jDeploy 6.1 Release Notes

Table of Contents

Overview

jDeploy 6.1 is an incremental release that builds on the multi-modal foundation of 6.0. The headline additions are a manifest-driven uninstaller, a local development mode for testing apps without publishing, and Java remote debugging support via JDWP. Alongside these, 6.1 delivers a series of platform hardening fixes — improved shell profile handling on Linux, certificate pinning correctness for platform-specific bundles, and MCP server compatibility improvements.

All new features are opt-in. Existing packages continue to work exactly as before with no configuration changes required.

Uninstaller Command

CLI commands can now implement a built-in uninstaller. When a user runs `<command> uninstall`, the launcher reads the XML uninstall manifest that was written during installation and removes all installed files, services, PATH entries, registry keys (Windows), and shell profile exports

(Linux/macOS) in a controlled sequence.

```
{
  "jdeploy": {
    "commands": {
      "myapp-cli": {
        "description": "My App CLI",
        "implements": ["uninstaller"]
      }
    }
  }
}
```

The uninstaller follows a phased approach:

1. **Detect running GUI instances** — warns if the GUI app is still open
2. **Stop helper processes** — terminates the system tray helper
3. **Stop and uninstall services** — polls for service shutdown (up to 30 seconds) before proceeding
4. **Delete manifest files** — removes all files tracked in the uninstall manifest
5. **Clean up runtime directories** — removes JRE caches, package caches, lock files, and PID files
6. **Remove PATH and shell profile entries** — cleans up shell config on Unix, registry PATH on Windows
7. **Self-delete** — the launcher binary removes itself (on Windows, via a background PowerShell script to avoid locking issues)

Path deletion is restricted to known safe directories (`~/.jdeploy/`, `~/Desktop/`, `~/Applications/` on macOS, `~/.local/` on Linux, `%APPDATA%` and `%LOCALAPPDATA%` on Windows) to prevent accidental damage.

By default, the uninstaller prompts for interactive confirmation. Automation tools can pass `--jdeploy:interactive=false` to skip the prompt.

The uninstall trigger argument defaults to `"uninstall"` but can be customized via `uninstallTrigger` in the command spec, similar to `updateTrigger`.

Reference: PR #410, client4jgo PR #67

Local Development Mode

Developers can now test jDeploy-packaged applications locally without first publishing to npm or GitHub. By specifying `local-package-json` and `local-bundle` paths in the launcher's `app.xml`, the launcher loads the package metadata and bundle directly from the local filesystem, skipping all network calls for package resolution.

This is useful for:

- **Rapid iteration** — test packaging and launch behavior without a publish-install cycle
- **Offline development** — work without network access
- **CI integration** — validate installer behavior in build pipelines

Path placeholders are supported: `{{ user.home }}`, `{{ executable.dir }}`, and others. Both GUI and command modes work with local bundles.

Reference: client4jgo PR #68

Java Remote Debugging (JDWP)

The native launcher can now start the JVM with JDWP (Java Debug Wire Protocol) debugging enabled, allowing IDEs like IntelliJ IDEA, Eclipse, and VS Code to attach debuggers to a running jDeploy application.

Two new attributes in `app.xml` control this:

- `debug-port` — the JDWP listen port (e.g., `5005`)
- `debug-suspend` — whether to suspend the JVM until a debugger attaches (default: `true`)

Environment variables `JDEPLOY_DEBUG_PORT` and `JDEPLOY_DEBUG_SUSPEND` override the `app.xml` values, making it easy to enable debugging without modifying configuration files.

Reference: client4jgo PR #69

Configurable Update Trigger

The `updater` command implementation previously used a hard-coded `"update"` trigger argument. Developers can now customize this via the `updateTrigger` field in the command spec.

```
{
  "jdeploy": {
    "commands": {
      "myapp-cli": {
        "description": "My App CLI",
        "implements": ["updater"],
        "updateTrigger": "upgrade"
      }
    }
  }
}
```

With this configuration, `myapp-cli upgrade` triggers the self-update instead of `myapp-cli update`. The default remains `"update"` for backward compatibility.

Reference: PR #408

NPX Launch Mode Detection

Applications launched via npm/npx now receive `jdeploy.mode=npx` as a system property, allowing runtime detection of the launch context. Previously, only native launcher modes (`gui`, `command`) were distinguished.

```
String mode = System.getProperty("jdeploy.mode", "gui");
if ("npx".equals(mode)) {
    // Running via npx -- may not have native launcher features
    runNpxMode(args);
}
```

Reference: PR #403

macOS SMAppService Integration

On macOS 13 (Ventura) and later, services configured with `service_controller` can now use Apple's `SMAppService` API for Launch Agent management instead of manually placing plist files. During the build, jDeploy generates embedded LaunchAgent plist files inside the macOS app bundle for commands that have fully static arguments.

