands manages an isCOBOL WebClient service named "myservice":

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

Output redirection

The WebClient services redirect all the console output (stderr and stdout) to two files named `<command>_err.log` and `<command>_out.log`. These files are located in the bin directory, which is the default directory of the service.

Service configuration

Command-line Java options must be put in the `<command>.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 WebClient service:

```
#memory settings
-Xmx256m
-Xms128m

#configuration
-Discobol.conf=/myapp/myconf

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

The WebClient service inherits the Classpath from the system and adds all jar libraries in the 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.

Unix daemon

On Linux systems, WebClient tools can be installed as daemon processes and maintained using the following commands: [webclient-admin](#), [webclient-and-admin](#), [webclient-cluster](#), [webclient-session](#) and [webclient](#)

Each command provides the following options:

Option	Action
restart	Stop and start again the daemon in background mode.
run	Run the daemon in foreground mode.
start	Run the daemon in background mode.
status	Show the daemon status.
stop	Stop the daemon.

Daemon configuration

WebClient tools on Linux look for two files located in the bin directory:

- *default_java.conf* includes the paths to the WebClient and Java installations:

Variable	Description
ISCOBOL	WebClient installation directory
ISCOBOL_JDK_ROOT	JDK installation directory
ISCOBOL_JRE_ROOT	JRE installation directory

- `<command>.options` includes environment variables that can alter the command behavior:

Variable	Description
<code>PREFIX_HOME</code>	Home directory of the command
<code>PREFIX_OPTS</code>	Command line options for the command. Use a space to separate multiple options. When the options string includes spaces, delimit it with quotes, for example: <code>WEBCIENT_OPTS="-p 1234 -c mywebc.config"</code>
<code>PREFIX_JAVA_EXE</code>	Alternative JVM for the command
<code>PREFIX_JAVA_OPTS</code>	Java options for the command. Use a space to separate multiple options. When the options string includes spaces, delimit it with quotes, for example: <code>WEBCIENT_JAVA_OPTS="XX:+HeapDumpOnOutOfMemoryError -Xmx4G"</code>
<code>PREFIX_LOG_FILE</code>	File where the command output is redirected
<code>PREFIX_PID_FILE</code>	Semaphore file to inform if the command is running or not. This file contains the process ID.

Depending on the `<command>.options` file, `PREFIX` can be:

- `WEBCIENT` for `webclient`
- `WEBCIENT_ADMIN` for `webclient-admin`
- `WEBCIENT_CL` for `webclient-cluster`
- `WEBCIENT_SP` for `webclient-session`

These files are generated by the setup process.

In these files, comments are prefixed by a hash and each entry is on a separate line.

Environment variables in `<command>.options` can alternatively be set also in the external environment through the `export` command. The external environment has priority over the content of `<command>.options`.

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. Alternatively you can move this file outside the isCOBOL SDK installatio folder and point it via the `-c <arg>` command line option.

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](#)

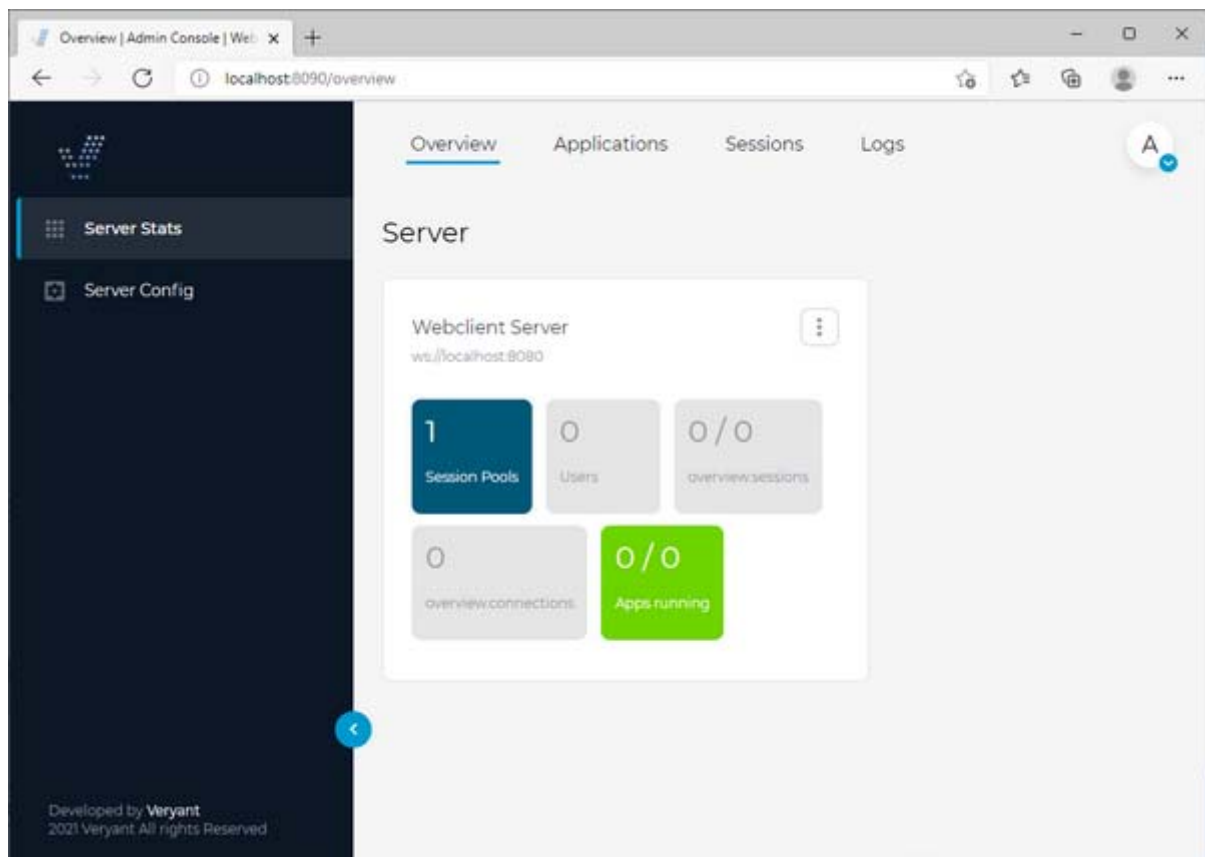
[Applications](#)

[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.



Server Config

The Server Config section allows you to configure the access to WebClient's home page and admin functions as well as provide default settings for new WebClient applications.

Some of these settings are duplicated in the Application configuration section and can be overwritten at the application level.

1. General

Entry	Meaning
Admin Console Url	URL of the Admin Console. This URL will be referenced by the Manage hyperlink in WebClient's home page.

1.1 Security

Entry	Meaning
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	NONE - No authentication is required to access this application. It's not good practice to have no authentication for admin functions, especially in production environments. If you wish to disable authentication anyway, do the following: <ol style="list-style-type: none">1. set <i>Security Module Name</i> to NONE2. In <i>1.1.2 Security Module Config - Extension</i> click the + button and select "AccessMapping" from the dropdown menu3. In <i>1.1.2.1. accessmapping</i> click the + button, choose "admin" from the dropdown and toggle the "Everyone?" radio button EMBEDDED - User authentication is required. 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. This is the default. The rules to configure this field are described in Configuring Users .
Security Context per Tab	Activates a separate security context for each browser tab

1.1.1. Security Module Config - General

Entry	Meaning
Users	Embedded users. By default only the admin user is present. See Configuring Users for information about how to add new users.
Logout URL	URL where you're redirect at logout. By default, WebClient's home page is used.

1.1.2. Security Module Config - Extension

Entry	Meaning
Extensions	List of security extensions enabled

2. Session

Entry	Meaning
Max. Connections Per User	Maximum number of simultaneous connections per user. The default value -1 stands for no limit.

3. Features

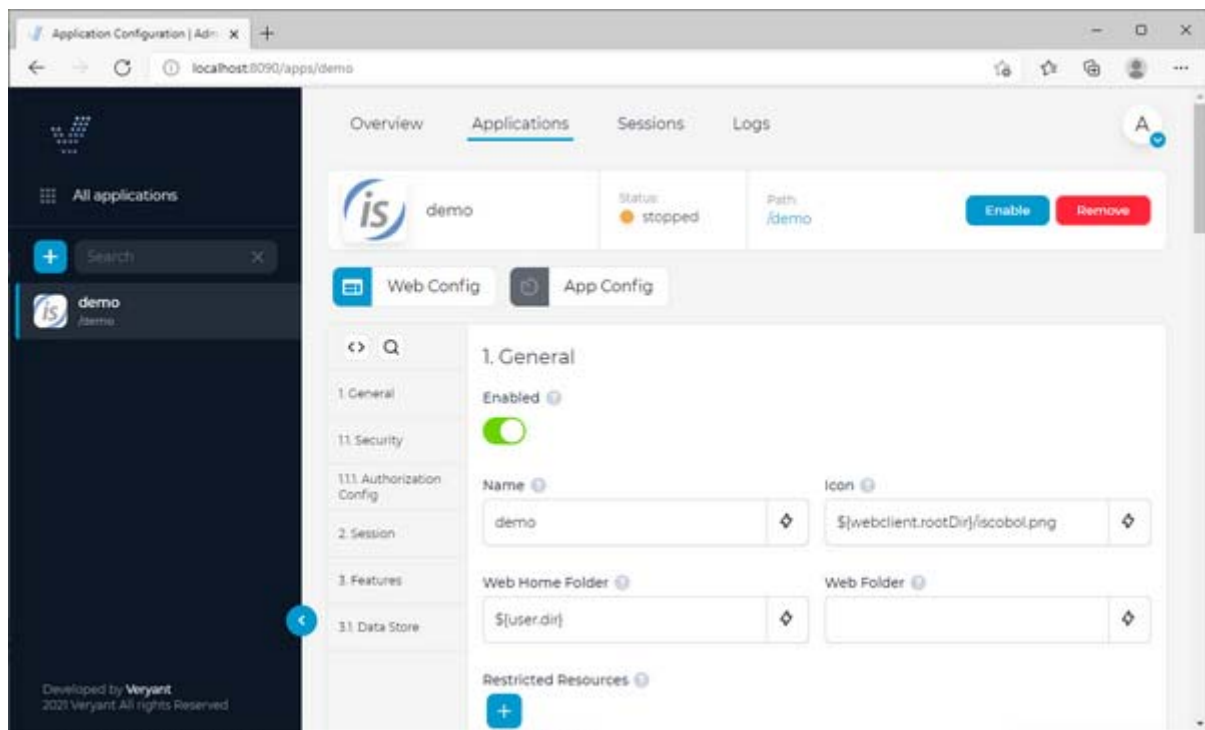
Entry	Meaning
Allow Session Recording	ON - The admin can record user actions to a video OFF - User actions can't be recorded
Allow Session Mirroring	ON - The admin can monitor user actions and take control if necessary OFF - The admin can't see what the user is doing on the application screen

3.1 Data Store

Entry	Meaning
Data Store Module Class Path	Reserved for future use
Data Store Module Name	Reserved for future use
Direct Transfer	Reserved for future use

Applications

The Applications section allows you to alter the settings of a specific application as well as create a brand new application.



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.

Configuration entries are distributed on two pages: [Web Config](#) and [App Config](#).

Web Config

1. General

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.
Name	WebClient application name. This name will be displayed on the left panel and used to alphabetize the list of WebClient applicaitons
Icon	Image icon shown in the application selection dialog
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	Defined Path-prefix restricts access to resources only to authenticated users. Applies to static resources inside Web Folder or packaged with WebClient
Localization Folder	Folder where customized messages and translations are stored
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

1.1 Security

Entry	Meaning
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
Security Context per Tab	Activates a separate security context for each browser tab

1.1.1. Authorization Config

Entry	Meaning
Authorized Users	List of WebClient users that will be allowed to use this application. See Configuring Users for information on how to create WebClient users

Entry	Meaning
Authorized Roles	List of WebClient roles that will be allowed to use this application. See Configuring Users for information on how to create WebClient roles

2. Session

Entry	Meaning
Session Mode	<p>Define if and how sessions can be restored.</p> <p>ALWAYS_NEW_SESSION - every time the application URL is loaded, a brand new runtime session is started.</p> <p>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.</p> <p>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 <i>Security Module Name</i> is not set to NONE, which would require users to perform authentication each time they open the application</p>
Max. Connections	Limit the maximum number of concurrent sessions for this application. By default, up to 10 concurrent sessions are allowed. A value of -1 means infinite
Max. Connections Per User	Limit the maximum number of concurrent sessions from the same user. If this value is greater than Max. Connections , the value of Max. Connections will be considered
Session Stealing	<p>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.</p> <p>OFF - Users can't resume the WebClient session if the connection is open in other browser</p>
Auto Logout	<p>ON - Users are automatically logged out after the application finished.</p> <p>OFF - Users remain logged in also after the application finished</p>
Goodbye URL	Absolute or relative URL to redirect to, when application exits. Use '/' to navigate back to Application selector.

3. Features

Entry	Meaning
Monitor App Responsiveness	<p>ON - Show a progress animation if Swing's Event Dispatch thread is not responding</p> <p>OFF - Don't advise users if Swing's Event Dispatch thread is not responding</p>
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
Allow Session Recording	<p>ON - The admin can record user actions to a video</p> <p>OFF - User actions can't be recorded</p>
Require Recording Consent	<p>ON - The user must confirm before the admin can record his actions</p> <p>OFF - The user is not notified when the admin records his actions</p>

Entry	Meaning
Allow Session Mirroring	ON - The admin can monitor user actions and take control if necessary OFF - The admin can't see what the user is doing on the application screen
