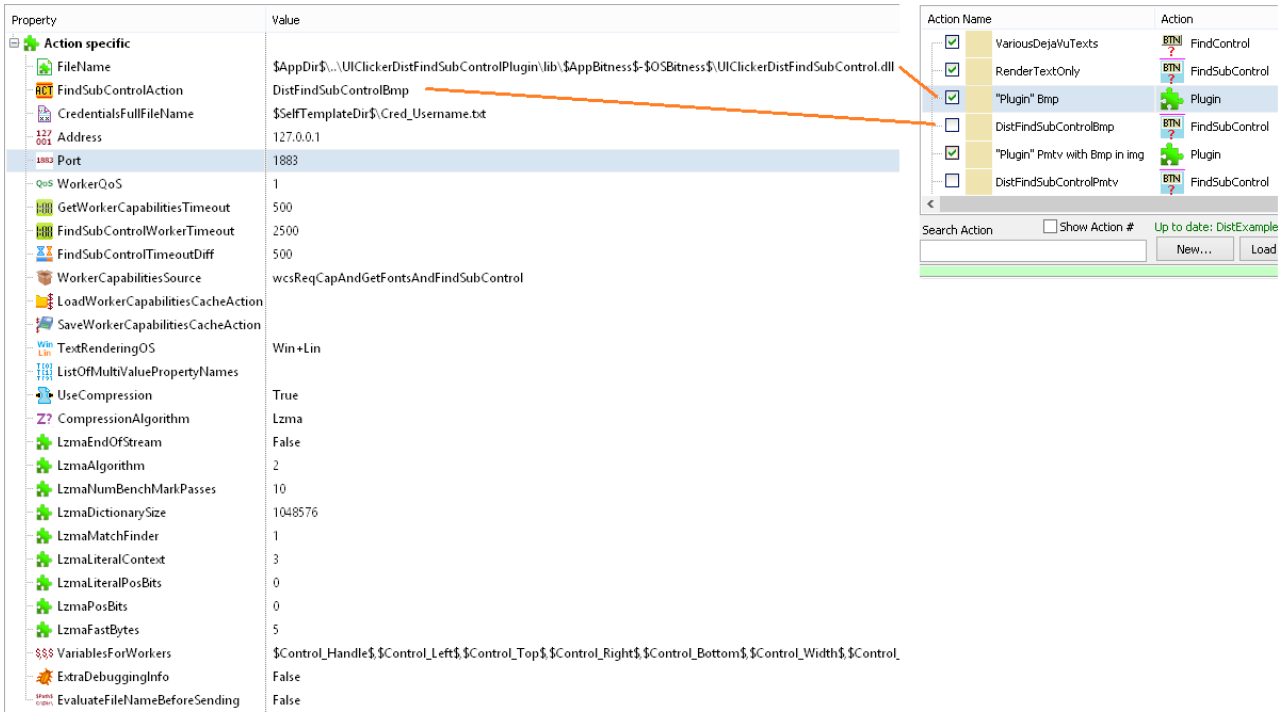
UIClickerDistFindSubControl

UIClickerDistFindSubControl is a plugin for UIClicker, used at executing the FindSubControl action, over a distributed set of UIClicker instances. It is intended to overcome the bottleneck of attempting to search using various settings, one after the other. Although not significant at a low number of font profiles and/or bmp/pmtv files to be searched, this bottleneck becomes visible when many more of these are to be "executed". The UIClickerDistFindSubControl plugin is able to split a FindSubControl action into multiple "smaller" actions and send them to remote UIClicker instances, to execute them in parallel. It has to be mentioned from the beginning, that although the plugin solves this bottleneck, it introduces a new one, which is the transmission latency. Because of these, the plugin becomes useful, only when its overall execution duration becomes smaller than the time it takes for the FindSubControl action it targets, to find.

