Plug-in Pitfalls

Things to avoid when customizing DITA-OT

Roger Fienhold Sheen

@infotexture

Caveat № 1:
This is not a tutorial

— but it links to several

Agenda

- What not to do
- What to do instead
- Where to go for help *(customization resources)*
- Please share!

What not to do

🚫 Don’t modify core code

- Set permissions on plug-in directories to “read-only”
- Prevents accidental modifications to defaults as you develop

🚫 Don’t copy default files

- Only copy attribute sets and templates you need to override
- Less risk of impact from new features or fixes in base code
- Makes your code more stable and easier to upgrade

🚫 Don't use PDF2 `Customization`

- Files in `org.dita.pdf2/Customization` override defaults
- You *could* use it to test changes without modifying defaults
- **Better:** Specify an external `customization.dir` to ensure overrides are not overwritten when you upgrade DITA-OT
- **Best:** Use a properly-constructed DITA-OT plug-in

🚫 Don’t change template order

- If you modify multiple default templates in the same file
- Preserve the original order to facilitate comparisons

🚫 Don’t mix custom code with defaults

- If you make changes to existing templates or modes
- Leave them in files named after the originals
- Put new custom templates and modes in separate files

What to do instead

✅ Version all the things

✅ Document your changes

- Every custom plug-in should include a ReadMe file
- Use code comments to explain purpose of changes
- Reputable consultants document their work
- Undocumented plug-ins are technical debt

✅ Use `custom.xsl` files

- If you only need to change a few attributes or templates
- Store your overrides in `custom.xsl` files
- or a simple folder hierarchy within your custom plug-in

✅ Mimic default hierarchies

- Use a folder structure that mimics the default hierarchy
- Facilitates comparisons with the defaults in base plug-ins
- Makes it easier to migrate changes to new toolkit versions
- See PDF plug-in structure for info on default files

✅ Invest in a three-way diff tool

✅ Upgrade regularly

- Do not wait through several releases before upgrading
- Each major release you skip makes it harder to migrate
- Porting changes from one release to the next is simpler
- Migration instructions provided for each release

✅ Validate `plugin.xml`

- DITA-OT includes a RELAX NG schema file that can be used to validate the `plugin.xml` files that define the capabilities of each plug-in
- Include an `xml-model` processing instruction at the beginning of the plugin.xml file, immediately after the XML prolog:

```
<?xml-model href="dita-ot/plugin.rnc" type="application/relax-ng-compact-syntax"?>
```

- If your authoring environment does not apply this schema automatically, point your editor to `dita-ot-dir/resources/plugin.rnc`

✅ Use a custom namespace

- For modified templates, modes, attribute sets, functions, and variables
- Helps clarify which portions of code are customized
- Isolates your changes if items with same name are added to base plug-ins

```
<xsl:template name="corp:searchbar"/>
```

Instead of:

```
<xsl:template name="searchbar"/>
```

✅ Use the plug-in directory property

- In Ant scripts, refer to files in other plug-ins with `dita.plugin.plugin-id.dir`

```
<property name="base" location="$
{dita.plugin.example.dir}/custom.xsl"/>
```

Instead of:

```
<property name="base" location="../example/custom.xsl"/>
```

- Finds plug-ins in custom folders that don't match the plug-in ID

Use the `plugin` URI scheme

- In XSLT, use the **`plugin`** URI scheme in **`xsl:import`** and **`xsl:include`** to reference files in other plug-ins.