Require Mirroring Consent	ON - The user must confirm before the admin can monitor his actions OFF - The user is not notified when the admin monitors his actions
Upload Size Limit	Maximum size of upload for single file in MB. A value of 0 means no limit. The default value is 5

3.1 Data Store

Entry	Meaning
Data Store Module Class Path	Reserved for future use
Data Store Module Name	Reserved for future use
Direct Transfer	Reserved for future use

App Config

1. Veryant

Entry	Meaning
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
Local configuration	Local configuration file for the application. This is equivalent to the -lc option of the isCOBOL Client

2. General

Entry	Meaning
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	<p>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.</p> <p>Every mapping is composed of two items: <i>Key</i> and <i>Value</i>. <i>Key</i> is the name of the font. <i>Value</i> is the path to the font file. Only True Type (ttf) fonts are supported.</p> <p>If no fonts are defined, necessary fonts are loaded from the system by WebClient as the COBOL application requests them. This rule applies to <i>iscobol.font</i> 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 <i>iscobol.font</i> 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:</p> <p><i>Name: "dialog", Value: "\${user.dir}/fonts/Roboto-Regular.ttf"</i> <i>Name: "dialoginput", Value: "\${user.dir}/fonts/RobotoMono-Regular.ttf"</i> <i>Name: "serif", Value: "\${user.dir}/fonts/RobotoSlab-Regular.ttf"</i></p>
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
Latency Optimized Rendering	When ON, it optimizes the rendering process for connections with high latency
JavaFX Support	ON - Ability to use also JavaFx components. OFF - Only Swing and AWT components allowed.
Enable Debug Mode	Reserved for future use
Enable Test Mode	Allows the Test Tool to record and playback actions on this application

3. Java

Entry	Meaning
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

4. Session

Entry	Meaning
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.

5. Logging

Entry	Meaning
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

6. Features

Entry	Meaning
Isolated Filesystem	ON - the Open and Save dialogs of C\$OPENSABEBOX can only browse the WebClient's Upload Folder and its subfolders. OFF - the Open and Save dialogs of C\$OPENSABEBOX 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\$OPENSABEBOX library routine.
Deleting Files	Enable the ability to delete files from the Open and Save dialogs generated by the C\$OPENSABEBOX library routine.
Downloading Files	Enable the ability to download files through the Save File dialog generated by the C\$OPENSABEBOX 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\$OPENSABEBOX is called.

Entry	Meaning
Transparent Open File Dialog	<p>ON - when C\$OPENSABEBOX is called to open a file, show only the client-side file browser and upload the selected file.</p> <p>OFF - when C\$OPENSABEBOX is called to open a file, show the Open dialog along with the client-side file browser and upload the selected file.</p> <p>Requires Isolated Filesystem on.</p>
Transparent Save File Dialog	<p>ON - when C\$OPENSABEBOX is called to save a file, ask only for the file name.</p> <p>OFF - when C\$OPENSABEBOX is called to save a file, show the Save dialog along with the prompt for file name.</p> <p>Requires Isolated Filesystem on.</p>
Upload Folder	<p>Folder where files uploaded by the user through C\$OPENSABEBOX are stored.</p> <p>Requires Isolated Filesystem on.</p>
Clear Upload Folder	<p>ON - Delete all files in the transfer folder when the application process is terminated.</p> <p>OFF - Don't delete any file when the application process is terminated.</p> <p>Requires Isolated Filesystem on.</p>
Allow JsLink	Reserved for future use
JsLink White List	Reserved for future use
Allow Local Clipboard	<p>ON - Allow access to the end user's PC clipboard.</p> <p>OFF - The user can't cut, copy or paste text in the application screen.</p>
Allow Server Printing	<p>ON - Allow access to the printers installed on the machine where WebClient is running.</p> <p>OFF - Print to PDF using the internal WebPrintService printer and send the PDF to the client browser.</p>
Docking Mode	<p>ALL - All windows can be undocked.</p> <p>MARKED - only windows marked with Dockable interface can be undocked.</p> <p>NONE - disable undocking.</p>

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.

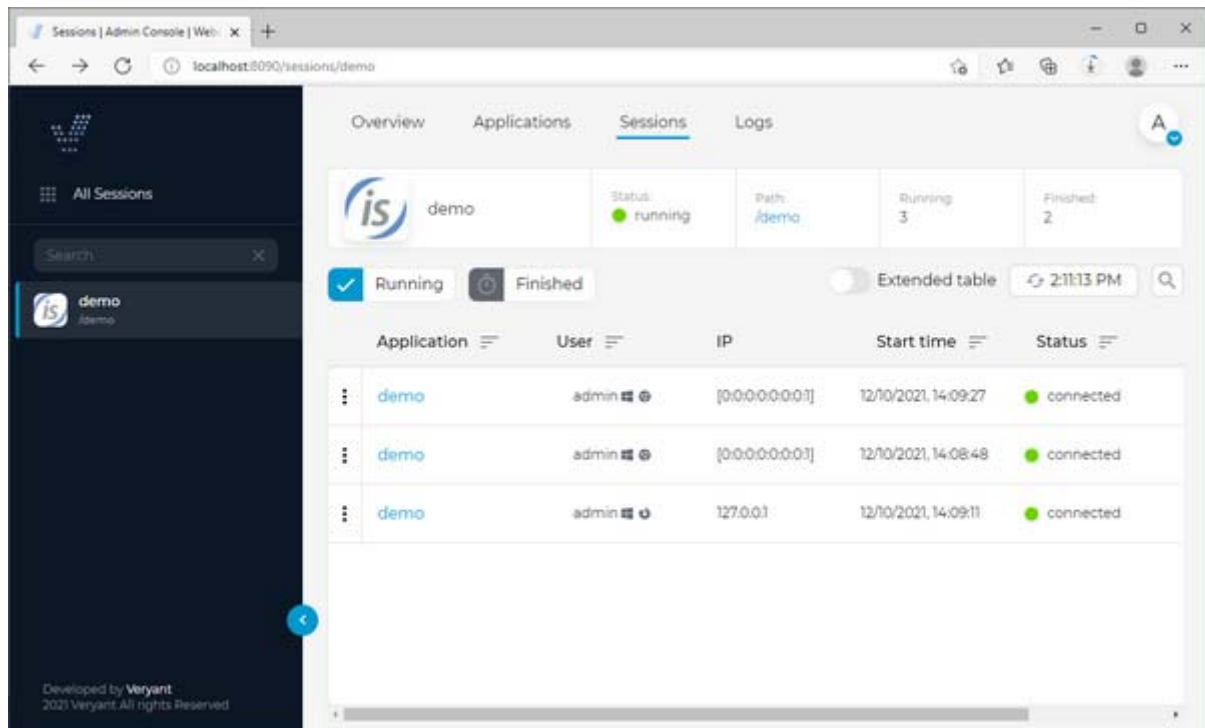
Remove an application

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.

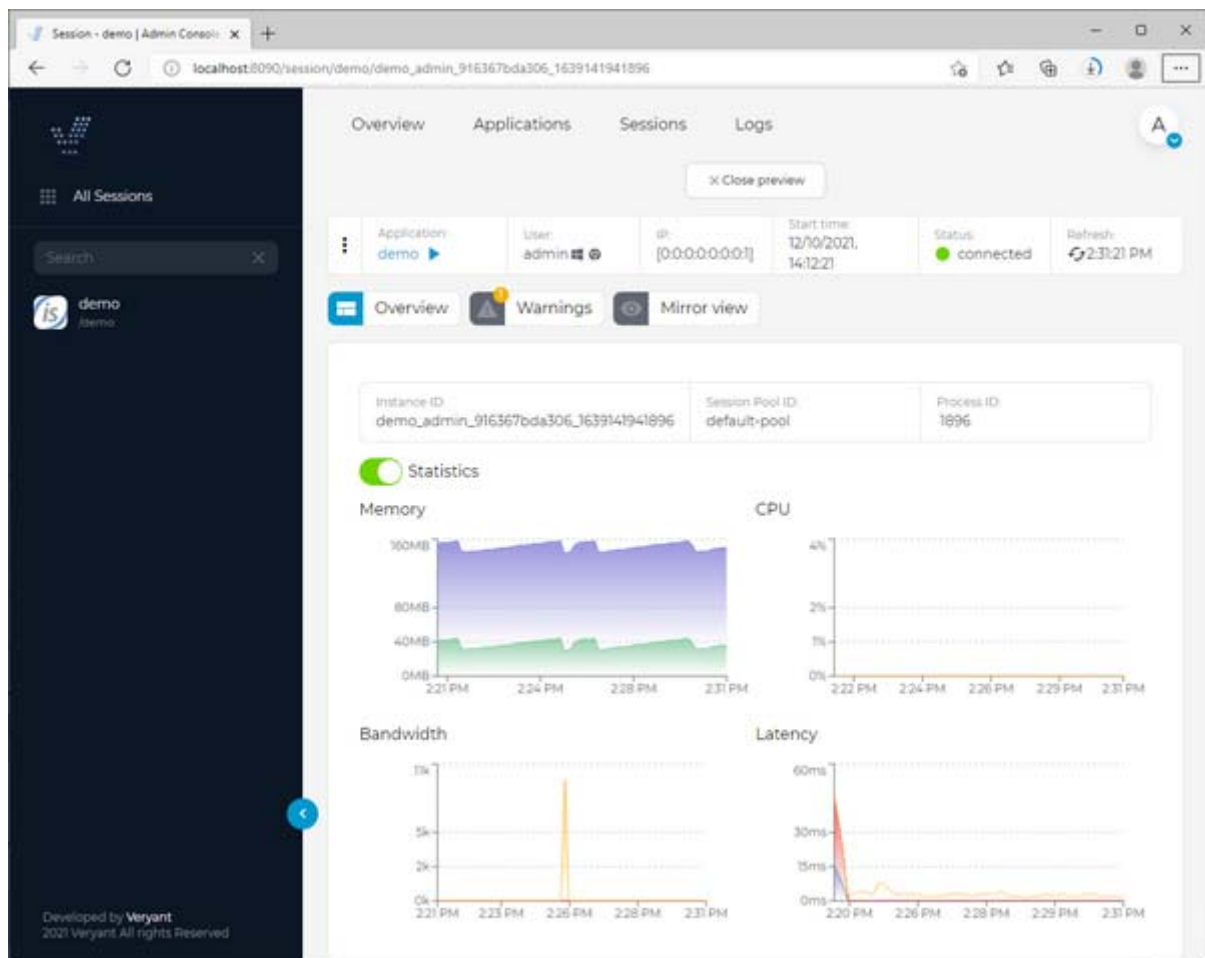
Sessions

The Sessions section lists the sessions of each application.



Click on the application name in the *Application* column to have detailed information on a specific session.

The Session view contains all the details, metrics and options for session management and monitoring. There are also features like mirror view, recording & playback, session logs and warnings.

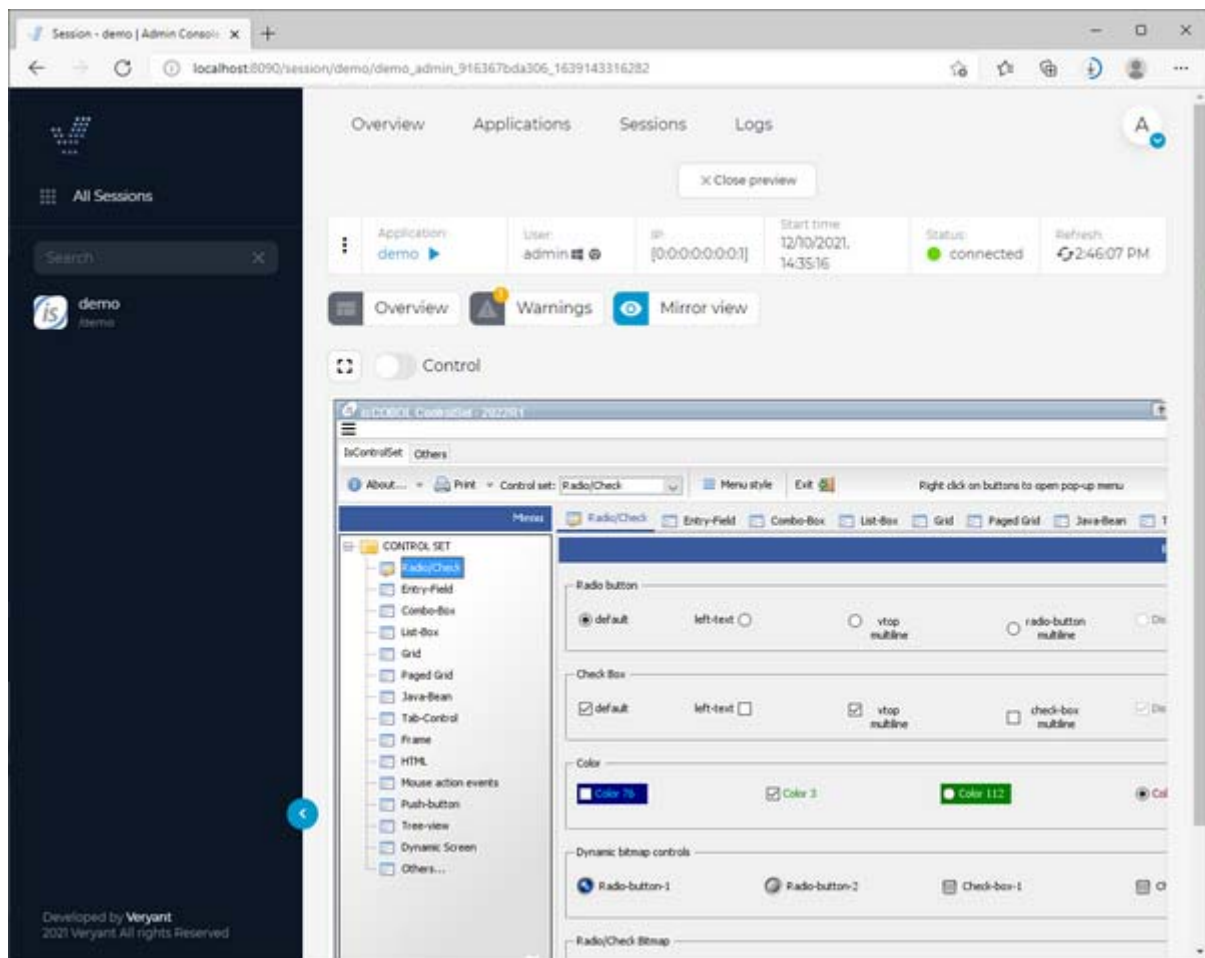


Note - the Process ID value not available when using Java 8.

The context menu provides the following actions:

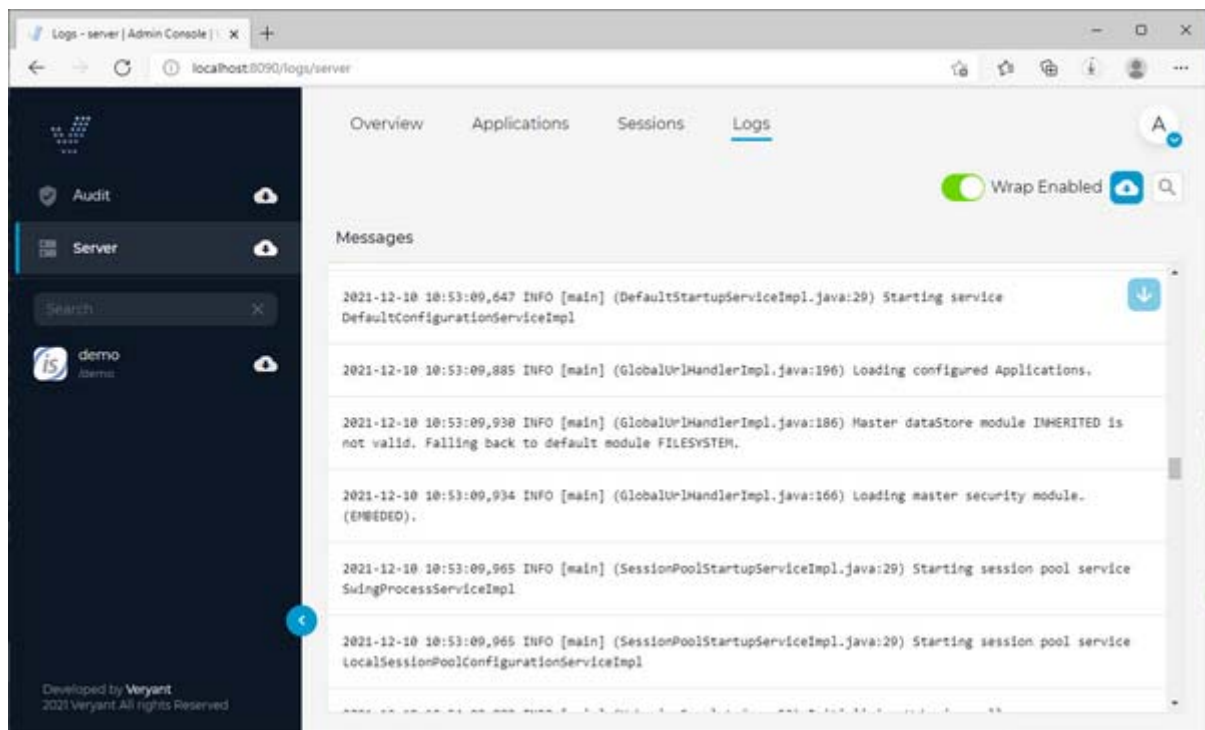
Thread Dump	Take a thread dump of the underlying JVM. The dump can be reviewed in the <i>Warnings</i> page
Record	Start recording the user actions. You can stop recording at any time by clicking the "Stop" button. The recording will automatically stop when the user session terminates. The recorded video will be playable by clicking on the "Play" button after the application name
Copy Inst.ID	Copies the unique session identifier to the clipboard
Shutdown	Kills the session. The JVM is terminated and the user is disconnected.

Click on *Mirror View* to monitor the user activity in real time. You can also take control:



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

1.1. Security

Security Module Class Path 



Security Module Name 




EMBEDDED 



Security Context per Tab 



1.1.1. Security Module Config - General

Users

No.	Username	Password	Roles
 1.	<input type="text"/>	<input type="password"/>	 

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.

Configuring users through property file or JDBC data source

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

- o NONE
- o EMBEDDED

to

- o NONE
- o EMBEDDED
- o org.webswing.security.modules.database.DatabaseSecurityModule
- o 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:

1. add the "database+property" security module to the *Security Module Class Path* as explained in [Configuring users through property file or JDBC data source](#),
2. set the *Security Module Name* field to "org.webswing.security.modules.property.PropertySecurityModule".

A new section named *Security Module Config - General* 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).

1.1.1. Security Module Config - General

File ?	Logout URL ?
<input type="text" value="\$webclient.rootDir/user.properties"/>	<input type="text"/>

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:

1. add the "database+property" security module to the *Security Module Class Path* as explained in [Configuring users through property file or JDBC data source](#),
2. set the *Security Module Name* field to "org.webswing.security.modules.database.DatabaseSecurityModule".

A new section named *Security Module Config - General* appears.

1.1.1. Security Module Config - General

DataSource Class 

DataSource Settings 



Authentication Query 

select password, password_salt from users where us

User Roles Query 

select role_name from user_roles where username =

Permissions Query 

select permission from roles_permissions where roli

Resolve Permissions



Salted Password Hash



Hash Matcher Algorithm 

NONE

Hash Hex Encoded 



Hash Iterations 

1

Logout URL 

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.

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:

\${clientId}	WebClient specific unique browser identifier
\${clientIp}	IP address of browser that started this application
\${clientLocale}	Locale of browser that started this application

<code>\${customArgs}</code>	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: <i>http://.../?args=ABC%20123</i> matches a command like: <i>iscclient PROG ABC 123</i>
<code>\${user}</code>	WebClient specific logged in user name
<code>\${webclient.rootDir}</code>	Root directory used to resolve relative paths

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 <code>\${clientIp}</code> <code>\${customArgs}</code>
----------------------------	--

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:

[illegible]

- 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 language 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. Extract the index.html file stored in webclient/webclient-server-war and copy it to the folder that you've just created. You can use an archive manager software like [7-Zip](#) to perform this operation easily.
3. Edit the index.html file. 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.

Embedding the COBOL application in an HTML page

With WebClient it is possible to embed your COBOL application in your own web page.

Create a folder that will contain the assets needed for the web page. In the WebClient configuration for the application set the [Web Folder](#) parameter to this folder.

You can copy this snippet to your web page:

```

<html class="ws-fullscreen" lang="en">

<head>
  <link rel="stylesheet" href="http://<webclient-host-and-port>/<application_path>/
css/style.css"/>
</head>

...

<div class="webswing-element" data-webswing-instance="webclientInstance">
  <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>

...

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

  var webclientInstance= {
    options: {
      autoStart: true,
      syncClipboard: true,
      args: '',
      connectionUrl: 'http://<webclient-host-and-port>/<application_path>'
    }
  }

  (function (window, document) {
    var loader = function () {
      var baseUrl = 'http://<webclient-host-and-port>/<application_path>';
      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 : "undefined";
          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();
    };
    window.addEventListener ? window.addEventListener("load", loader, false) : window.
attachEvent("onload", loader);
  })(window, document);

</script>

```

In the snippet you need to replace all instances of the placeholders <webclient-host-and-port> <application_path> with the actual values for your deployment.

The *data-webclient-instance="webclientInstance"* is needed to identify the div element as a WebClient instance. A global object will be created with the name specified as the attribute value (*webclientControl* in this case), and it can be used to control the instance.

The following functions are available in this global object:

configure(options)	configures this WebClient instance. This function reads the options from <data-webswing-instance_value>.config (ie. webclientInstance.options) global variable if exists. Argument options is object with following properties: <ul style="list-style-type: none">• <i>autoStart</i> - tells WebClient to execute configure() and start() right after the instance is initialized. If it is false, start() function has to be triggered manually.• <i>syncClipboard</i> - tells WebClient to interact with the system clipboard in order to enable the Paste option in the context menu of input fields (not supported by all browsers).• <i>autoReconnect</i> - number of milliseconds to wait until connection auto-retry when disconnected• <i>disableLogout</i> - removes Logout button from all dialogs• <i>args</i> - additional Java application arguments. Appended to those defined in configuration• <i>recording</i> - record this application session (default: false)• <i>clientId</i> - set the clientId, used with mirror session or to continue running session• <i>mirror</i> - for starting a mirror session. Only admin role is allowed to use this (default: false)• <i>connectionUrl</i> - base URL for connecting to websocket service (default: current location url)• <i>recordingPlayback</i> - file for session recording playback. Only the admin role is allowed to use this
disconnect()	disconnects the current WebClient session, but leaving the swing application running
logout()	disconnects the current WebClient session and logs user out
kill()	disconnects the current WebClient session with stopping the COBOL application
setControl(boolean)	enables/disables the control of application. If set to false, no user events are sent to the COBOL application
start()	this will initiate the connection to Webclient server and start the COBOL application. If the autoStart is set to false or not defined in config object, the start function has to be called manually, otherwise the WebClient will call start function automatically
repaint()	notify the WebClient server that the application needs repaint
instanceId()	get the instanceId of the currently running instance
getConnectionInfo()	get information about current connection

You can control the size of the div element where WebClient is displayed with regular CSS . You can even change the size of the div dynamically.

If you are embedding WebClient to a page on a different domain, you will have to enable Cross-origin resource sharing (CORS) in applications configuration's [CORS Origins](#) option. Set * to allow all domains, or use list of allowed domains.

Bootstrapping WebClient JavaScript

When the WebClient JavaScript is loaded it will scan the DOM for div elements with attribute *data-webswing-instance* and will initialize them automatically. For use cases when WebClient instance has to be started later, it is possible to export global WebClient variable exposing the bootstrapping API. By defining the attribute *data-webswing-global-var="webclientControl"* on the `<script>` element of embedded snippet, you tell WebClient to create a global variable called *webclientControl*, which has following functions:

scan()	re-runs the full DOM search for data-webswing-instance attribute
bootstrap(divElement)	instantiate WebClient in div element specified in argument

Communicating between the browser and the COBOL application

Javascript API:

Once the application is embedded on the web page, you have access to the Javascript API:

The *webclientInstance* object contains a method, *performAction*, that is used to send messages to the COBOL app.

Usage:

```
performAction(options)
```

options is an object with the following properties:

- `actionName` [string], mandatory
- `data` [string], optional
- `binaryData` [binary] optional

Example:

```
webclientInstance.performAction({actionName: "myAction", data: "100", binaryData: someBinaryData})
```


To be able to receive messages coming from the COBOL app, a listener callback can be created in the webclientInstance's customization property:

```
var webclientInstance = {
  options: {
    ...
    customization: function(injector) {
      injector.services.base.handleActionEvent = function(actionName, data, binaryData)
      {
        // javascript code to handle action
      }
    }
  }
}
```

COBOL Api

To support communication on the COBOL app, a data structure needs to be defined to map the messages:

```
01 IWC-STRUCT.
   03 IWC-ACTION PIC X(n) .
   03 IWC-DATA   PIC X(n) .
   03 IWC-BYTES  PIC X(n) .
```

New routines are available to handle communication: IWC\$INIT, IWC\$GET, IWC\$SET, and IWC\$STOP.

CALL "IWC\$INIT" USING crt-status-value	This CALL is used to initiate the communication between COBOL and html and specifies the value of crt status that will be used to terminate the ACCEPT when a message sent by the javascript application is received
CALL "IWC\$GET" USING iwc-struct [timeout]	This CALL is used to retrieve the incoming message. The function will wait for a message to be available, in the message queue is empty unless a value is specified as timeout (in hundreds of a second)
CALL "IWC\$SEND" USING iwc-struct	This CALL is used to send a message to the javascript application. Sending an action will cause the browser to execute the callback defined in the handleActionEvent function.
CALL "IWC\$STOP"	This CALL stops communication with the javascript app. All messages sent from javascript after this routine is executed will be ignored.

All ACCEPT statements in execution will terminate as a result of receiving a message, and the program needs to call the **IWC\$GET** routine to check if the message needs to be handled. If not, the ACCEPT can be reissued.

Message are stored in a queue and are removed from the queue as soon as **IWC\$GET** has read them. Every run unit (the main run unit and separate run unit generated by CALL RUN statements) has its own queue and every message from html is duplicated in these queues.

IWC-PANEL

Javascript components can be embedded in the COBOL application screen section, when running in WebClient.

```
03 my-panel IWC-PANEL
   LINE 2, COL 2
   LINES 20, SIZE 6
   js-name "<component-name>"
   VALUE IWC-STRUCT
   EVENT EV-PROC
```

IWC-PANEL is a placeholder control, that is ignored unless the application is run in WebClient. When it is, it will trigger events in the Javascript application that allow developers to create web components and handle interactions with the COBOL program.

The *webclientInstance* object has a *compositingWindowsListener* listener object that can be used to capture IWC-PANEL creation events.

