ADAMS

Advanced **D**ata mining **A**nd **M**achine learning **S**ystem

Module: adams-meta

Peter Reutemann

January 10, 2024

Contents

List of Figures

Chapter 1

Dynamic use of templates

The templating mechanism described in the "core-module" manual, shows how to speed up the inception of new flows. But the templates can also be used in a dynamic way at runtime using the following actors:

- *TemplateStandalone* – for templates that generate standalones
- *TemplateSource* – for templates that generate sources
- *TemplateTransformer* – for templates that generate transforming sub-flows
- *TemplateSink* – for templates that generate sinks

The sub-flow generation is done in a lazy way, i.e., only when the aforementioned template actor is executed, the template is generated. The sub-flow is used till either the end of the flow execution or if a variable changes that is attached to the template itself. In the latter case, the sub-flow gets re-generated the next time the template actor gets executed. This dynamic sub-flow generation in conjunction with variable use, allows to adapt and change the flow at runtime. The example *adams-core-template.flow* demonstrates this.

Chapter 2

Copying callable actors

Callable actors can not only be used as synchronization points in the flow. It is also possible to *copy* them, using them as templates. If you don't want to use external flows, but still need to use the same sub-flow multiple times and avoid the bottle next of synchronous execution, then you can use one of the following actors to create a copy of the callable at the very same location:

- *CopyCallableStandalone* – copies a callable standalone
- *CopyCallableSource* – copies a callable source
- *CopyCallableTransformer* – copies a callable transformer
- *CopyCallableSink* – copies a callable sink

Chapter 3

Including external actors

Similar to the external actors of the *adams-core* module, the following actors allow the use of external flow snippets. However, these flows simply replace themselves with the content of the external flow and cannot be changed dynamically with variables. Flexibility has been traded here for performance.

- *IncludeExternalStandalone* – includes an external standalone
- *IncludeExternalSource* – includes an external source
- *IncludeExternalTransformer* – includes an external transformer
- *IncludeExternalSink* – includes an external sink

Chapter 4

Auto-generated actors

In some cases, flows get generated on the fly with actors being added. If these actors should need to be removed again, e.g., when restarting the flow, then it is possible to use the following wrappers for *auto-generated* actors:

- *AutogeneratedStandalone* – encapsulates auto-generated standalones
- *AutogeneratedSource* – encapsulates auto-generated actors that act as source
- *AutogeneratedTransformer* – encapsulates auto-generated actors that form a transformer
- *AutogeneratedSink* – encapsulates auto-generated actors that behave as sink

NB: With the *RemoveAutogeneratedActors* processor it is possible to remove all these actors again.

Chapter 5

Inactive actors

In some cases, flows get generated on the fly with actors being added. Original actors may get replaced, but you can keep them as *inactive* ones in the flow (no setup or execution occurs), by wrapping them in one of these meta-actors:

- *InactiveStandalone* – encapsulates inactive standalones
- *InactiveSource* – encapsulates inactive actors that act as source
- *InactiveTransformer* – encapsulates inactive actors that form a transformer
- *InactiveSink* – encapsulates inactive actors that behave as sink

NB: With the *ReactivateActors* processor it is possible to activate all these actors again, replacing the wrappers.

Chapter 6

Miscellaneous

Below are other actors that haven't been covered by the previous chapters:

- *NewFlow* – generates a new actor/flow using the specified template.
- *CurrentFlow* – just outputs the current flow.
- *ExecuteActor* – executes the actor passing through.
- *FlowFileReader* – reads a flow file from disk and forwards it.
- *FlowFileWriter* – writes the incoming flow to disk.
- *FlowDisplay* – displays an actor or flow.
- *ProcessActor* – applies an *ActorProcessor* to the incoming flow (eg listing variables, updating variable names).
- *SpecifiedActor* – outputs the actor identified via its path in the flow, e.g., for storing an actor setup in a file.

Bibliography

[1] *ADAMS* – Advanced Data mining and Machine learning System
`http://adams.cms.waikato.ac.nz/`