As a simple operating principle, the plugin is configured to point to an existing FindSubControl action, it splits the action in memory into multiple actions, each with equally/balanced (as much as possible) number of font profiles, bmp and pmtv files to be searched on, archives these "smaller" actions, together with their relevant bmp/pmtv files and sends them to other UIClicker instances (called "worker UIClickers"). It takes into account the availability of installed fonts on every machine with "worker UIClickers" and the operating system they are running on (Win vs. Lin). The transmission is done over MQTT, between the plugin and worker applications, and HTTP, between workers and the actual UIClicker instances, which will execute the FindSubControl actions.



As with other plugins, this one can be compiled for 32-bit or 64-bit, to match the UIClicker which loads it (see the detailed tooltip of the "FileName" property in ObjectInspector). The targeted FindSubControl action ("DistFindSubControl" in the above screenshot) has to be disabled in the list of actions, to avoid being directly executed by UIClicker. It is used as a "content action" only, from which the plugin gets the settings to create the distributed actions. It is identified by name in the list of actions.

The plugin does not talk directly to the UIClicker instances, which will execute the FindSubControl action, but instead, it will connect to an MQTT broker. By the number of practical worker/UIClicker instances and the machines they are to be run on, the broker may also run on the same machine with these workers. This reduces the network latency a bit. Of course, this just a

proposed model. Users may choose a different one, where the broker runs on a separate machine.

There are plugin properties, for getting broker credentials ("CredentialsFullFileName") and address/port settings ("Address" and "Port"). The credentials file is a simple ini file, of the following format:

```
[Credentials]
Username=VCC
Password=Pass-je4-5o9g5yju4it23r
```

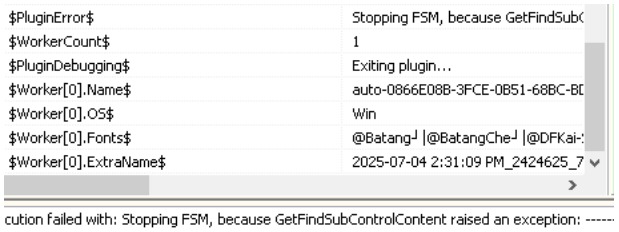
Depending on filename, the credentials file can be looked up on disk or in the UIClicker's In-Mem file system for plugins (see UIClicker documentation about its various In-Mem FSes). If the path starts with a "Mem:\" prefix, UIClicker expects the file to be present in the mentioned In-Mem FS. The file may be created there by another plugin (specifically made for this purpose).
The UIClickerDistFindSubControlPlugin repository contains various tools, which help with dynamic generation of MQTT credentials and how to make them saved in a file in the mentioned In-Mem FS.

The plugin can be configured to use the QoS 1 or QoS 2 modes of operation of the MQTT protocol. The QoS 2 adds an extra overhead and increased latency over QoS 1, and may be required if the connection is unreliable and misses responses. As this plugin is intended to run on a local network, to have a small latency, QoS 1 should be enough.

In order to get information about the number of available workers and what they can be used for in terms of installed fonts and operating system (capabilities), the plugin provides the "WorkerCapabilitiesSource" property, which currently implements five modes of operation:
- wcsReqCapAndFindSubControl – gets worker OS, and runs FindSubControl
- wcsReqCapAndGetFonts – gets worker OS and fonts, and updates some UIClicker variables
- wcsReqCapAndGetFontsAndFindSubControl – gets OS and fonts, and runs FindSubControl (default)
- wcsReqCapAndUpdateCache – gets worker OS, and executes the configured caching action
- wcsLoadCacheAndFindSubControl – loads worker info from cache and runs FindSubControl

The default value is to get the worker capabilities and execute the FindSubControl action in one plugin run. The others tend to optimize the execution by caching some information, for later use. If all the connected workers implement the same list of fonts and are running on the same OS, then the plugin can get this information in one run and use it later in multiple runs for decreased latency. This information can even be cached and saved to disk, to be loaded later (see the last to modes).

While executing, the plugin saves the $WorkerCount$ and $Worker[#].<field>$ variables. $WorkerCount$ represents the number of workers which responded as available, and for each of them, there is a set of four variables about worker name, OS, list of fonts and extra name. The $Worker[#].Name$ variable is the name allocated by broker on connection. The $Worker[#].ExtraName$ variable is user-defined and configured when starting a worker, with the "--SetWorkerExtraName" command line switch.



