Runtime Troubleshooting Reference

Resolve common runtime issues.

Reference **Solution** **Runtime** UI | Engine | Troubleshooting | Diagnostics | Client-Server

---

Runtime Troubleshooting (Reference):

Comprehensive troubleshooting guidance for runtime issues, best practices for environment isolation, and solutions to common problems.

Best Practices for Runtime Isolation
Troubleshooting Runtime Issues
Common Issues and Solutions
Troubleshooting by Symptom
Diagnostic Resources
Emergency Response
Platform-Specific Issues
Windows Desktop Heap Exhaustion (Session 0)
Support Resources

Best Practices for Runtime Isolation

1. Use Separate Environments

Working in separate environments for development, testing, and production prevents configuration changes from affecting the runtime environment until tested and deployed.

2. Use Containers

Docker and similar technologies create isolated containers for different solution parts, ensuring modifications in one container don't impact others.

3. Use Version Control

Version control manages solution configurations and code changes, enabling experimentation, rollback capabilities, and change tracking without runtime impact.

4. Use Access Control

Access control mechanisms restrict runtime environment modifications to authorized personnel only, reducing accidental disruptions or unauthorized access.

Troubleshooting Runtime Issues

Troubleshooting Steps

1. Review application logs for error messages or warnings
2. Use debugging tools to inspect application state at runtime
3. Monitor system resources (CPU, memory, disk usage) for bottlenecks
4. Consult documentation or support resources for known issues
5. Verify appropriate execution profiles for development lifecycle stages
6. Test configuration changes before production deployment
7. Document custom configurations or modifications
8. Keep runtime environment updated with security patches

Common Issues and Solutions

Performance Bottlenecks

Problem: Slow solution execution from inefficient code or inadequate hardware resources.

Solution: Optimize code, allocate more resources, or employ parallel processing techniques.

Configuration Errors

Problem: Incorrect solution configuration settings leading to unexpected behavior or failure.

Solution: Review configuration files, verify parameters, and ensure documentation adherence.

Runtime Environment Mismatch

Problem: Incompatibilities between development and production environments causing runtime issues.

Solution: Ensure consistent dependencies across environments; use containers or virtual machines for isolation.

Module Conflicts

Problem: Conflicts between modules causing unexpected behavior or crashes.

Solution: Check for duplicate or outdated modules; ensure module compatibility.

Database Connectivity Issues

Problem: Connection problems from network issues, incorrect credentials, or misconfigured settings.

Solution: Verify database connection settings, ensure server accessibility, and check credentials.

Client-Side Display Errors

Problem: Issues from incompatible browsers, outdated JavaScript libraries, or misconfigured display settings.

Solution: Ensure browser version support, update libraries, and verify display settings.

Hot Reloads Not Applied

Problem: Hot Reloads not being applied correctly to running solution.

Solution: Confirm solution configuration for online changes; verify update mechanism functionality.

Build and Publish Failures

Problem: Errors during build/publish from incorrect settings, code issues, or dependency problems.

Solution: Review build/publish configuration, fix code issues, and resolve all dependencies.

Troubleshooting by Symptom

Symptom	Check	Action
Solution won't start	License status	Verify license validity
Slow performance	Resource usage	Monitor CPU/memory
Data not updating	Module status	Check device connections
Display errors	Browser console	Review JavaScript errors
Communication failures	Network connectivity	Test port accessibility
Unexpected behavior	Execution profile	Verify correct profile

Diagnostic Resources

Log Locations

Component	Default Location	Content
Runtime	Logs\Runtime\	Execution events
Modules	Logs\Modules\	Module-specific logs
Client	Logs\Client\	Client connections
System	Logs\System\	System-level events

Performance Monitoring

Monitor these key indicators:

- CPU utilization per module
- Memory consumption trends
- Network throughput
- Database query times
- Tag update rates

Emergency Response

Severity Levels

Level	Description	Response Time
Critical	Production down	Immediate
High	Major functionality impaired	< 1 hour
Medium	Minor functionality affected	< 4 hours
Low	Cosmetic or non-critical	Next business day

