jDeploy 5.3 Release Notes

Table of Contents

Overview

jDeploy 5.3 introduces powerful new features for runtime configuration and file handling. This release adds dynamic placeholder expressions for flexible runtime argument configuration and directory association support for project-based workflows.

What's New in 5.3

Dynamic Placeholder Expressions

Applications can now use enhanced placeholder expressions in runtime arguments, enabling dynamic path construction, environment variable access, and conditional logic without external configuration files.

New Placeholder Features

Environment Variable Access

Access system environment variables in configuration:

```
{
  "jdeploy": {
    "configFile": "{{ env.APPDATA }}/MyApp/config.json"
  }
}
```

Also works in `args`:

```
{
  "jdeploy": {
    "args": [
      "-Duser.name={{ env.USERNAME }}"
    ]
  }
}
```

Coalesce Operator

Provide fallback values when environment variables are not set:

```
{
  "jdeploy": {
    "args": [
      "-Dconfig.dir={{ env.XDG_CONFIG_HOME ?? path(user.home, '.config') }}"
    ]
  }
}
```

String Concatenation

Combine literals and variables to build custom values:

```
{
  "jdeploy": {
    "args": [
      "-Dapp.title={{ 'MyApp - User: ' + env.USERNAME }}"
    ]
  }
}
```

Cross-Platform Path Function

Construct platform-native paths automatically:

```
{
```

```
  "jdeploy": {
    "args": [
      "-Ddata.dir={{ path(user.home, 'Documents', 'MyApp') }}"
    ]
  }
}
```

Produces: - Unix: `/Users/john/Documents/MyApp` - Windows: `C:\Users\john\Documents\MyApp`

Path Normalization

Normalize paths with mixed separators:

```
{
  "jdeploy": {
    "args": [
      "-Dpath={{ normalize('C:/mixed\\slashes/path') }}"
    ]
  }
}
```

Conditional Blocks

Include arguments conditionally based on environment:

```
{
  "jdeploy": {
    "args": [
      "{{ if env.DEBUG }}--verbose{{ end }}",
      "{{ if env.MODE }}{{ env.MODE }}{{ else }}production{{ end }}"
    ]
  }
}
```

Available Placeholders

Placeholder	Description	Example (macOS)	Example (Windows)
`{{ user.home }}`	User's home directory	`/Users/john`	`C:\Users\john`
`{{ executable.path }}`	Full path to launcher executable	`/Applications/MyApp.app/Contents/MacOS/MyApp`	`C:\Program Files\MyApp\MyApp.exe`
`{{ executable.dir }}`	Directory containing the executable	`/Applications/MyApp.app/Contents/MacOS`	`C:\Program Files\MyApp`
`{{ app.path }}`	Path to application bundle (macOS) or executable (other platforms)	`/Applications/MyApp.app`	`C:\Program Files\MyApp\MyApp.exe`

Placeholder	Description	Example (macOS)	Example (Windows)
`{{ app.dir }}`	Directory containing the app bundle (macOS) or executable (other platforms)	`/Applications`	`C:\Program Files\MyApp`
`{{ env.VARNAME }}`	Environment variable access	`{{ env.HOME }}`	`{{ env.USERPROFILE }}`
`{{ pathsep }}`	Platform-specific path separator	`/`	`\`

Expression Syntax

Environment Variables

```
{{ env.VARNAME }}              # Access environment variable
{{ env("VARNAME") }}            # Function syntax
{{ env("VARNAME", "default") }} # With default value
```

Coalesce Operator (`??`)

```
{{ env.APPDATA ?? user.home }}
{{ env.VAR1 ?? env.VAR2 ?? "fallback" }}
```

String Concatenation (`+`)

```
{{ user.home + "/Documents" }}
{{ "prefix-" + env.USERNAME + "-suffix" }}
```

Path Function

```
{{ path(user.home, "Documents", "MyApp") }}
{{ path(env.APPDATA ?? user.home, "MyApp", "config.json") }}
```

Normalize Function

```
{{ normalize("C:/path/with\\mixed/slashes") }}
{{ normalize(user.home + "/docs") }}
```