The list of fonts installed on the machine where the workers are running, is also obtained from the workers and saved in the $Worker[#].Fonts$ variable. The $Worker[#].OS$ variable can be Win or Lin, depending on what OS the worker was built for.

The first request this plugin sends to the MQTT broker, is to get the list of workers (and their

capabilities). It then waits for that response, according to the timeout set by the "GetWorkerCapabilitiesTimeout" property. During this time, all the available workers, which respond, will be added to the list, to be used subsequently when allocating tasks for each these workers, which are the FindSubControl action parts. Task allocation is a complex operation which has to take into account the worker OS, its available fonts, the plugin setting to target a specific OS (see the "TextRenderingOS" property), for both "usual" text and also for text found in pmtv files as primitives. Although the task allocation algorithm is predictable, the available workers and the order they respond to the plugin, make the allocation itself generate different results on every run (this can be observed from the $Worker[#].Name$ variables). If the number of tasks is less than the number of available workers, (obviously) not all workers are going to receive work. If it is greater, some workers will receive more than the others (in a round-robin fashion). The only thing that prevents the tasks to be allocated to the exact same workers, is the randomness with which they respond to the availability request. The bmp files are a small exception to this, since they are font-independent, so a random number is used to offset the worker index on every plugin execution. Unfortunately, when allocating font-based tasks (text and pmtv files), this is not available. The actual load-balancing of task allocation, happens across multiple plugin runs.

The second configurable timeout is the "FindSubControlWorkerTimeout" property, which represents how much the plugin waits for a worker to respond to a task (which may contain multiple font profiles, bmp and pmtv files). To take into account the transmission duration, the actual FindSubControl action will have to execute in less time, set as the difference between the value configured in the "FindSubControlWorkerTimeout" property and the "FindSubControlTimeoutDiff" property. If the configured difference is smaller than 100ms, the plugin limits it to 100ms. This is to prevent invalid values. A reasonable minimum value would be 500ms.

When the "TextRenderingOS" property is set to Win+Lin (default), the plugin can allocate tasks to both types of workers, again filtered by their available fonts. When set to Win or Lin, the tasks are allocated to those types of workers only. If there are no available workers for that OS, the plugin action fails with an error.

Currently, the "ListOfMultiValuePropertyNames" is reserved for future implementations.

Many other settings involve archive compression (the FindSubControl action and its files are archived by the plugin, sent to workers and extracted there), which can be switched off from the "UseCompression" property. Whether to use compression or not, is a user decision, based on the action content and expected latency. There are two available compression algorithms, currently used by the plugin, which are Lzma and Zlib, configured from the "CompressionAlgorithm" property. The default values for the Lzma parameters were experimentally obtained, based on the recommended values.