```
webclientInstance = {
  options: {
    compositingWindowsListener : {
      windowOpening: function(win) {},
      windowOpened: function(win) {},
    }
  }
}
```

The callbacks will be invoked just before and right after the panel is created. The win object passed as parameter of the callback identifies which panel has been created. The js-name attribute of the IWC-PANEL will be available in the win.name property in the callback.

For example:

```
compositingWindowsListener:{
  windowOpened: function(win) {
    if (win.name === 'f-map'){
      createMap(win)
    }
  },
},
```

Other useful properties available in the win object are:

id [string]	unique string identifier of this panel
ownerId [string]	unique string identifier of this panels's owner
tabId [string]	id (window.name) of the browser window where this canvas is rendered
element [Element]	DOM element (canvas) representing this panel
name [string]	panel name in your application (js-name property in the screen section)
webswingInstance [object]	refers back to the webclientInstance object that owns the panel

handleActionEvent [function]	assign a callback to handle communication with the COBOL program (discussed later) <code>handleActionEvent = function(actionName, data, binaryData)</code>
performAction(options)	is used to send events to the control's event procedure in the COBOL application. <code>options</code> is an object with properties <code>actionName [string]</code> , <code>data [string]</code> , <code>binaryData [binary]</code> . Only the <code>actionName</code> property is required.

When the COBOL program needs the web component to perform an action, it can issue the following statement:

```
modify my-panel value iwc-struct
```

This code will cause the `handleActionEvent` callback of the `win` object in the javascript code to be executed, and the `iwc-struct` parameters will be passed to the callback.

For example, the following code captures messages coming from COBOL's `MODIFY VALUE` statement:

```
win.handleActionEvent = function(actionName, data, binaryData) {
  if (actionName === 'addOffices'){
    offices = JSON.parse(data);
    ...
  }
  if (actionName === 'selectOffice'){
    let office = JSON.parse(data);
    ...
  }
}
```

The javascript program may need to perform actions on the control, and can do so by calling the `win.performAction` methods, for example:

```
win.performAction({actionName: 'pinClicked', data: marker.title});
```

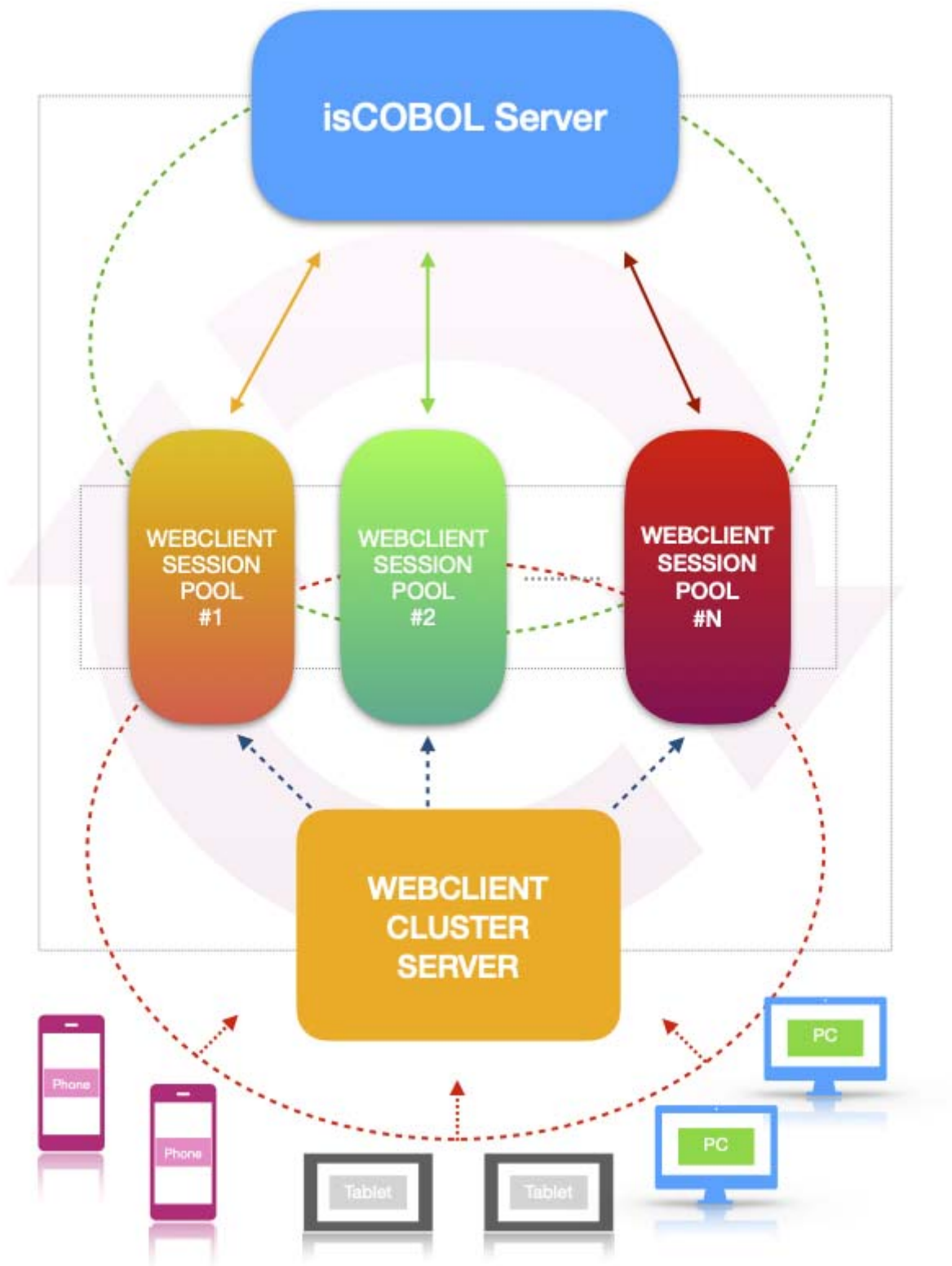
In the COBOL application the `IWC-PANEL` event procedure will be called, the message can then be retrieved and handled, for example:

```
EV-PROC.
  if event-type = ntf-iwc-event
    inquire f-map value in fmap-struct
    if fmap-action = "pinClicked"
      ...
    end-if
  end-if
.
```

Cluster Deployment

The main concept of clustered WebClient is to divide the former WebClient Server into separate modules which handle web requests (WebClient Cluster Server) and application processing (WebClient Session Pool).

In a minimal setup you need one Cluster Server and one Session Pool. You can use multiple Cluster Servers for redundancy, but it should not be necessary to have more than two Cluster Servers in most cases. Session Pools can be scaled dynamically, depending on your needs of resources or connected users. Note that it is not possible to connect more than one WebClient Admin in the cluster.



In this guide we're going to setup clustered environment with three machines: a Cluster server and two Session Pool servers.

The clustered environment will be tested using the isCOBOL Demo program (Iscontrolset).

Preliminary check (optional)

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 on every server where you're going to install a WebClient Session Pool:

```
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.

Setup of the Cluster server

In the Cluster server the following products must be installed using the appropriate setups:

- WebClient Admin
- WebClient Cluster Server

Note - these products can also be installed on separate machines, but in this guide we're going to keep them on the same machine for simplicity.

Use the following command to start the WebClient Cluster service:

- Windows

```
cd %ISCOBOL%\bin  
webccclient-cluster.exe
```

- Linux

```
cd $ISCOBOL/bin  
./webccclient-cluster
```

Note - the above snippets assume that the ISCOBOL environment variable is set to the isCOBOL WebClient installation directory.

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

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

Setup of the Session Pool servers

In a Session Pool server the following product must be installed using the appropriate setup:

- WebClient Session Pool

Edit the *webclient/cluster/session-pool/webclient-sessionpool.properties* configuration file to provide an unique ID to the current Session Pool and the web socket URL to the Cluster server. For example, assuming that the Cluster server was started on the default port 8080 at the IP address 192.168.0.100, set these entries in *webclient-sessionpool.properties* of the first Session Pool server:

```
sessionpool.id = session-pool-1
webclient.server.websocketUrl = ws://192.168.0.100:8080
```

And set these entries in *webclient-sessionpool.properties* of the second Session Pool server:

```
sessionpool.id = session-pool-2
webclient.server.websocketUrl = ws://192.168.0.100:8080
```

Use the following command to start the WebClient Session Pool service:

- Windows

```
cd %ISCOBOL%\bin
webccclient-session.exe
```

- Linux

```
cd $ISCOBOL/bin
./webccclient-session
```

Note - the above snippets assume that the ISCOBOL environment variable is set to the isCOBOL WebClient installation directory.

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

```
Successfully connected to server [ws://192.168.0.100:8080]!
```

Configure the application on the Cluster server

In the Cluster Server use the following command to start the WebClient Admin:

- Windows

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

- Linux

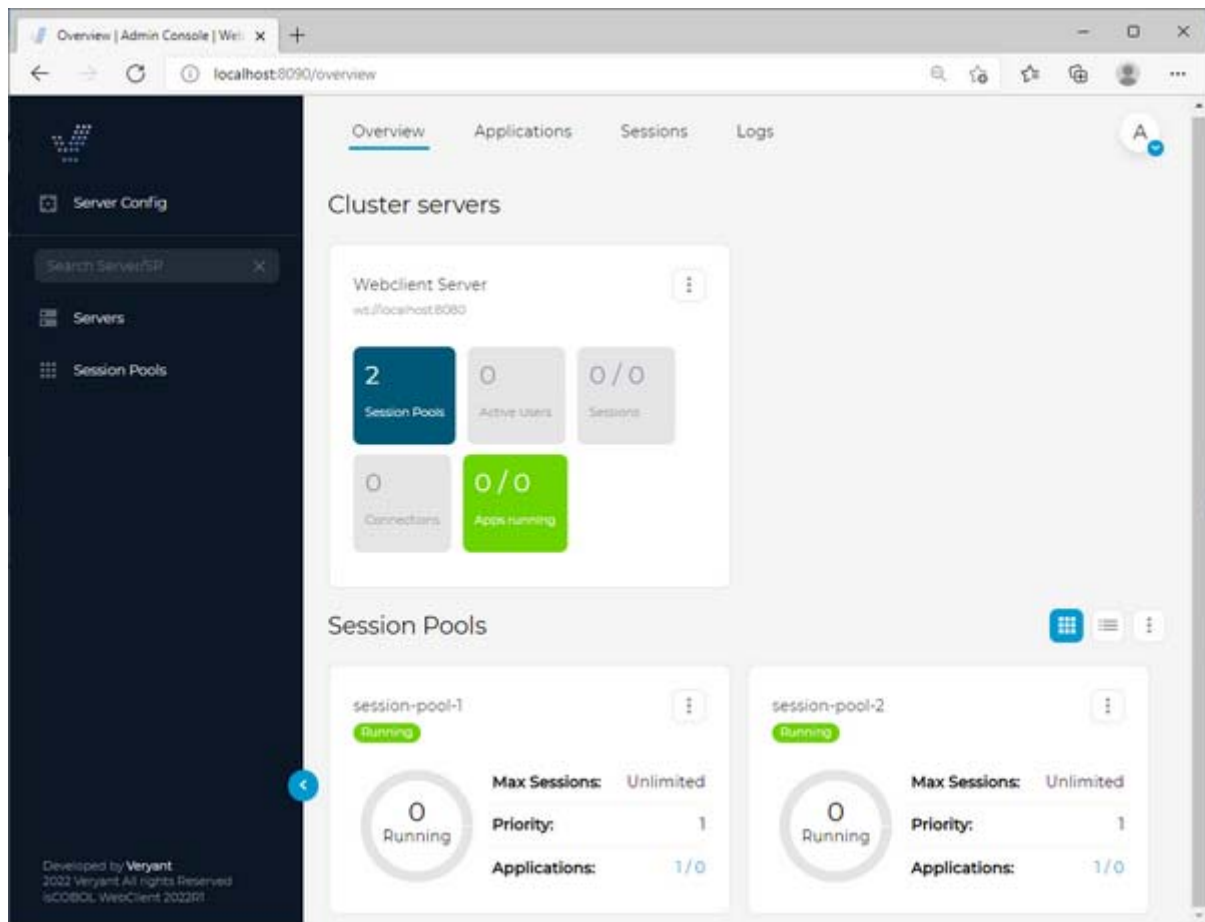
```
cd $ISCOBOL/bin
./webccclient-admin
```

Note - the above snippets assume that the ISCOBOL environment variable is set to the isCOBOL WebClient installation directory.

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

```
oejs.Server:main: Started @3792ms
```

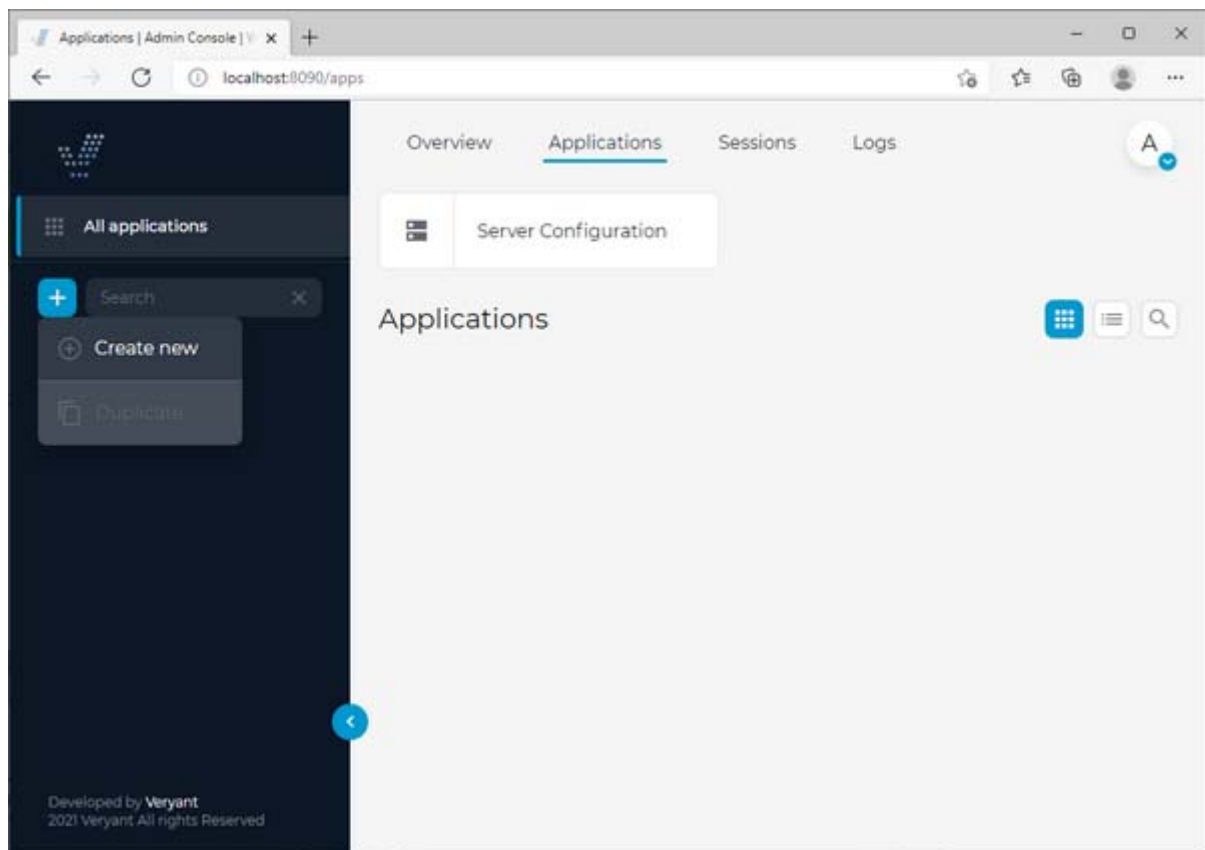

Browse to <http://localhost:8090> to reach the WebClient Admin web application. In the home page you can see an overview of active session pools.