Recovery Procedures

1. **Assess Impact** - Determine scope and severity
2. **Isolate Problem** - Identify affected components
3. **Apply Fix** - Implement solution or workaround
4. **Verify Resolution** - Confirm normal operation
5. **Document Incident** - Record cause and solution

Platform-Specific Issues

Windows Service

Issue	Solution
Service won't start	Check event viewer for errors
Permission denied	Verify service account rights
Port conflicts	Change port or stop conflicting service

Linux/Docker

Issue	Solution
Container exits	Check container logs
Permission errors	Verify file ownership
Resource limits	Adjust container constraints

Windows Desktop Heap Exhaustion (Session 0)

On Windows, every process that creates windows or graphical handles draws from a fixed-size memory region called the **desktop heap**. The desktop heap is not per-process: it is shared by all processes running in the same window station. When the FrameworX Runtime runs as a Windows service it lives in **Session 0**, which uses the small non-interactive desktop heap (default 768 KB). Large solutions that run many modules in Session 0 can exhaust this shared pool, and modules then fail to start or crash at runtime.

When it happens

The desktop heap is consumed by handles (windows, menus, hooks) created by any process in the session, including the operating system and any other software installed on the server. A solution with a high module count, for example dozens of communication channels, creates many handle-consuming processes at once. As the shared pool approaches its ceiling, new handle allocations fail and the affected modules stop or restart.

Typical signature (Windows Event Viewer, Application log), repeating for each affected module:

- **Event ID 1000** (Application Error): the faulting application is the runtime module process (for example `TRunModule.exe`), the faulting module is `KERNELBASE.dll`, and the exception code is `0xe0434352`.
- **Event ID 1026** (.NET Runtime), logged at the same timestamp: the unhandled managed exception is `System.ComponentModel.Win32Exception`, thrown while the module creates a window resource. The call stack ends in window-resource creation, typically one of:
  - `System.Windows.Forms.NativeWindow.CreateHandle(...)` (creating a window handle), or
  - `System.Windows.Forms.NativeWindow+WindowClass.RegisterClass()` (registering a window class).
- The exception code `0xe0434352` is the standard code Windows records for **any unhandled .NET (CLR) managed exception**, not a desktop-heap-specific code (see CLR Exception E0434352, Microsoft Learn). What identifies this specific failure is the `Win32Exception` raised from window-resource creation in the Event 1026 stack, together with the correlation to a high module count and execution as a service (Session 0).
- Symptoms appear only when the Runtime runs **as a service** (Session 0). The same solution run interactively uses a much larger desktop heap and usually will not reproduce the failure.

Resolution: increase the Session 0 desktop heap

Raise the non-interactive desktop heap by editing the `SharedSection` parameter in the registry, then reboot.

- **Key:** `HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystems\Windows`
- **Value:** the string contains `SharedSection=xxxx,yyyy,zzzz`. The three numbers are different heaps: `xxxx` is the shared heap, `yyyy` is the interactive desktop heap, and **`zzzz` is the non-interactive (Session 0 / service) desktop heap** that matters here.
- **Change:** if `zzzz` is the default `768`, raise it to `1536`. Record the original value first, it is your rollback. If it is already non-default, find out why before increasing it further.
- **Apply:** the change takes effect only after a reboot.

On 64-bit Windows, desktop heap allocations are dynamic, so a moderate increase does not risk the boot. Change only the third value and keep the other two as they are.

Multiplatform (.NET) solutions

The cross-platform .NET Runtime runs as a console process and creates close to zero windows, so a Multiplatform solution is much less likely to reach the desktop heap limit than the classic WinForms Runtime. It is less likely, not impossible: each instance still has one real companion window (`conhost.exe`), and any thread that calls `user32` or `gdi32` begins consuming desktop heap. With enough instances the limit can still be approached, so the same diagnosis and remedy apply.

Support Resources

Documentation

- Error code reference
- Known issues database
- Configuration guides

Community

- User forums
- Knowledge base
- FAQ sections

Professional

- Technical support tickets
- Remote assistance
- On-site consultation

In this section...