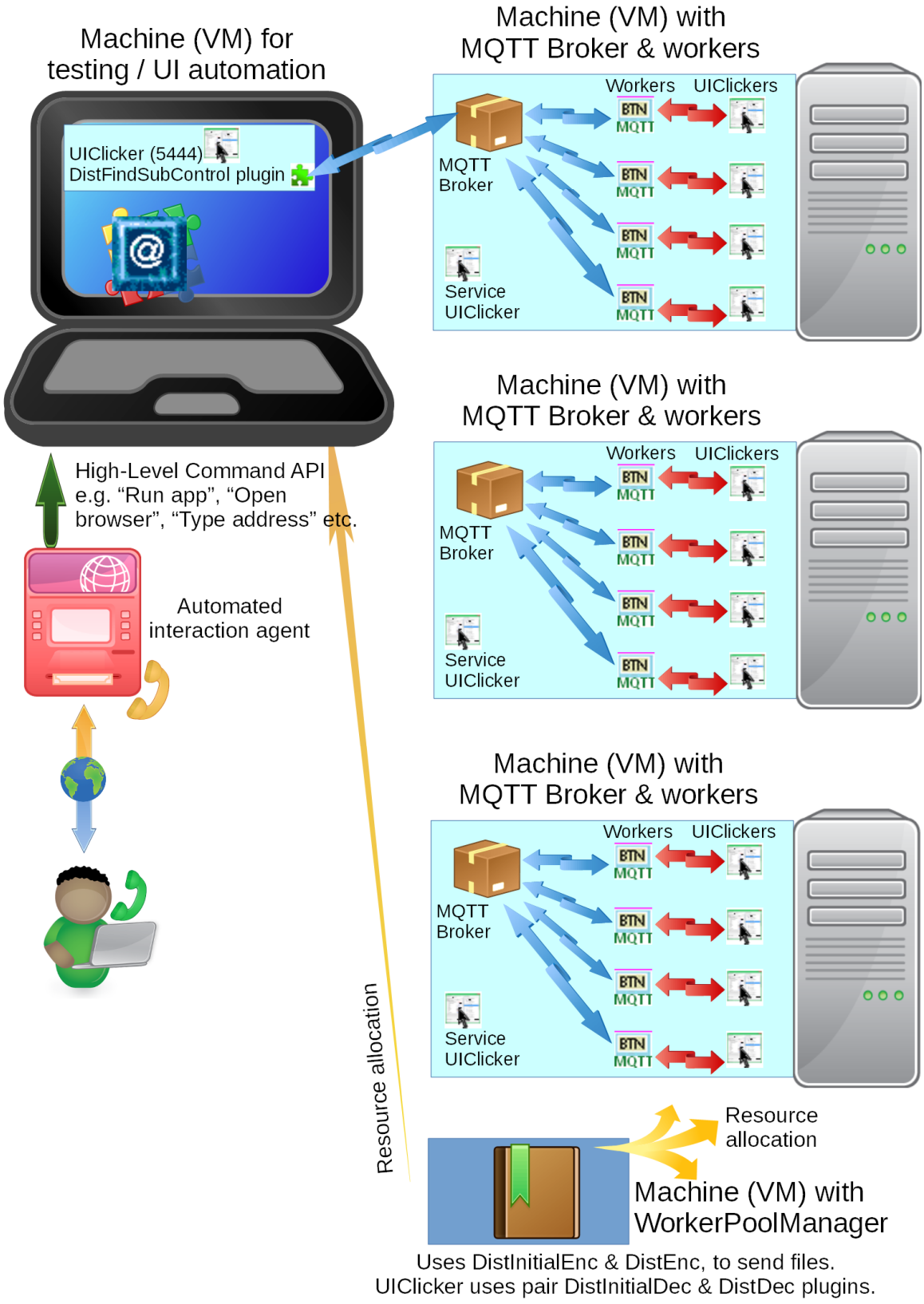
Before the actual execution of the FindSubControl action, various variables can be updated at worker side. Their values are read from the UIClicker instance, which runs the plugin. These variable are comma-separated names and configured to the "VariablesForWorkers" property.

There is also a file caching feature, which prevents sending files, which are already present at workers, from previous executions. In the current implementation, although the files are already present at workers, they also have to be present at the UIClicker instance, which runs the plugin.

Because the $AppDir$, $TemplatesDir$ and $SelfTemplatesDi$ (special) variables are different between, the UIClicker running the plugin and the worker UIClicker instances, some of the bmp and pmtv files won't be found and the action will fail. These variables are automatically evaluated. The EvaluateFileNameBeforeSendingProperty configure whether to evaluate the file paths, before archiving the files, which contain other variables from UIClicker's list of variables.

The main/current plugin limitations are the updating multiple finding results (the "result" images) and the worker/UIClicker implementations for Linux.

This plugin comes with various tools and helper plugins, for generating MQTT credentials, sending files, managing resources (allocating workers) etc. One possible use case of this plugin is depicted in the following diagram, where a machine, used for UI automation (or testing), exposes an API for interacting with a web page. If multiple such machines are to be run in parallel, then multiple VMs, with workers, UIClickers and a broker, are avaialable for each machine, where the plugin runs.