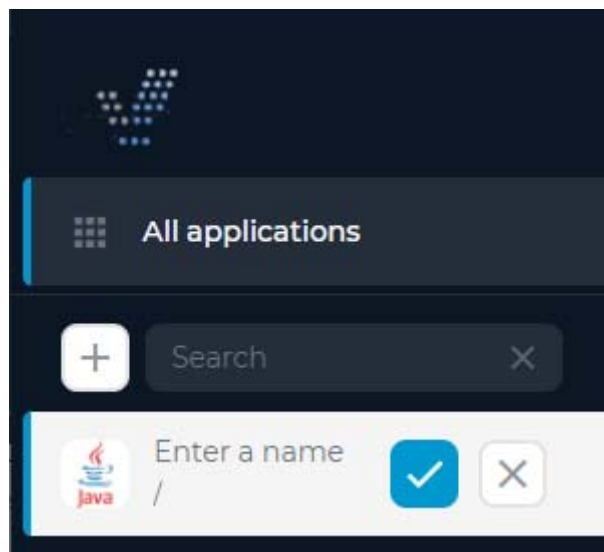
Switch to the *Applications* tab.

The menu on the left shows the list of available applications. This list is currently empty.

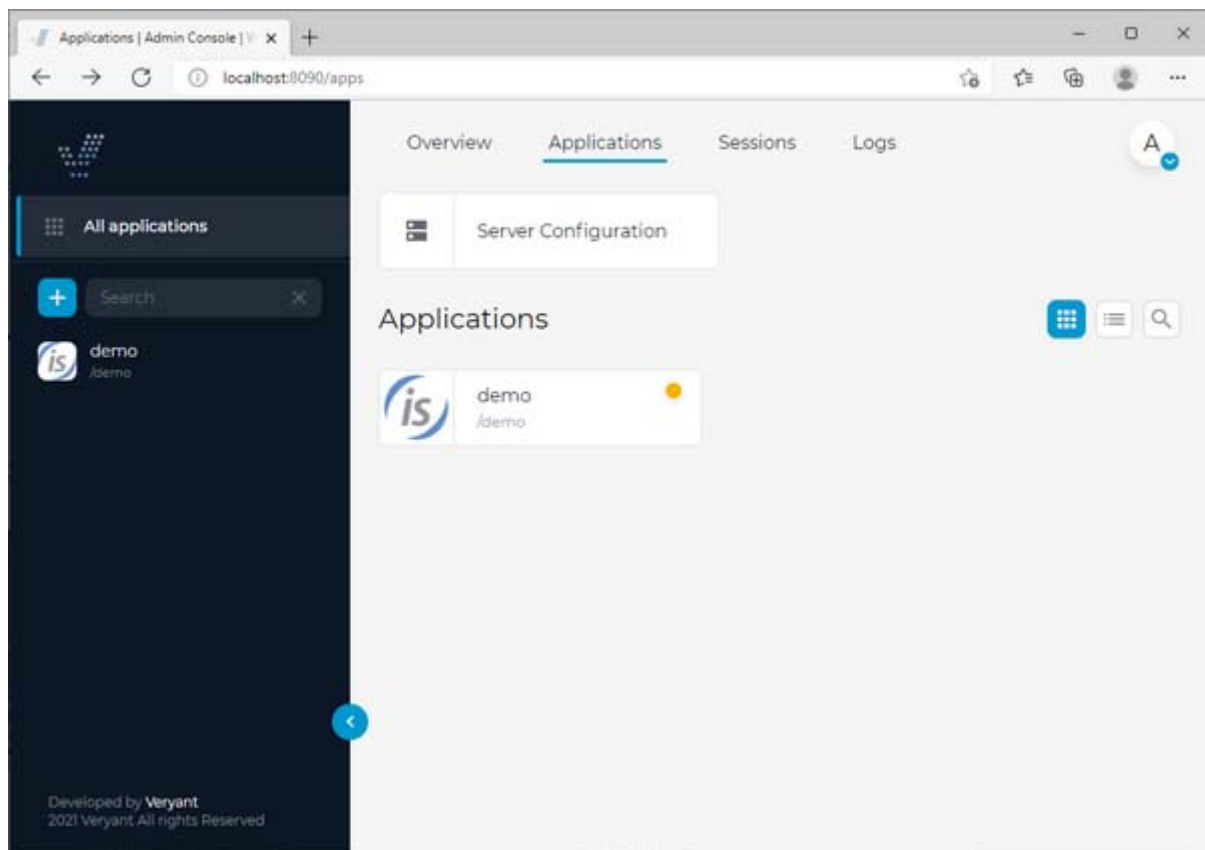
Click the plus button before the search field to show the “Create new” option.



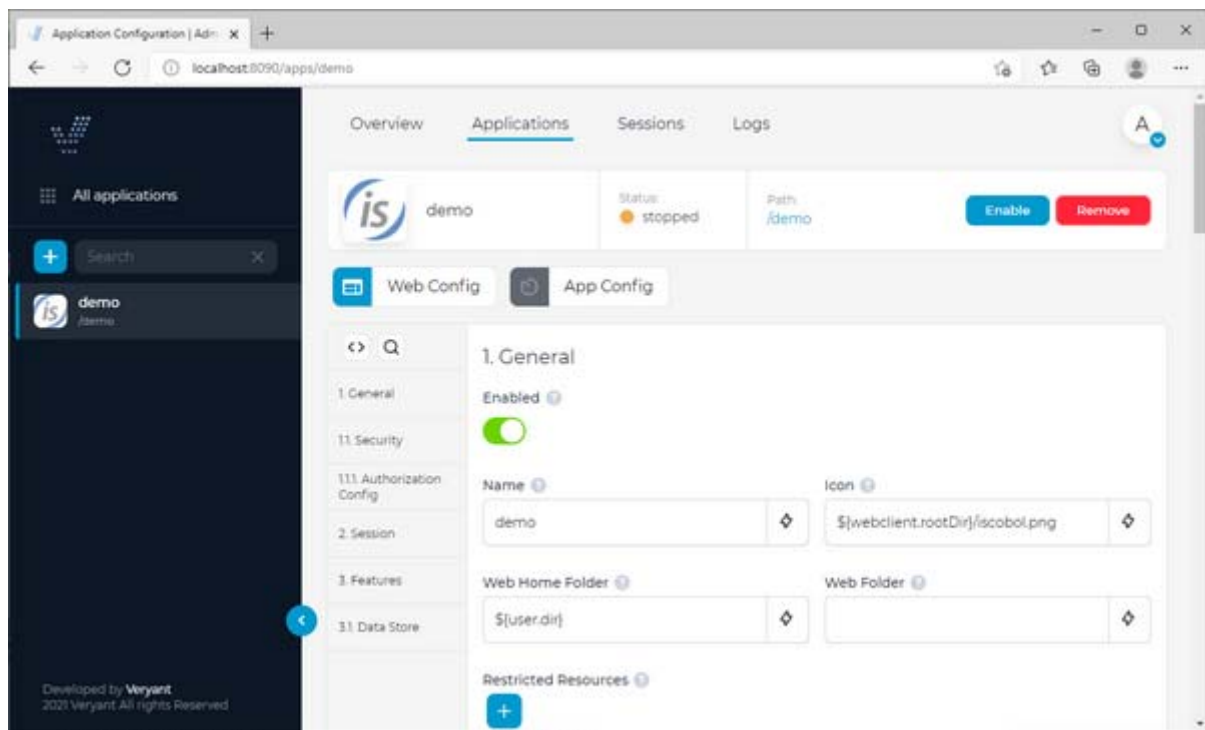
By clicking "Create new" you will be prompted for the application name:



Type "demo" in the field. This is the name that will be used in the URL in order to use the COBOL application.
Click the confirmation button to make the new application appear in the list:

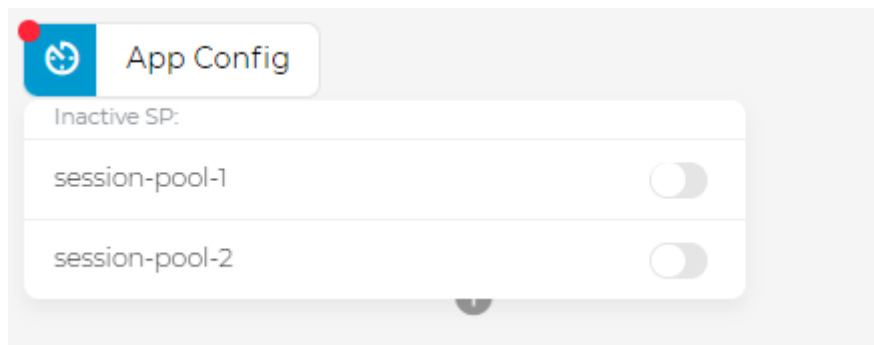


Click on the *demo* button to reach the application configuration.



Click the *Enable* button in order to activate the application, then switch to the *App Config* section.

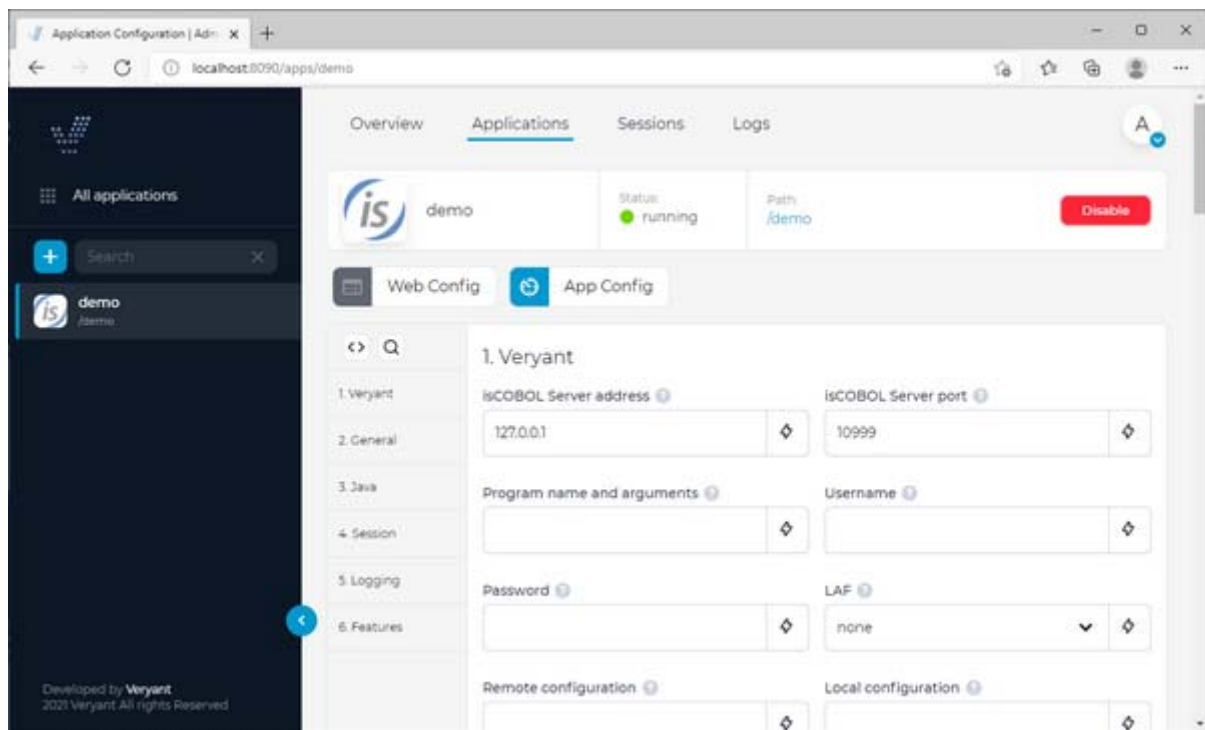
Before configuring the applicaiton, you have to associate it to the desired Session Pool servers. Click the clock icon in the *App Config* button in order to get the list of available Session Pool servers.



In order to associate the desired Session Pool servers, activate the corresponding check box.

Note - every time you change the Session Pool association, the app configuration is reset and you have to configure it again from scratch.

Now you can proceed in configuring the application.



Type the IP address or the name of the server where isCOBOL Server is started and listening in the *isCOBOL Server address* field.

If isCOBOL Server is not listening on the default port 10999, type the correct port number in the *isCOBOL Server port* field.

