Installation Folders and Utilities Reference

Understand folder structure and built-in utilities.

Reference **Installation** Folders | Licensing | Updates | SecureGateway | Web Server

Overview

Directory structure and file locations created by FrameworX installation.

Executables Installation
User Document Folders
Public Document Folders
Log Locations
Log File Size and Retention
Installed Utilities
AutoStartClient.exe
Custom Installation Paths

Executables Installation

Default location: `C:\Program Files\Tatsoft\FrameworX\`

Custom paths can be selected during installation.

User Document Folders

Created on first execution under `<User>\Documents\FrameworX\`:

Folder	Purpose
Demos	Demo solutions from installer
Solutions	Default location for new solutions
ToolsSettings	Product tool configurations
Utilities	Command-line utilities

Public Document Folders

Created under `C:\Users\Public\Documents\FrameworX\`:

Folder	Contents
MachineSettings	Settings for all users on machine
SystemSolutions	Library.dbsln (Symbol Library), Monitor.dbsln (Monitoring tools)
Templates	Solution templates - add .dbsln files for custom templates
TraceLogs	Diagnostic logs and exception reports
Exchange	Import/Export/Upload/Download operations

Log Locations

The runtime and supporting services write trace logs to a configurable location. The default is user-visible under `Public\Documents`, and enterprise deployments can relocate to `ProgramData`:

Location	When used	Notes
`C:\Users\Public\Documents\FrameworX\TraceLogs\`	Default for interactive and desktop installs	User-visible, shared across all users on the machine

`C:\ProgramData\Tatsoft\FrameworX\Logs\`	Recommended for enterprise server deployments (configurable)	Matches Windows conventions for operational data; automatically scanned by Splunk forwarder, Datadog agent, Azure Monitor, and ELK agents
`<custom>`	Configurable via Solution Settings	Set the log folder per solution or per deployment

Install logs are written separately:

- Silent install: path specified by `/LOG=<path>` command-line switch
- Interactive install: `%TEMP%\Setup Log YYYY-MM-DD #xxx.txt`

Windows Event Log: Windows logs basic service lifecycle events (start, stop, unexpected termination) to the **System** event log under source `Service Control Manager` for any installed FrameworX Windows service. SCOM, Azure Monitor, Datadog, and similar Windows-native monitoring tools pick these up automatically without any custom configuration. Application-level events (solution load errors, license failures, module exceptions) are written to the runtime TraceLogs — check there for detail when an Event Log entry indicates a failed start.

For the enterprise deployment workflow, see IT Deployment RunBook.

Log File Size and Retention

Each runtime process (TServer, TStartup, TRunModule, TRichClient) can cap the size of its active log file and limit how many daily archives are kept. Both settings are optional and read from the process's JSON configuration file (`<ProcessName>.json` — for example `TServer.json`) under `appSettings.Logging`. When the file or the section is absent, the default behavior applies: no size limit, and every daily archive is kept.

Setting	Type	Default	Effect
MaxFileSizeMB	Integer (MB)	`0`	`0` or absent = no limit. When greater than 0 and the current day's log file reaches this size, the process writes one final notice (*"Log file size limit of N MB exceeded; logging to this file is stopped."*) and stops writing to that file for the rest of the day. Logging resumes automatically in the next day's file.
KeepAllZips	Boolean	`true`	`true` or absent = keep every daily `.zip` archive. `false` = keep only the most recent archive and delete the older ones.

Example `TServer.json`:

```
{
  "appSettings": {
    "Logging": {
      "MaxFileSizeMB": 50,
      "KeepAllZips": false
    }
  }
}
```

These settings apply only to the trace/execution log files described under **Log Locations** above, and only when logging is enabled for the process (the `SaveLog` startup setting). Each process reads its own JSON file, so limits can differ per process.

Installed Utilities

The `Utilities` folder holds the stand-alone utilities below. Other command-line tools ship in the product root or in the versioned `fx-10` folder, as shown in the Location column.