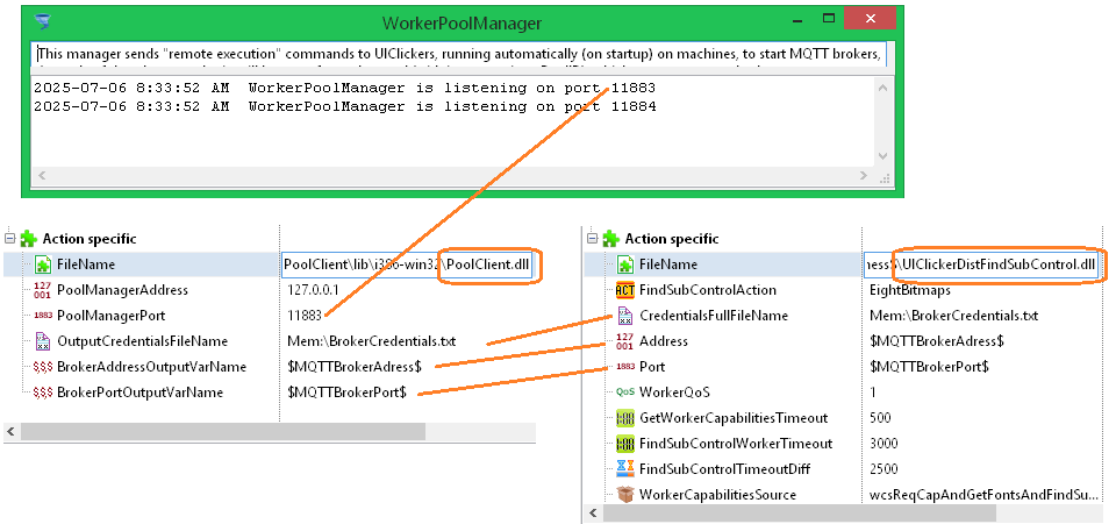
These resources can be managed by the WorkerPoolManager tool (in work at the time of writing).

This tool is capable of starting brokers, workers and their UIClicker instances on a machine, via a "service UIClicker", i.e. an additional UIClicker instance, used exactly for this purpose (see the above schematic). WorkerPoolManager can also generate unique MQTT usernames and passwords for both the plugin and the workers, on every such VM allocation. It expectes the Mosquitto broker to be installed on the VM, where the broker is to be run. Unfortunately, the current implementation of the used MQTT library does not support TLS. Although the archives, used for sending FindSubControl actions and their files support encryption, this is not used for now.

There is a helper plugin, called PoolClient, which gets the generated credentials from WorkerPoolManager. This plugin is run by the same UIClicker instance ("Dist"), which runs the UIClickerDistFindSubControl plugin. WorkerPoolManager returns the broker address, broker port and the credentials. The plugin saves those credentials in a file, called by default "BrokerCredentials.txt", and placed in In-Mem FS (see screenshot below).

The PoolClient plugin connects to WorkerPoolManager via HTTP and for that, it requires a pool username and a pool password (for the actual request), which it reads from a text file, called by default "PoolCredentials.txt", and placed in In-Mem FS. This file is sent to UIClicker by WorkerPoolManager, right after the VMs are created and WorkerPoolManager receives a "MachineOnline" (Get) request, which contains the addresses of the "worker" VM and "Dist" VM.

After calling the PoolClient plugin, the UIClickerDistFindSubControl plugin should have all it needs to connect to the right broker.



The BrokerParams plugin is intended to be run by the "service UIClicker", on the machine where the broker is running. It is called by WorkerPoolManager, to edit broker config files. It relies on variables like $ConfFile$, $PasswordFile$ and $BrokerPortNumber$ and expects the files pointed to by $ConfFile$ and $PasswordFile$ variables, to exist on disk. The "ConfFile" is a typical Mosquitto configuration file, which will have its "listener" and "password_file" fields updated by this plugin. The "PasswordFile" is a text file, generated/updated by the "mosquitto_passwd.exe" tool, which comes with Mosquitto. This tool is started by WorkerPoolManager, right before running the BrokerParams plugin.

In the same repository, there are also tools for encryption and decryption of plugins and their files (the UIClickerDistFindSubControl plugin in this case). These are DistInitialEnc and its pair plugin DistInitialDec, which are used to send the DistDec plugin to the "service UIClicker". Its pair tool is DistEnc.