Developers can control this behavior per command:

```
{
  "jdeploy": {
    "commands": {
      "myapp-server": {
        "description": "Background server",
        "implements": ["service_controller"],
        "embedPlist": true
      }
    }
  }
}
```

This provides a more native service management experience on modern macOS, where Login Items appear in System Settings and users have explicit visibility and control over background services.

Reference: PR #395, client4jgo PR #61

Bug Fixes

Certificate Pinning Fixed for Platform-Specific Bundles

When platform-specific bundles had their JAR files filtered by `processJarsWithIgnoreService()` after initial package signing, the signatures became invalid, breaking certificate pinning verification on the client side.

Problem: Platform-specific tarballs and default bundles were signed before JAR filtering. After filtering modified the JARs, the signatures no longer matched, causing client-side verification failures.

Solution: The signing service is now passed through to both `PlatformBundleGenerator` and `DefaultBundleService`, ensuring tarballs are re-signed after JAR filtering. Both GitHub and NPM publishing paths are covered.

Reference: PR #400

Linux Shell Profile Chain Preserved

Multiple fixes address a class of issues where jDeploy's PATH management could break the Linux shell configuration chain.

Problem: Creating `~/.bash_profile` on Linux caused bash to stop reading `~/.profile`, which typically sources `~/.bashrc` and sets up the user's entire environment. This effectively broke aliases, environment variables, and other customizations on Ubuntu and other Linux distributions.

Solution: jDeploy now uses platform-specific behavior for shell profile management:

- **Linux**: PATH entries are written to `~/.bashrc`, `~/.zshrc`, and `~/.profile`. `~/.bash_profile` is never created or modified. `~/.profile` provides coverage for login shells, SSH sessions, cron jobs, and other non-interactive environments.
- **macOS**: PATH entries are written to `~/.bash_profile`, `~/.bashrc`, and `~/.zshrc` (macOS Terminal.app uses login shells where `~/.bash_profile` is the correct location).

An opt-out mechanism is available: adding a `NO_AUTO_PATH_MARKER` comment to `~/.profile` prevents jDeploy from modifying it. Existing `~/.profile` content is always preserved.

Reference: PR #407, PR #422, PR #424

PATH Updates in GUI-Launched Installers

Problem: When the installer was launched from a GUI application (such as IntelliJ IDEA or a file manager), the `SHELL` environment variable was not set. PATH entries were not written to shell config files because the installer relied on `SHELL` to determine which files to update.

Solution: The installer now writes to both bash and zsh config files regardless of the `SHELL` environment variable.

Reference: PR #410

MCP Server Compatibility

Problem: The "Downloading java runtime environment" progress message was written to stdout, which corrupted the JSON-RPC stream when running as an MCP server.

Solution: Changed to stderr so the message no longer interferes with MCP protocol traffic.

Reference: PR #402

Template Substitution with Missing GitHub Repository

Problem: When generating a project without a `githubRepository` parameter (e.g., from the MCP server), template tokens like `{{ githubRepository }}` were replaced with the literal string `"null"`.

Solution: Missing values now substitute as empty strings instead of `"null"`.

Reference: PR #401

NPM Publish from GUI

Problem: The npm authentication token was lost during a validation step, preventing NPM publishing from working when initiated from the desktop GUI.

Solution: The token is now preserved through the validation flow.

Reference: PR #404

Publish Settings Not Saving

Problem: Changes made in the Publish Settings panel of the project editor were not persisted.

Solution: Fixed the save handler to correctly persist publish settings, including the GitHub publish target in `publishTargets`.

Current Working Directory in Command Mode

Problem: CLI commands launched via the native launcher had their working directory changed to the user's home directory, breaking tools that operate on relative paths.

Solution: The launcher now preserves the current working directory in command mode. GUI mode continues to use the home directory as before.

Reference: client4jgo PR #64

GitHub Actions Workflow Triggers

Problem: The generated GitHub Actions workflow could trigger on unintended branches.

Solution: Workflow triggers are now restricted to snapshot branches and `v`-prefixed tags, with a concurrency group to prevent duplicate runs.

Migration Guide

Upgrading from 6.0

jDeploy 6.1 is backward compatible with 6.0. Existing packages continue to work with no changes required.

To adopt the new features:

1. **Add an uninstaller**: Add `"implements": ["uninstaller"]` to a CLI command to give users a clean uninstall path
2. **Customize update triggers**: Set `"updateTrigger"` on commands that implement `updater` if you prefer a different trigger word
3. **Enable local development**: Configure `local-package-json` and `local-bundle` in your launcher's `app.xml` to test without publishing
4. **Enable remote debugging**: Set `JDEPLOY_DEBUG_PORT=5005` as an environment variable or configure `debug-port` in `app.xml`

All new features are opt-in and can be adopted incrementally.

Resources

- **Official Website**: https://www.jdeploy.com/
- **Download**: https://github.com/shannah/jdeploy-desktop-gui/releases/latest
- **Documentation**: https://www.jdeploy.com/docs/manual/
- **GitHub Repository**: https://github.com/shannah/jdeploy
- **Support**: GitHub Issues and Discussions