Conditional Blocks

```
{{ if env.DEBUG }}debug-mode{{ end }}
{{ if env.DEBUG }}debug-mode{{ else }}release-mode{{ end }}
{{ if user.home }}exists{{ else }}missing{{ end }}
```

Use Cases

Platform-Specific Configuration Files

Use placeholders in `configFile` properties to specify different configuration files per platform:

```
{
  "jdeploy": {
    "configFileMac": "{{ path(user.home, 'Library', 'Application Support', 'MyApp',
'config.json') }}",
    "configFileWin": "{{ path(env.APPDATA, 'MyApp', 'config.json') }}",
    "configFileLinux": "{{ path(env.XDG_CONFIG_HOME ?? path(user.home, '.config'),
'MyApp', 'config.json') }}"
  }
}
```

Or use a single cross-platform config file with fallbacks:

```
{
  "jdeploy": {
    "configFile": "{{ path(env.APPDATA ?? path(user.home, '.config'), 'MyApp',
'config.json') }}"
  }
}
```

XDG Base Directory Support

```
{
  "jdeploy": {
    "args": [
      "-Dconfig.dir={{ env.XDG_CONFIG_HOME ?? path(user.home, '.config') }}"
    ]
  }
}
```

Multi-Level Fallbacks

```
{
  "jdeploy": {
    "args": [
      "-Dcache.dir={{ path(env.TMPDIR ?? env.TEMP ?? '/tmp', 'myapp-cache') }}"
    ]
  }
}
```

Executable-Relative Resources

```
{
```

```
  "jdeploy": {
    "args": [
      "-Dresource.path={{ path(executable.dir, '..', 'Resources', 'config.xml') }}"
    ]
  }
}
```

Debug Mode Configuration

```
{
  "jdeploy": {
    "args": [
      "{{ if env.DEBUG }}--verbose{{ end }}",
      "{{ if env.DEBUG }}--log-level=trace{{ else }}--log-level=info{{ end }}"
    ]
  }
}
```

Backward Compatibility

All existing placeholders continue to work unchanged. Applications using simple placeholders like `{{ user.home }}` require no modifications.

Directory Association Support

Applications can now register as handlers for directories and folders, enabling project-based workflows and workspace management.

Configuration

Add directory associations to `package.json`:

```
{
  "jdeploy": {
    "documentTypes": [
      {
        "type": "directory",
        "role": "Editor",
        "description": "Open folder as project"
      }
    ]
  }
}
```

Configuration Fields

type

Set to `"directory"` to indicate a directory association

role (optional)

Either `"Editor"` or `"Viewer"`. Default: `"Viewer"`

description (optional)

Human-readable text shown in context menus

icon (optional)

Custom icon path for directory associations

Platform-Specific Behavior

Windows

Creates context menu entries for:

- Right-clicking on folders
- Right-clicking in empty folder space ("Open folder with...")

Registry keys:

- `HKEY_CURRENT_USER\Software\Classes\Directory\shell{ProgId}`
- `HKEY_CURRENT_USER\Software\Classes\Directory\Background\shell{ProgId}`

macOS

Registers application to handle folders via `CFBundleDocumentTypes`:

```
<key>LSItemContentTypes</key>
<array>
    <string>public.folder</string>
</array>
```

Enables: - Drag-and-drop folders onto application icon - "Open With" context menu for folders

Linux

Adds `inode/directory` to `.desktop` file `MimeType` field:

```
MimeType=inode/directory;
```

Provides "Open With" context menu integration in GNOME and KDE.

GUI Editor Support

The jDeploy project editor GUI now includes a directory association panel in the file associations section with:

- Enable/disable checkbox
- Role selection (Editor/Viewer)
- Description text field
- Custom icon browser
- Validation for required fields

Use Cases

- **IDEs and Code Editors**: Open project directories
- **Build Tools**: Execute builds on project folders
- **File Managers**: Browse directory structures
- **Archive Tools**: Compress/extract folder contents
- **Project Management**: Open workspace directories

Example Configuration

Complete example with both file and directory associations:

```
{
  "jdeploy": {
    "title": "My IDE",
    "documentTypes": [
      {
        "extension": "java",
        "mimetype": "text/x-java-source",
        "editor": true
      },
      {
        "type": "directory",
        "role": "Editor",
        "description": "Open as Project",
        "icon": "resources/folder-icon.png"
      }
    ]
  }
}
```

Bug Fixes

Improved JRE Selection for Custom JavaFX Versions

Fixed a minor issue affecting applications that explicitly specify a custom `javafxVersion` (e.g., to use early access versions of JavaFX).