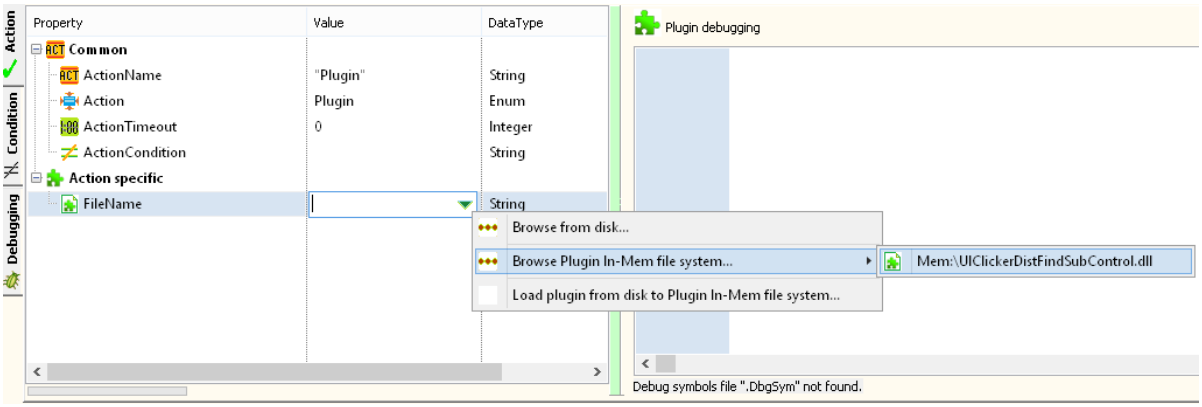
A possible execution flow would be:

1. DistInitialEnc sends DistInitialDec plugin (unencrypted).
2. DistInitialEnc sends DistDec plugin, as an archive, which is extracted by DistInitialDec plugin.
3. DistEnc sends UIClickerDistFindSubControl plugin, which is extracted by DistDec plugin.
4. The UIClickerDistFindSubControl plugin is now in UIClicker's memory and can be used.

```
.\DistInitialEnc.exe --ClickerClient
C:\UIClicker\ClickerClient\ClickerClient.dll --PluginToBeSent
C:\UIClickerDistFindSubControlPlugin\DistInitialDec\lib\i386-
win32\DistInitialDec.dll --PluginToBeSentDestName DistInitialDec.dll
--UIClickerAddress 127.0.0.1 --UIClickerPort 5444

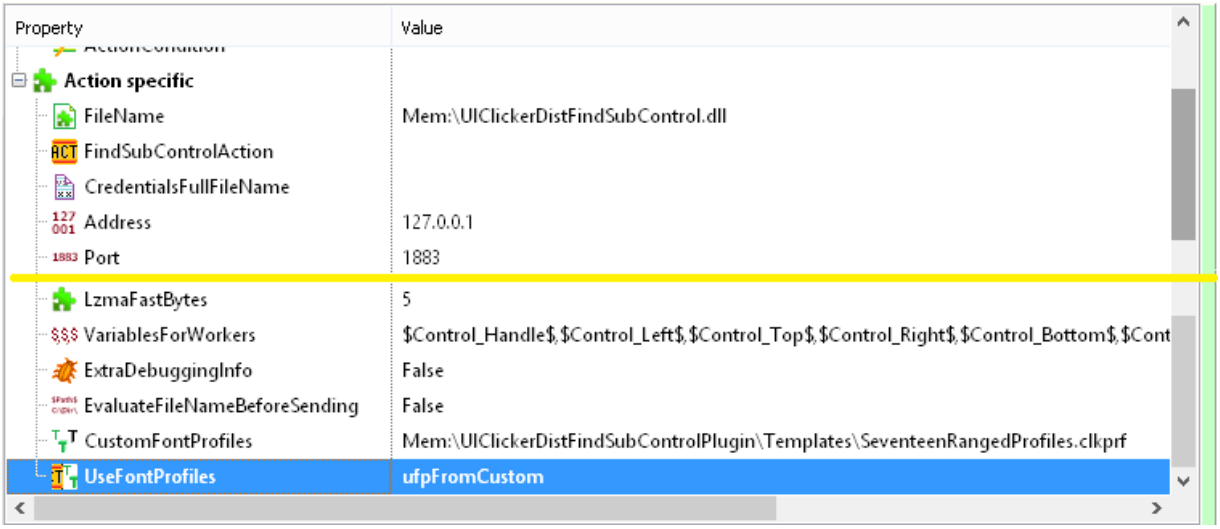
.\DistInitialEnc.exe --ClickerClient
C:\UIClicker\ClickerClient\ClickerClient.dll --PluginToBeSent
C:\UIClickerDistFindSubControlPlugin\DistDec\lib\i386-win32\DistDec.dll
--PluginToBeSentDestName DistDec.dll --UIClickerAddress 127.0.0.1
--UIClickerPort 5444

.\DistEnc.exe --ClickerClient C:\UIClicker\ClickerClient\ClickerClient.dll
--PluginToBeSent C:\UIClickerDistFindSubControlPlugin\lib\i386-
win32\UIClickerDistFindSubControl.dll --PluginToBeSentDestName
UIClickerDistFindSubControl.dll --UIClickerAddress 127.0.0.1 --UIClickerPort
5444 --DecryptionPluginName "DistDec.dllarc|Mem:\DistDec.dll"
```