Utility	Location	Function
AutoStartClient.exe	`fx-10\Utilities\`	Auto-start web pages or thin clients based on server status. See the section below.
CertificateConverter.exe	`fx-10\Utilities\`	Convert .pem to .pfx certificates for MQTT drivers
SolutionCreator.exe	`fx-10\`	Headless solution build and creation utility
TDatabaseBuilder.exe	`fx-10\`	SQLite database administration

Windows Service management and the AI/MCP listeners have their own reference pages:

- **TManageServices.exe** (product root) installs and uninstalls the FrameworX Windows services (TStartup, TWebServices, TSecureGateway, TMQTTBroker, THardkey, RuntimeMCPHttp), including `/removeallservices`. See TManageServices CLI.

- **DesignerMCPHttp.exe**, **RuntimeMCPHttp.exe** and **ConsoleMCPHttp.exe** (`fx-10\net10.0\`) expose the AI integration listeners. See MCP and Claude Setup and AI Designer Connector.

AutoStartClient.exe

AutoStartClient starts a client automatically once a runtime server is ready to accept it. The utility polls the configured server and launches the configured action as soon as that server can actually serve a client, which avoids the common failure of a Windows Startup-folder shortcut firing before the runtime is ready.

AutoStartClient covers the client side only. The runtime itself starts at boot as a Windows Service: see Runtime Startup Reference, Enabling AutoStart.

The Windows Startup folder only runs at logon. On a station that sits without a logged-on user, configure Windows automatic logon as well, otherwise the client never starts.

Settings are read from `AutoStartClient.exe.config`, in the same folder as the executable. For that reason the utility is started through a **shortcut**, never a copy of the executable: a copy placed elsewhere does not find its configuration file.

Run AutoStartClient at logon

1. Open the `fx-10\Utilities` folder of the FrameworX installation (by default `C:\Program Files\Tatsoft\FrameworX\fx-10\Utilities`), right-click `AutoStartClient.exe` and select **Create shortcut**. If Windows offers to place the shortcut on the desktop instead, accept.
2. Press **Win+R**, type `shell:common startup` and press **Enter**. This opens the Startup folder for all users of the computer. Moving a file into it requires administrator rights; for a single user, use `shell:startup` instead.
3. Move the shortcut into that Startup folder.
4. Right-click the shortcut, select **Properties**, and set **Start in** to the `Utilities` folder from step 1.
5. If nobody logs on to the station, configure Windows automatic logon, as described above.

To test it, sign out of Windows and sign in again: the AutoStartClient window opens, or its tray icon appears when `hidden` is set to `true`.

What each poll does

1. **Checks whether a client is already running.** Any process named in `clientprocessnames` counts — by default the Rich Client and the Smart Client — whatever started it, so the configured action is not invoked again while a client is on screen. This includes a Rich Client opened by the solution itself (the **Displays** module under Runtime Startup): on that computer the utility then starts nothing. The utility also tells a client that is *connected* to a server from one that is merely running, so a client stopped on an error dialog is not counted as a working client.
2. **Checks whether the server is ready for a client**, on each configured address in turn: the address answers, the runtime reports that it has finished starting, and — on a redundant pair — that node reports it is the **active** one. A node that is up but running as standby refuses every client, so it is not used until it takes the active role.
3. **Starts the configured action** for the first address that passes: `batch` / `batch2`, or `url` / `url2`. A batch file is started without a console window, so nothing flashes over the operator screen.

Reading the status window. With `hidden` set to `false`, the utility lists each step. While the runtime is not up yet it shows `Waiting remote server starts (127.0.0.1:3101)... [127.0.0.1:3101: not accepting connections]`; once a client is up it shows `Client is running and connected...`. A configuration value it cannot use is reported in a dialog when the utility starts.