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

Scroll down to find the *Class Path* setting and

- replace "\${webclient.rootDir}/../lib/*.jar" with "\${webclient.rootDir}/../../lib/*.jar"
- replace "\${webclient.rootDir}/../jars/*.jar" with "\${webclient.rootDir}/../../jars/*.jar"

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`.

Multiple sessions will be automatically redistributed on the two Session Pool servers that we configured.

JETTY and JMS Configuration

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

WebClient uses the Java Message Service (JMS) to communicate with the underlying COBOL applications.

Jetty Configuration

By default the WebClient and WebClient Cluster start on port 8080 while the WebClient Admin Console starts on port 8090. WebClient Test Tool starts on port 8888.

You can change these settings by editing the files *webclient/jetty.properties*, *webclient/cluster/cluster-server/jetty.properties*, *webclient/admin/jetty.properties* and *webclient/test-tool/jetty.properties* under the 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. Replacing 0.0.0.0 with any IP address will allow connections only from that IP address.
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

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
webccclient.exe -J-Dwebclient.jmsUrl=nio://127.0.0.1:12345
```

- Linux

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

How to enable HTTPS

By default WebClient tools communicate over the standard HTTP protocol.

In order to switch to the HTTPS protocol, proceed as follows:

1. Be sure you have a valid keystore file, either self-signed or released by a CA authority
2. Edit the `jetty.properties` files to disable HTTP and enable HTTPS, providing the path to your keystore and the keystore password, i.e.

```
...
jetty.webclient.server.http=false
...
jetty.webclient.server.https=true
jetty.webclient.server.https.truststore=C:/TestSSL/test.jks
jetty.webclient.server.https.truststore.password=secret
jetty.webclient.server.https.keystore=C:/TestSSL/test.jks
jetty.webclient.server.https.keystore.password=secret
```

The following files are affected:

- o `webclient/jetty.properties`
 - o `webclient/admin/jetty.properties`
 - o `webclient/cluster/cluster-server/jetty.properties`
3. Update the `adminConsoleUrl` value in `webclient/webclient.config` and `webclient/cluster/cluster-server/webclient-server.config` to reflect the HTTPS port set in `webclient/admin/jetty.properties`, i.e.

```
"adminConsoleUrl" : "https://localhost:9443/"
```

4. Edit the `webclient.properties` files to configure HTTPS by providing the path to your keystore and the keystore password, i.e.

```
webclient.server.websocket.truststore = C:/TestSSL/test.jks
webclient.server.websocket.truststore.password = secret
webclient.server.websocket.hostnameVerifier.disabled = true
```

The following files are affected:

- o webclient/webclient.properties
 - o webclient/admin/webclient-admin.properties
 - o webclient/cluster/cluster-server/webclient.properties
 - o webclient/cluster/session-pool/webclient-sessionpool.properties
5. Edit webclient/admin/webclient-admin.properties to provide SSL-enabled URLs according to the settings configured for Jetty, i.e.

```
...  
webclient.server.publicUrl = https://localhost:8443  
...  
webclient.server.websocketUrl = wss://localhost:8443  
...
```

6. Edit webclient/cluster/session-pool/webclient-sessionpool.properties to provide SSL-enabled URLs according to the settings configured for Jetty, i.e.

```
...  
webclient.server.websocketUrl = wss://localhost:8443  
...
```

7. Start (or restart) the services.

Note - the SSL security can be configured at a higher level (e.g. on the HTTP Server) when using [Reverse Proxy](#).

Reverse Proxy

Providing access to WebClient through a HTTP reverse proxy is recommended.

In this chapter we provide an example for [Apache HTTPD](#).

This example expects that WebClient server is running at localhost port 8080. Replace 127.0.0.1 with actual IP or domain name if the HTTPD is running on a different server.

httpd.conf:

```
<VirtualHost *:80>
    ServerName demo.veryant.com
    Redirect permanent / https://demo.veryant.com/
</VirtualHost>

<VirtualHost *:443>
    ServerName demo.veryant.com
    SSLEngine on
    SSLCertificateFile /etc/letsencrypt/live/demo.veryant.com/fullchain.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/demo.veryant.com/privkey.pem

    RewriteEngine on
    RewriteCond %{HTTP:Upgrade} =websocket
    RewriteRule /(.*) ws://127.0.0.1:8080/$1 [P,L]
    RewriteCond %{HTTP:Upgrade} !=websocket
    RewriteRule /(.*) http://127.0.0.1:8080/$1 [P,L]

    ProxyPass "/" "http://127.0.0.1:8080/"
    ProxyPassReverse "/" "http://127.0.0.1:8080/"

</VirtualHost>
```

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 JEE servers can work as well, as long as they support the Servlet 3.0 spec.

WebClient Server & Admin

To deploy WebClient Server and WebClient Admin to Tomcat follow these steps:

1. Copy *webclient/webclient-server.war* and *webclient/admin/webclient-admin-server.war* to Tomcat's *webapps* folder.
2. Copy *webclient/admin/webclient-admin.properties* to *webclient*. You need this file in WebClient's root folder, along with *webclient.properties*.
3. Edit *webclient/webclient.properties* to enable and set the *webclient.server.websocketUrl* property, e.g.

```
# websocket URL of this server, if deployed outside embedded Jetty
webclient.server.websocketUrl = ws://localhost:8080
```

4. Edit *webclient/webclient-admin.properties* to add the webapp name E.g.

```
# public URL on which applications are accessible on public internet
webclient.server.publicUrl = http://localhost:8080/webclient-server
...
# comma-separated list of websocket URLs to webclient cluster servers
webclient.server.websocketUrl = ws://localhost:8080/webclient-server
```

5. Edit *webclient/webclient.config* to set the *adminConsoleUrl* property as follows:

```
"adminConsoleUrl" : "http://localhost:8080/webclient-admin-server"
```

6. In Tomcat's *catalina.properties* add the following properties:

```
webclient.warLocation=webapps/webclient-server.war
webclient.configFile={webclient_home}/webclient.config
webclient.tempDirBase={webclient_home}/tmp
webclient.rootDir={webclient_home}
```

Notes:

- Replace *{webclient_home}* with the directory where WebClient was installed. For example, on Windows, use "C:\Veryant\isCOBOL_WEBC2023R1\webclient".
 - Tomcat should be executed from its root folder, otherwise the path to *webclient.war* needs to be adjusted.
 - The above snippets use the Tomcat's default port 8080. Change this port number if you started Tomcat on a different port.
7. Browse to "http://localhost:8080/webclient-server" to reach the WebClient home page.

WebClient Cluster Server & Admin

The WebClient's cluster environment is implemented using three pieces of software:

- WebClient Cluster Server
- WebClient Admin
- WebClient Session Pool

WebClient Session Pool is a Java application and can't be deployed in Tomcat.

WebClient Cluster Server and WebClient Admin, instead, are web applications and can be deployed in Tomcat.

To deploy WebClient Cluster Server and WebClient Admin to Tomcat follow these steps:

1. Copy *webclient/cluster/cluster-server/webclient.war* and *webclient/admin/webclient-admin-server.war* to Tomcat's *webapps* folder.
2. Copy *webclient/admin/webclient-admin.properties* to *webclient/cluster/cluster-server*. You need this file in WebClient Cluster Server's root folder, along with *webclient.properties*.
3. Edit *webclient/cluster/cluster-server/webclient.properties* to enable and set the *webclient.server.websocketUrl* property, e.g.

```
# websocket URL of this server, if deployed outside embedded Jetty
webclient.server.websocketUrl = ws://localhost:8080
```

4. Edit *webclient/cluster/cluster-server/webclient-admin.properties* to add the webapp name, e.g.

```
# public URL on which applications are accessible on public internet
webclient.server.publicUrl = http://localhost:8080/webclient
...
# comma-separated list of websocket URLs to webclient cluster servers
webclient.server.websocketUrl = ws://localhost:8080/webclient
```

5. Edit *webclient/cluster/cluster-server/webclient-server.config* to set the *adminConsoleUrl* property as follows:

```
"adminConsoleUrl" : "http://localhost:8080/webclient-admin-server"
```

6. In Tomcat's *catalina.properties* add the following properties:

```
webclient.warLocation=webapps/webclient-server.war
webclient.configFile={webclient_home}/cluster/cluster-server/webclient-server.config
webclient.tempDirBase={webclient_home}/cluster/cluster-server/tmp
webclient.rootDir={webclient_home}/cluster/cluster-server
```

Notes:

- Replace *{webclient_home}* with the directory where WebClient was installed. For example, on Windows, use "C:\Verant\isCOBOL_WEBC2023R1\webclient".
 - Tomcat should be executed from its root folder, otherwise the path to *webclient.war* needs to be adjusted.
 - The above snippets use the Tomcat's default port 8080. Change this port number if you started Tomcat on a different port.
7. Browse to "http://localhost:8080/webclient" to reach the WebClient home page.

Deploying in Tomcat on headless Linux

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

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

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.

Clipboard

Copying and pasting text among the fields of the COBOL application inside the browser is always supported.

Sharing text with external software instead is supported only when the [Allow Local Clipboard](#) option is enabled in the app configuration. For security reasons, the user is prompted for confirmation before accessing the system clipboard for storing or retrieving text.

The Paste option in context menus is not supported by all browsers.

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\$OPENSABVEBOX 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.browser.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.

We suggest you open your windows maximized, and use the [LM-RESPONSIVE](#) layout manager to resize the controls.

When the virtual keyboard appears over a maximized window, the [NTF-RESIZED](#) event is fired.

Test Tool

WebClient Test Tool allows you to create test cases for your application running in WebClient. Test tool is a web application that lets you record a test case, playback the test case and also automate multiple test cases using Selenium Grid.

Setup

You need a running instance of WebClient with the application you want to test to be able to run WebClient Test Tool.

Make sure your WebClient application allows connections from the test tool server setting [CORS Origins](#) application configuration.

Make sure you have following configuration of your WebClient application:

- [Session Mode](#) = ALWAYS_NEW_SESSION,
- [Max. Connections](#) = 100,
- [Auto Logout](#) = OFF,
- [Enable Test Mode](#) = ON.

It is recommended to create and test in a maximized window with the same resolution.

It is suggested to set [Security Module Name](#) to NONE to speed up the recording and playback process by skipping the user authentication.

Configuration and startup

Start the Test Tool application by running the command

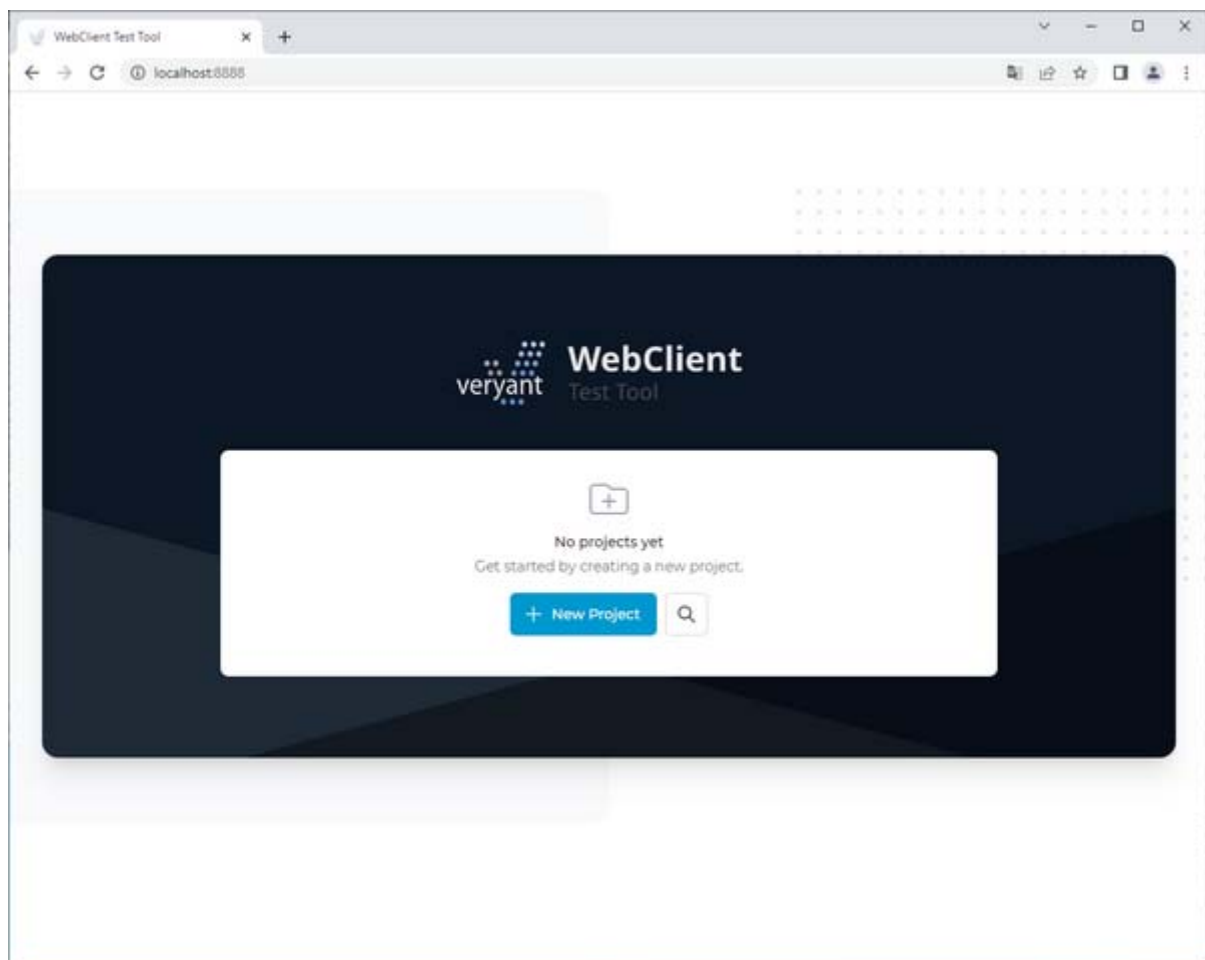
```
webcclient-testtool
```

You can change the default ports in the [Jetty Configuration](#).

There is also *webclient-testtool.properties* configuration file where you can configure following:

config.hubUrl	Selenium Grid URL to be used in automated testing, must be accessible by Test Tool server
config.testToolUrl	URL of Test Tool server that will be accessible from browser opened by Selenium node
testtool.projects.folder	projects folder
test.implicitWaitSec	implicit wait for assertion in seconds
test.appInitWaitSec	wait in automated test run for WebClient application to be initialized
test.appStartWaitSec	wait in automated test run for WebClient application to be started and ready to be tested

Open the Test Tool on <http://localhost:8888>.

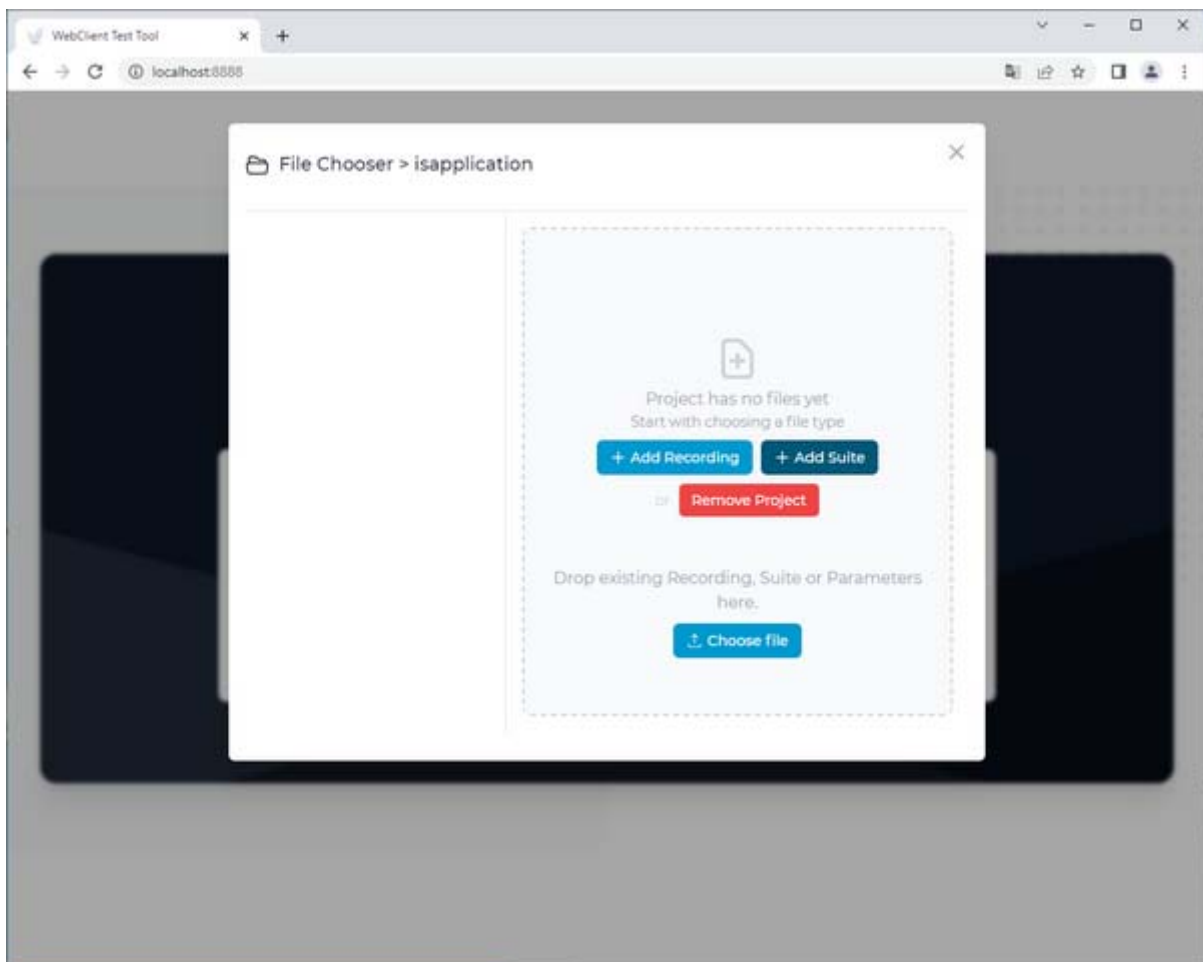


Manual Testing

First you need to create a new project. This way you can group your recordings, test suites and parameter files. Selecting your project opens File Chooser dialog. Here you can manage your project files:

- create new,
- upload existing,
- rename,
- delete,
- download and open the file.

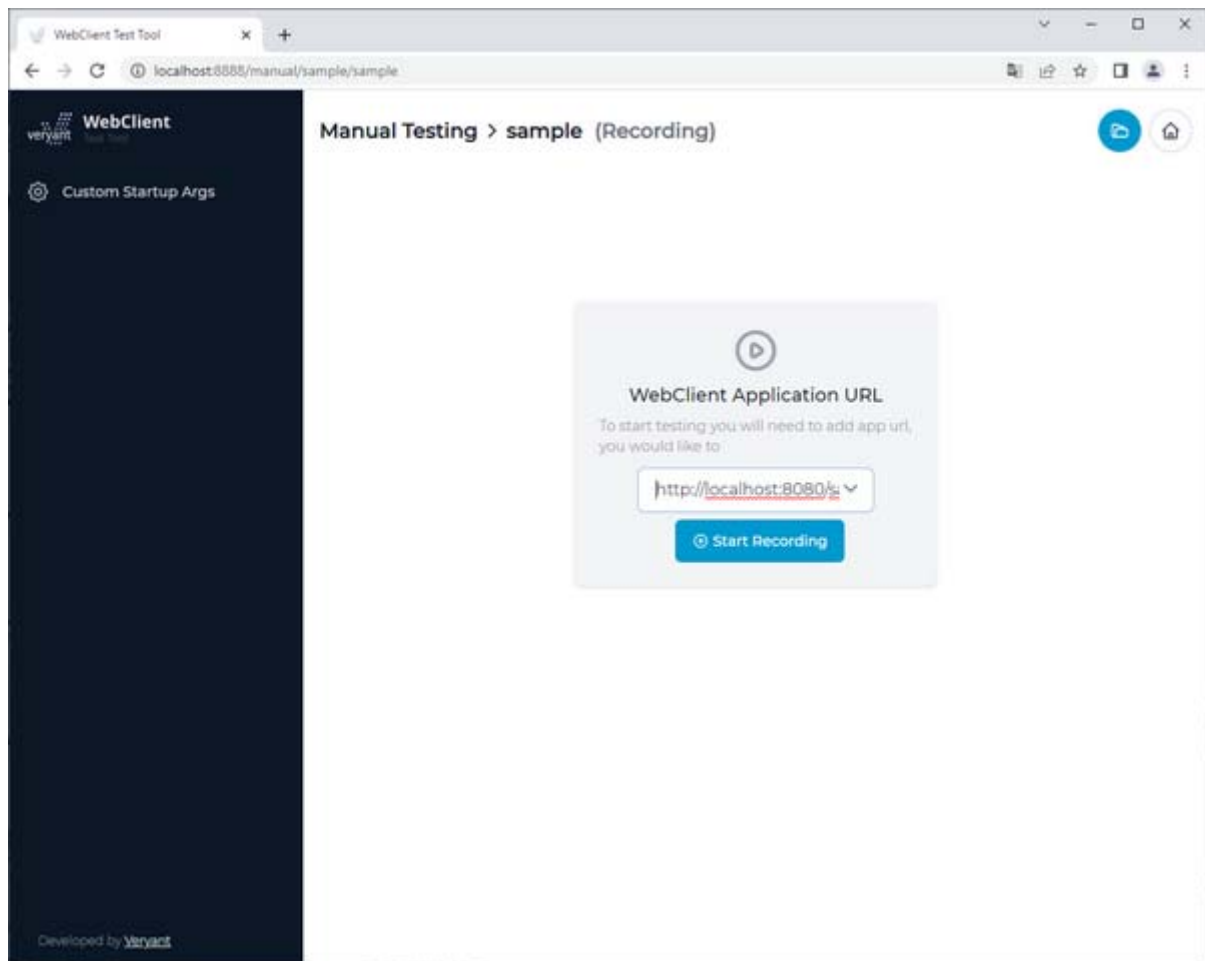
Recording



To create a new recording, click the *Add Recording* button, provide a name for your recording and click the *Open* button.

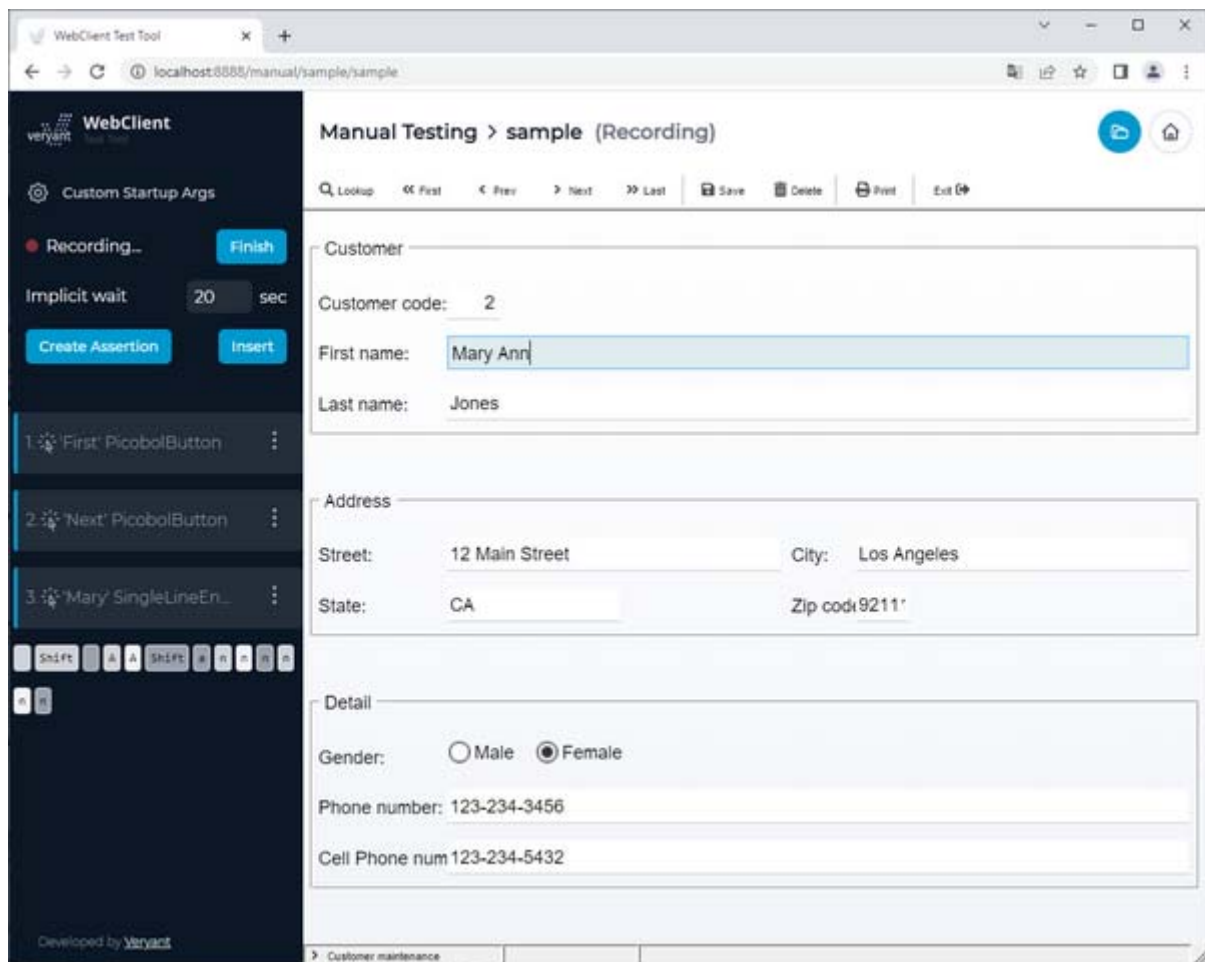
Note - the instructions and screenshots below use the isCOBOL Modernization sample program for the demonstration. They assume that you have a WebClient application that runs the CUSTOMER program and you configured it as explained in the [Setup](#) paragraph.

When you open a new recording file it will open in manual testing mode. To record a test case enter URL of your WebClient application (i.e. "http://localhost:8080/sample") and click *Start recording*.



Once you click the *Start recording* button the tool starts recording your actions with the WebClient application. Every time you click in the application a new assertion is created. An assertion is a set of properties that will be validated in the test case, like component type, value, path, etc. Path is especially important to be able to find and exactly identify the component when running the test later. These properties are coming from a component tree that is requested directly from the application each time. Depending on the application size and structure, connection latency, and performance of the environment setup this may take from a few milliseconds to a few seconds, so please be patient after each click you make when creating a test case.

For example, after clicking the *First* button, then the *Next* button and finally typing "Ann" after "Mary" in the First Name field, the screen will look like this:



On the left you find the list of recorded actions.

After you have done some actions and you want to create a custom assertion, use the *Create Assertion* button. Now hover over the application with your mouse to pick a component in the application view. Click the component which you want to make an assertion for. Now check component properties in the toolbox which you want to be validated by the assertion. Click *Add* to save the assertion. The recording of test case now continues.

When typing text during test creation, it is recommended to first click on the input or text area component to make sure there is an assertion created for the input component, even if the input component is already focused when it appears in UI. This way you make sure the test tool runner first waits for the input component to appear in the UI and then starts typing the text.

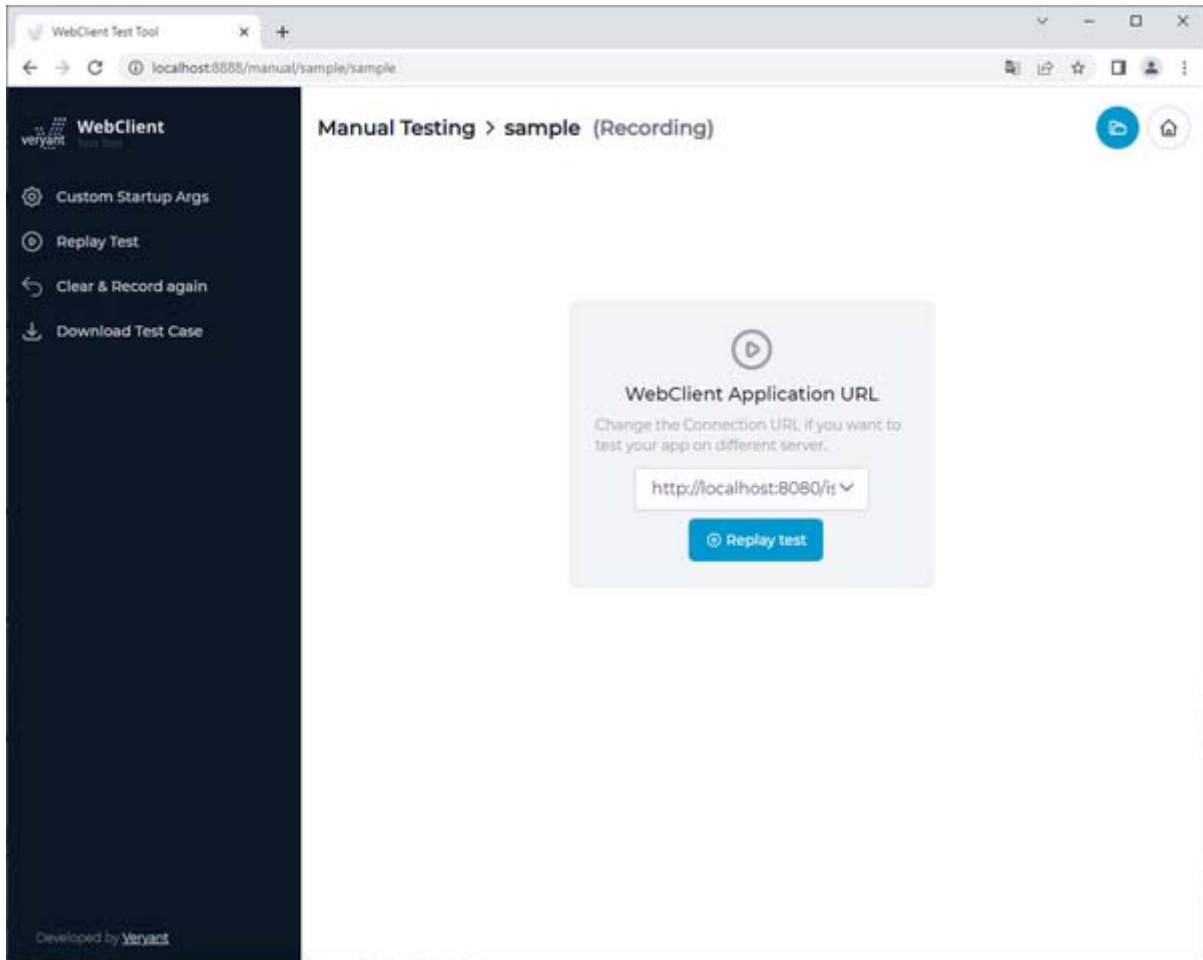
There is a 20 seconds implicit wait period by default. This is a time period that applies to all future created assertions and represents maximum wait time for the assertion to become valid. During this period, (while running a test case) Test Tool will try to validate the assertion multiple times until it passes. If the assertion fails to validate, the whole test case fails. You can adjust the implicit wait time at the beginning or during the test creation, or you can adjust explicit wait time for each assertion. You can also set a default implicit wait period in *webclient-testtool.properties* file *test.implicitWaitSec = 20*.