In the same way, the BrokerParams can be sent to the "service UIClicker" on the workers machine.

Since the current implementation has no authentication mechanism, these tools and plugins (and their implementation) are for example purposes only. Users should modify/improve their implementation before use.



As examples, here are some commands for using the above described tools.
The following commands are processed by WorkerPoolManager, which expects a (service) UIClicker to be running on the worker machine (10.0.2.7) and listening, and the other, a (dist) UIClicker on a different machine (10.0.2.x or 10.3.4.5).

1. Adding a broker/worker machine and a dist machine to the list in WorkerPoolManager:

```
http://127.0.0.1:11884/MachineOnline?
WorkerMachineAddress=10.0.2.7&MachineOS=Win&DistPluginMachineAddress=10.0.2.8
```

2. Adding a second dist machine, which will be connected to an existing broker/worker machine:

```
http://127.0.0.1:11884/MachineOnline?
WorkerMachineAddress=10.0.2.7&MachineOS=Win&DistPluginMachineAddress=10.0.2.9
```

Since the worker machine has the same address as in the first example, a new worker machine will not be added to list. The dist machine will be added, only if WorkerPoolManager is configured to allow multiple brokers / machine (i.e. multiple pools / machine).

3. Removing a worker machine from list:

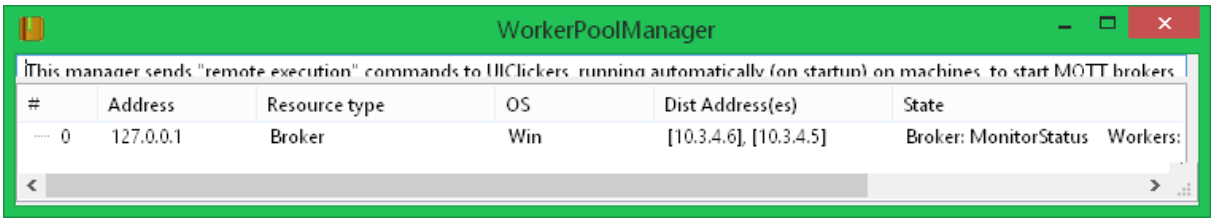
```
http://127.0.0.1:11884/RemoveWorkerMachine?WorkerMachineAddress= 10.0.2.7
```

4. Removing a dist machine from an existing worker machine entry:

```
http://127.0.0.1:11884/RemoveDistMachine?
WorkerMachineAddress=10.0.2.7&DistPluginMachineAddress=10.0.2.9
```

5. Adding another dist machine to an existing worker machine entry:

```
http://127.0.0.1:11884/MachineOnline?
WorkerMachineAddress=10.0.2.7&MachineOS=Win&DistPluginMachineAddress=10.3.4.5
```