A process guard in the batch file is no longer needed. Earlier versions invoked the configured action on every poll, and the common workaround was a `tasklist` check inside the `.bat` to stop clients accumulating. From 10.1.5e the utility performs that check itself and calls the action only when no client is running at all, so by the time the batch file executes there is nothing for the guard to find. It is redundant, not harmful, and leaving it in place changes nothing. Remove it or keep it as you prefer.

What the guard never did, in any version, was retry: a client stopped on a connection-error dialog is handled by `recoverstuckclient`, described below, not by anything in the batch file.

Key	Default	Function
`ip`	`127.0.0.1`	Address of the server to monitor. For a redundant pair, fill in both `ip` and `ip2` with the two node addresses, so the client is started as soon as either node is active.
`port`	`3101`	Port of the server named by `ip`. Must match the port that runtime actually listens on.
`batch`	empty	Path to a `.bat` file executed when the server named by `ip` is ready. Use this to launch a Rich Client, or the Smart Client without the browser prompt (see below). Leave `url` empty when `batch` is used.
`url`	empty	Full web address, including `http://` or `https://`, opened in the `browser` when the server named by `ip` is ready. Use this instead of `batch` for an HTML5 client. A bare address such as `127.0.0.1` is rejected when the utility starts, with *"Invalid 'url' or 'ip' parameter: Invalid URI"*. A Smart Client address opens through the browser, which asks for confirmation on every launch.
`ip2` / `port2` / `batch2` / `url2`	empty / `3101`	Second server and its corresponding action. Use for a redundant pair, so the client starts against whichever node is active first.

`timer`	`30`	Polling interval in seconds. The utility keeps checking until a server is ready.
`hidden`	`false`	Run the utility without a visible window. Recommended on an unattended station; notices are then raised as tray notifications.
`browser`	`msedge`	Browser used to open `url`. Ignored when `batch` is used.
`username` / `password`	`guest` / empty	Credentials used for the readiness handshake with the server, and passed to the client started through `url`. A client started through `batch` takes its parameters from the batch file.
`connectiontimeout`	`5`	Connection timeout, in seconds, passed to the client started through `url`. It does not change how long the utility itself waits for a server to answer. A client started through `batch` takes its parameters from the batch file.
`relaunchclient`	`true`	Start the client again after its process closes. With it on, the utility acts as a viewer watchdog for an unattended station: a client that exits, crashes, or is closed by an operator comes back on the next poll. With it off, the client is started once and not started again.
`clientprocessnames`	`TRichClient, TSmartClient`	Process names counted as a running client, comma separated (the `.exe` suffix is optional). Change it only when the configured action starts a viewer under a different process name.
`recoverstuckclient`	`true`	Close a client that **this utility started** and that never established a connection, then run the full readiness check again on the next poll. A client the utility did not start is never closed.
`stuckclienttimeout`	`120`	Seconds a client started by this utility may run without connecting before `recoverstuckclient` closes it. Values below 30 are raised to 30. Ignored when `recoverstuckclient` is turned off.

A client that stops responding. If a client is running but never establishes a connection, the utility reports it in its own message list and as a tray notification. By default it is then closed and replaced: only a client the utility itself started is ever closed, and only after `stuckclienttimeout` seconds with no connection, after which the next poll runs the full readiness check again before starting a new one. A viewer opened by an operator, or by any other tool, is reported but never closed. Set `recoverstuckclient` to `false` to have the utility report such a client and leave it in place, which means an unattended station stays on the error dialog until somebody dismisses it.

Which client to start

- **Rich Client (recommended for unattended stations):** set `batch` to a batch file that runs `TRichClient.exe /ip1:<ServerIP> /port1:3101`, as in the example below. Requires FrameworX installed on the station.
- **HTML5 client:** set `url` to `http://<ServerIP>/html5`. Nothing to install on the station.
- **Smart Client:** the client itself is not installed on the station; it needs .NET Framework 4.8 (see Client Types Reference). Through `url`, the browser opens it and asks for confirmation on every launch (*"Do you want to open TSmartClient.application?"*), so an unattended station stops on that dialog. Use `batch` instead, as described in Smart Client without the browser prompt below.