In addition to assertions you can also insert text, delay or a parameter using the *Insert* dialog. You can insert text and parameter only at the current position in test case. The inserted value will be immediately applied. Delays can be inserted at any position at any time.

When you are done with creating the test case click the *Finish* button. The Test Tool will save your current recording to the file. You should be able to replay the tests you just recorded, download the test case or clear and record test case again.

Playback

Recording files are saved to the projects folder and you can open them from the file chooser.



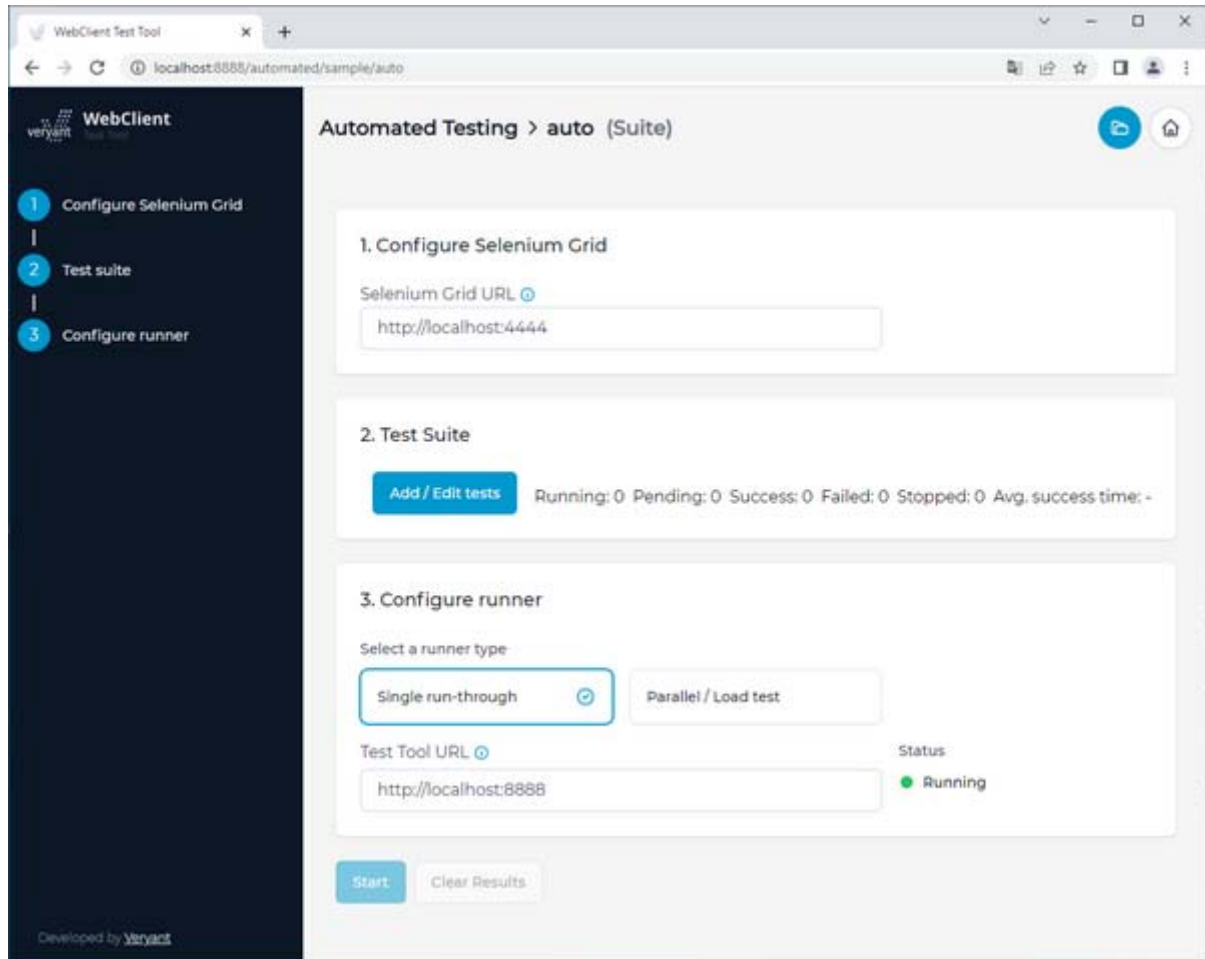
To run a test click *Replay Test*. You will now see a playback of your recorded test case. Assertions are validated in the toolbox. There are also options to pause the playback, make a single step (disabled until there is a breakpoint) or stop playback. When you are replaying your recording and something is missing or you are not satisfied with it, you can either set a breakpoint before playback, or pause on next step. When the replay is paused you are able to

- edit assertions,
- create an assertion,
- insert (text, param, delay),
- go to next step or continue playback.

When the test finishes you can see and download the test results from the toolbox.

Automating test cases

To automate the recorded test cases you have to create a Test Suite file. To create a new suite, click the *Add Suite* button, provide a name for your suite and click the *Open* button.



Automated test cases require Selenium Grid.

You can setup a Selenium Grid either in your local environment or use a 3rd party service. Please note that current Test Tool supports Selenium version 4 only.

For a 3rd party service just search for "Selenium Grid in cloud" or use one of these - <https://www.gridlastic.com>, <https://saucelabs.com>, <https://testingbot.com>, <https://seleniumbox.com>.





To setup Selenium Grid yourself please refer to the [Selenium documentation](#).

First configure your Selenium Grid. Enter the URL of the running Selenium Hub, e.g. <http://localhost:4444>. The connection will be validated, wait until you see "Status: Running". Test Tool also tries to retrieve information about running nodes.

1. Configure Selenium Grid

Selenium Grid URL [?](#)

Status
● Running

Nodes
 Running
  

To automate WebClient test cases you have created, you need to define test suites. A test suite is a simple configuration file where you define which test cases to run and other options.

You can add or edit the test cases using the *Add / Edit tests* button in the second step. Pick a recording from a list of recording files in the current project. Make sure to choose Platform and Browser based on the availability of the environment on your Selenium nodes.

Add or Edit Test Suite

Test Case #1 ✖

Recording

WebClient app URL [?](#)

Name

Enabled ☒

WebClient login username

WebClient login password

Platform

Browser










Headless ☐

[Advanced settings](#)

Add Test Case

Cancel Save

After you save the edited test suite it will be stored in the file.

	Test Case	File	App URL	Platform	Browser	Headless	Status	Time	Result
<input checked="" type="checkbox"/>	sample	sample				<input type="checkbox"/>	Ready		-
<input checked="" type="checkbox"/>	sample	sample				<input type="checkbox"/>	Ready		-
<input checked="" type="checkbox"/>	sample	sample				<input type="checkbox"/>	Ready		-


Finally, configure the test case runner. Enter the URL of test tool server instance (this can be the same instance, i.e. <http://localhost:8888> or some other instance). This URL must be accessible from the Selenium node.


You can choose between a single run-through or a parallel run. Single run-through will simply execute test cases one-by-one, one instance at a time.


With parallel you can set how many instances you want to run at the same time (Max running parallel). Use Ramp-up period to define how long should it take to start the Max running parallel instances. For example if you want 10 instances in parallel and start them in 100 seconds, there will be 1 instance started every 10 seconds. After 100 seconds the next instances start ad-hoc after a previous test finishes. To control how many tests will run in total, set Run count. This value is per test, so if you have 3 tests in your test suite and run count set to 10, the runner will run 30 instances total, 10 per each test.


3. Configure runner


Select a runner type

Test Tool URL 













Status  Running

Max running parallel 

Ramp-up period (sec) 

Run count 

Click *Start* to start the runner. You can see the progress in the test suite table. If the test case is successful the row turns green, otherwise red. For a failed test case you can take a look at a screenshot from the browser from the time the assertion failed or exception occurred. You can also see an exception stack trace from the test case and optionally also from browser console logs (Chrome, Edge).

	Test Case	File	App URL	Platform	Browser	Headless	Status	Time	Result
<input checked="" type="checkbox"/>	sample	sample				<input type="checkbox"/>	Success	32s	 Success
<input checked="" type="checkbox"/>	sample	sample				<input type="checkbox"/>	Running	2ms	 Running
<input checked="" type="checkbox"/>	sample	sample				<input type="checkbox"/>	Failed	20s	 Failed

Automated test REST API

There is a simple REST API in case you want to execute a test suite with one or more tests from an external application, e.g. for a WebClient server health check.

Send a POST request to <http://localhost:8888/rest/runTest>, set Content-Type header to "application/json" with following body:

```
{
  "parallel": false,
  "maxRunningParallel": 1,
  "rampUpPeriod": 1,
  "runCount": 1,
  "testCaseParameters": [{
    "project": "auto",
    "file": "sample",
    "name": "sample",
    "webswingAppUrl": "http://localhost:8080/sample",
    "webswingUsername": "admin",
    "webswingPassword": "admin",
    "platform": "WINDOWS",
    "browser": "chrome",
    "headless": false,
    "enabled": true
  }]
}
```

Adjust the parameters to your needs, you can also let the tests run in parallel, similar to the automated testing web UI.

Keep in mind that you get a response after all the tests you specified finish, so make sure to watch the timeout. If all tests finish successfully you get a 200 OK response, otherwise you get 500 Internal Server Error.

Known Limitations

WebClient Test Tool has the following known limitations:

- it's necessary to click on every new window to activate it before issuing any keyboard input (including Space bar or Enter to activate default buttons)
- the following controls are currently not fully supported:
 - o Tab-Control with the Allow-container style
 - o Scroll-bar
 - o List-box paged
 - o Tree-view
- encapsulated apps are not supported
- complex user interface may not work.