A FindSubControl action provides a mode of operation, in which no bmp comparison is done, instead it renders text from the configured text profiles and keeps the generated bmp files in memory (In-Mem FS) . This is configured from the "MatchBitmapAlgorithm" property, when set to "mbaRenderTextOnly". Such a mode of operation is required exactly when using the UIClickerDistFindSubControl plugin, and the antialiasing algorithms (or their implementations) are OS dependent or hardware dependent, leading to different results between the machine where the plugin is running and the machine(s) where the workers and their UIClickers are running. Thus, the text rendering is done at the plugin side, where it is expected to look the same as the searched text, but the actual searching is done at workers' side. The rendering itself is not the real bottleneck in a FindSubControl action, so if this plugin fails with an error saying that the text could not be found, although it is clearly there, users should have a look at reconfiguring the FindSubControl action. In this case, there would be two FindSubControl actions, one for rendering (set to enabled), executed before the plugin, and the other (set to disabled), executed by the plugin, as described before.

If multiple font profiles are required, and they are not easy to configure from a FindSubControl action, then the plugin can load a clkprf file (custom font profiles), and update the loaded action, in memory, with the file content. Currently, this file has to be manually edited (until an editor is implemented). Such a file supports describing multiple font profiles, each with multiple ranges of the available fields.

It has to be mentioned, that UIClicker has a hardcoded limit for the number of font profiles it accepts when executing a FindSubControl action in server mode. This applies to each worker.
See "MatchBitmapText.Count is out of range." error message in ClickerActionProperties.pas.

Clkprf content example ("\Tests\TestFiles\NineteenRangedProfiles.clkprf"):

```
[First]
ForegroundColor=000000
BackgroundColor=008000..008001
FontName=Verdana,Segoe UI
FontSize=11..13
Bold=0
Italic=0
Underline=0
StrikeOut=0
FontQuality=0
FontQualityUsesReplacement=0
FontQualityReplacement=
ProfileName=Default
CropLeft=
CropTop=
CropRight=
CropBottom=
IgnoreBackgroundColor=0

[Second]
ForegroundColor=000000,000007..000008
BackgroundColor=888888
FontName=Tahoma
FontSize=8
Bold=0
Italic=0
Underline=0
StrikeOut=0
FontQuality=3
FontQualityUsesReplacement=0
FontQualityReplacement=
ProfileName=Default
CropLeft=
CropTop=
CropRight=
CropBottom=
IgnoreBackgroundColor=0

[Third]
ForegroundColor=008807..008509
BackgroundColor=EEEEEE
FontName=Fixedsys
FontSize=8
Bold=True
Italic=False
Underline=False
StrikeOut=False
FontQuality=4
FontQualityUsesReplacement=0
FontQualityReplacement=
ProfileName=Default
CropLeft=
CropTop=
CropRight=
CropBottom=
IgnoreBackgroundColor=0
```

The sections, are called profile groups, because each of them can result in multiple font profiles. Each field can have a value as an enumeration, or a range of values, or a combination of both. See for example: `ForegroundColor=000000,000007..000008`. This translates to 3 font profiles with `ForegroundColor` of `000000`, `000007` and `000008`. The two fields, `ForegroundColor` and `BackgroundColor` are interpreted as hex colors, in two digit BGR format (BBGGRR). In the `Third` group, the `ForegroundColor` is set to `008807..008509`, which defines 4 values for the green channel (88 to 85) and 3 values for the red channel (7 to 9). This results in a maximum of 4 profiles. Because of a limitation of the current implementation, the generated values for channel(s), with the smaller range (red in this case), may not have the proper endpoints. This is caused by rounding real numbers, when scaling the interval. In this case, the generated values will be: 008807, 008708, 008608, 008508. The last value is 008508, instead of 008509.

If a profile group has multiple fields with enumerations and ranges, the resulted number of generated font profiles is going to be the product of all the possible combinations. For example, the `First` group has two values for `BackgroundColor`, two values for `FontName`, and three values for `FontSize`. This results in a total number of 2*2*3 = 12 combinations for this group. From the above example, the total number of font profiles, generated by the three groups, is 12 + 3 + 4 = 19.