Example: Rich Client

The most common setup: the station starts the Rich Client as soon as the runtime is ready. `batch` takes the path of a batch file, never the `TRichClient.exe` command line itself. The client parameters go inside the batch file.

First the batch file. Create it yourself, for example as `C:\Program Files\Tatsoft\FrameworX\fx-10\Utilities\StartRichClient.bat`:

```
@echo off
start "" "C:\Program Files\Tatsoft\FrameworX\fx-10\TRichClient.exe" /ip1:127.0.0.1 /port1:3101
```

- Replace `127.0.0.1` with the server address when the runtime runs on another computer. Use the same address and port as `ip` and `port` in the configuration file.
- The empty `""` after `start` is the window title. Without it, `start` reads the quoted executable path as the title and opens an empty console window instead of the client.
- For a redundant pair, give both node addresses: `/ip1:192.168.1.1 /port1:3101 /ip2:192.168.1.2 /port2:3101`. The Designer generates this command line ready to copy under **Runtime / Startup / Redundancy**; see Redundancy Reference.
- To check the batch file on its own, double-click it: the Rich Client opens and connects.

Then the matching `AutoStartClient.exe.config`:

```
<appSettings>
  <add key="ip" value="127.0.0.1"/>
  <add key="port" value="3101"/>
  <add key="url" value=""/>
  <add key="batch" value="C:\Program Files\Tatsoft\FrameworX\fx-10\Utilities\StartRichClient.bat"/>
  <add key="timer" value="30"/>
  <add key="hidden" value="true"/>
</appSettings>
```

Write the `batch` path without quotes, even when it contains spaces. The configuration file is XML, and quotes inside `value="..."` make the whole file invalid. For a redundant pair, also set `ip2` and point `batch2` at the same batch file: the client command line already carries both addresses.

Smart Client without the browser prompt

Optional. Use this when a station runs the Smart Client and nobody is there to confirm the browser prompt. Instead of opening the Smart Client address in a browser, the batch file hands it straight to Windows ClickOnce, so no browser window and no prompt appear.

First the batch file. Create it yourself, for example as `C:\FrameworX\StartSmartClient.bat`:

```
@echo off
rundll32.exe dfshim.dll,ShOpenVerbApplication http://192.168.1.1:3101/TSmartClient.application?ip1=192.
168.1.1^&port1=3101^&connectiontimeout=5^&userName=guest
```

- The address is the server's `TSmartClient.application` followed by the client parameters: `ip1` and `port1` for the server, `connectiontimeout` and `userName`. For a redundant pair add `ip2` and `port2`, as in the Smart Client URL on Redundancy Reference.
- Write every `&` in the address as `^&`. In a batch file a plain `&` ends the command, and the rest of the address is lost.
- To see the exact address your server hands out, open the Smart Client once through the browser and copy the `TSmartClient.application` download link.
- To check the batch file on its own, double-click it: the Smart Client opens with no browser window.

Then the matching `AutoStartClient.exe.config`:

```
<appSettings>
  <add key="ip" value="192.168.1.1"/>
  <add key="port" value="3101"/>
  <add key="url" value=""/>
  <add key="batch" value="C:\FrameworX\StartSmartClient.bat"/>
  <add key="timer" value="30"/>
  <add key="hidden" value="true"/>
</appSettings>
```

Leave `url` empty: the batch file replaces it. `TSmartClient` is already in the default `clientprocessnames`, so the utility recognizes the Smart Client as the running client and, with `relaunchclient` on, starts it again if it closes. The first time the Smart Client runs on a station, Windows may ask once to trust the application.

Custom Installation Paths

For Linux/Docker deployments, create `custompaths.txt` in installation folder:

```
MyDocuments=<folder>
CommonDocuments=<folder>
CommonApplicationData=<folder>
```

Typically points to memory areas outside Docker image for read/write access.

In this section...