Note: This fix only affects applications using the `javafxVersion` configuration option. The vast majority of applications that use the JavaFX bundled with the JRE are unaffected.

What Changed

When `javafxVersion` is specified in package.json, the launcher now ensures that only JREs without bundled JavaFX are selected. This prevents potential conflicts between the Maven-downloaded JavaFX version and any JavaFX libraries bundled with the JRE.

```
{
  "jdeploy": {
    "javaVersion": "11",
    "javafxVersion": "25-ea+29"
  }
}
```

The launcher will now skip JREs with bundled JavaFX (e.g., "11fx") and only use JREs without JavaFX (e.g., "11") when this configuration is present.

Technical Details

Placeholder Expression Parser

- Zero external dependencies (uses only Go standard library)
- Recursive descent parser with operator precedence
- Context caching with `sync.Once` for performance
- Comprehensive error handling with descriptive messages
- ~550 lines of Go code with 102 passing tests

Directory Association Implementation

Model Changes

- `DocumentTypeAssociation` extended to support both files and directories
- Single `isDirectory` flag differentiates association types
- Unified model for all document type handling

Platform Integration

- Windows: Registry-based context menu integration
- macOS: Info.plist `CFBundleDocumentTypes` with `LSItemContentTypes`
- Linux: `.desktop` file `MimeType` field

Uninstallation

Directory associations are properly removed during uninstallation on all platforms.

JavaFX JRE Selection

Updated Logic

- `findMatchingJREPath` now accepts `requireNoJavaFX` parameter
- When `javafxVersion` is specified, `requireNoJavaFX` is set to `true`
- JRE selection skips "fx" suffix variants when `requireNoJavaFX` is true

Backward Compatibility

Applications without `javafxVersion` specified experience no change in behavior.

Migration Guide

Upgrading from 5.2

jDeploy 5.3 is backward compatible with 5.2. No changes are required for existing applications.

Using Dynamic Placeholders

1. Identify configuration values that vary by environment or platform
2. Replace hardcoded paths with placeholder expressions
3. Test on each target platform
4. Consider using `if/else/end` blocks for platform-specific logic

Before:

```
{
  "jdeploy": {
    "args": [
      "-Dconfig.path=/Users/john/.config/myapp/config.json"
    ]
  }
}
```

After:

```
{
  "jdeploy": {
    "args": [
      "-Dconfig.path={{ path(env.XDG_CONFIG_HOME ?? path(user.home, '.config'),
'myapp', 'config.json') }}"
    ]
  }
}
```

Adding Directory Associations

1. Determine if your application should handle directories
2. Choose appropriate role (Editor for modification, Viewer for read-only)
3. Add directory association to package.json
4. Test on each platform

```
{
  "jdeploy": {
    "documentTypes": [
      {
        "type": "directory",
        "role": "Editor",
        "description": "Open folder as workspace"
      }
    ]
  }
}
```

Best Practices

Placeholder Expressions

- Use `path()` function for cross-platform path construction
- Provide fallbacks with `??` operator for optional environment variables
- Use `if/else/end` blocks for environment-specific configuration
- Test expressions on all target platforms
- Keep expressions readable - extract complex logic to multiple arguments

Directory Associations

- Use "Editor" role only if application modifies folder contents
- Provide clear, concise descriptions for context menus
- Test with various directory types (empty, large, network)
- Document directory handling behavior for users
- Consider custom icons to distinguish from file associations

Known Issues

- Placeholder expressions are evaluated at launch time only (not runtime)

- Only one directory association per application is supported
- macOS sandboxed apps require appropriate entitlements for folder access
- Linux desktop environment support varies (tested on GNOME and KDE)

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