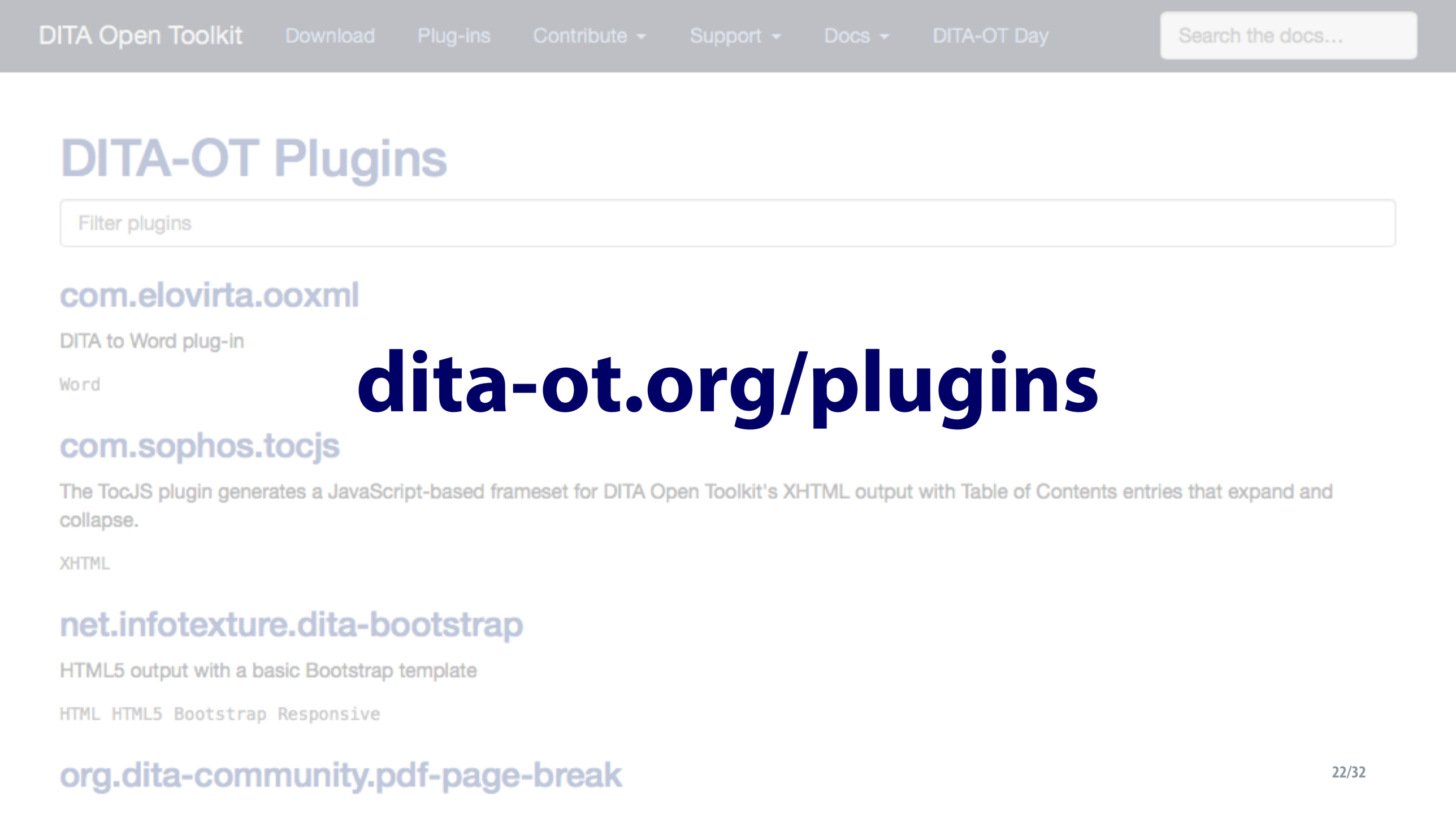
```
<xsl:import href="plugin:org.dita.base:xsl/common/output-
 message.xsl"/>
```

Instead of:

```
<xsl:import href="../../org.dita.base/xsl/common/output-
 message.xsl"/>
```

- Resolves to the correct directory even when plug-ins move

Where to go for help *(customization resources)*



dita-ot.org/plugins

✨ DITA-OT Plug-in Registry

The plug-in registry at dita-ot.org/plugins makes it easier to discover & install plug-ins by name via the `dita` command:

```
dita --install net.infotexture.dita-bootstrap
```

The registry is a Git repository at github.com/dita-ot/registry.

New plug-ins or new versions can be added by sending a pull request that includes a new plug-in entry in JSON format.

Read The Docs

See the DITA-OT docs sections on customization:

- Bundling CSS in a custom HTML plug-in
- Embedding web fonts in HTML output
- Inserting JavaScript in generated HTML
- Customizing PDF output

Jarno Elovirta's PDF Plugin Generator

The online tool at dita-generator.elovirta.com allows you to adjust a variety of PDF settings and automatically creates a new custom PDF plug-in:

- Define the target environment *(current and 2 previous versions of DITA-OT)*
- Select the formatting engine *(FOP, Antenna House Formatter, or RenderX XEP)*
- Specify page size, columns, and margins
- Select options for headers and footers
- Specify layout options for chapters

Leigh White's *"DITA for Print"*

A DITA Open Toolkit Workbook walks readers through developing a PDF customization from scratch.

DITA for Print is for anyone who wants to learn how to create PDFs using the DITA Open Toolkit without learning everything there is to know about XSL-FO, XSLT, or XPath, or even about the DITA Open Toolkit itself.

Please share!

✅ Share your code

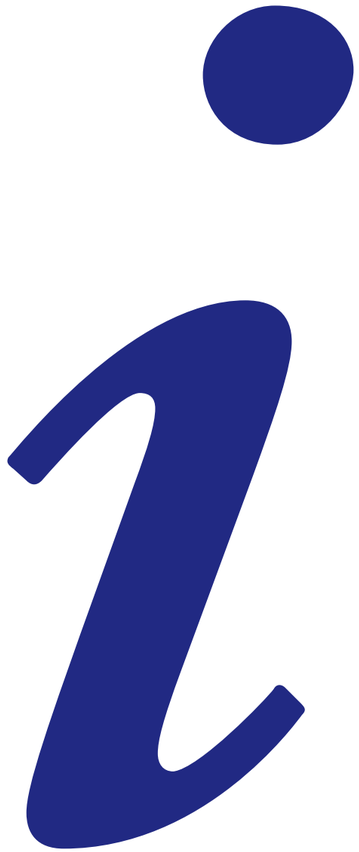
— Tag releases on GitHub for stable URLs

✅ Publish your plug-ins

— Submit pull requests to the plug-in registry

Any other pet peeves?

infotexture.net/dita-europe-2019



infotexture