Papyrus 2.0 Migration Guide

Christian W. Damus

22 January, 2016

Contents

Papyrus 2.0 API Migration Guide

This document provides a guide to migrating applications that extend Papyrus to the new 2.0 version APIs. In the 2.0 (Neon) release, several refactorings were implemented to fix problems of invalid bundle dependencies. These dependencies are of two kinds:

- dependency on Eclipse UI and/or SWT in bundle that should provide only headless API
- dependencies that violate the Papyrus layer architecture. For example, bundles in the "Infra" layer using UML or GEF APIs, bundles in the core NatTable component using diagram APIs, etc.

In most cases, these refactorings are simply moving types from one bundle to another bundle that is better suited to hosting them. As such, migration generally requires one or two steps:

1. Updating imports to change package names: type names are usually not changed, but the package namespaces do change to reflect the new containing bundle name
2. Adding new bundle dependencies to bring the new packages into the classpath. This step will sometimes not be required when the bundle to which a type was moved is already on the dependent bundle's classpath

Note	Owing to the Papyrus project's policy prohibiting `Require-Bundle` re-exports, it will sometimes be necessary to change your bundles' dependencies in this migration task even when none of the APIs that it uses are moved, because of refactorings in the transitive dependencies of APIs that your bundles do use.

Migration details are presented by Papyrus component/layer and theme in the following sections.

Infra Layer

UI Dependencies

APIs Moved out of the infra.tools Bundle

Several UI-dependency classes are moved from the `org.eclipse.papyrus.infra.tools` bundle to `org.eclipse.papyrus.infra.ui`. These simply require package renames in imports and adding the `infra.ui` bundle dependency.

Some packages are moved in their entirety (`*` replaces the prefix `org.eclipse.papyrus`):

Old Package	New Package
*.infra.tools.converter	*.infra.ui.converter
*.infra.tools.dnd	*.infra.ui.dnd
*.infra.tools.messages[1]	*.infra.ui.messages
*.infra.tools.preferences	*.infra.ui.preferences
*.infra.tools.preferences.ui.dialog	*.infra.ui.preferences.dialog

And in the `org.eclipse.papyrus.infra.tools.util` package, several individual types are moved to `org.eclipse.papyrus.infra.ui.util`:

- `AbstractCreateMenuFromCommandCategory`
- `EclipseCommandUtils`
- `EditorHelper`
- `ICallableWithProgress`
- `LocalMemento`
- `SelectionHelper`
- `UIUtil`

[1]which should not have been API in any case

- `WorkbenchPartHelper`

The UIUtil API for asynchronous code execution

The `UIUtil` class presents some interesting use cases. It is often used by clients to provide a simple means of deferring execution of some block of code or ensuring that some code is executed on the UI thread, even in an otherwise headless API (usually because it is known or assumed that common use cases for that API are initiated by an application UI). To that end, a new API is introduced in the `org.eclipse.papyrus.infra.tools` bundle that provides a headless-compatible access to this UI execution capability:

- `org.eclipse.papyrus.infra.tools.util.CoreExecutors`. This new class is intended to provide access to a variety of useful `Executor` implementations for asynchronous execution of code. The `getUIExecutorService` method provides one such executor for posting code to the UI thread. If there is no UI thread because the application is a headless Eclipse instance, then this executor simply uses a daemon thread as per the standard Java platform single-thread executor service
- `org.eclipse.papyrus.infra.tools.util.IExecutorService`. This extension of the Java Platform `ExecutorService` interface provides methods for synchronous execution of `Runnable`s and `Callable`s on the UI thread. The UI executor service provided by the `CoreExecutors` class implements that protocol using `Display.syncExec` when the UI is available

APIs Moved out of the infra.core Bundle

Several UI-dependent APIs are moved out of the `org.eclipse.papyrus.infra.core` bundle into `org.eclipse.papyrus.infra.ui`, most notably the `IMultiDiagramEditor` interface and its related service APIs.

Some packages are moved in their entirety:

Old Package	New Package
*.infra.core.contentoutline	*.infra.ui.contentoutline
*.infra.core.editor.reload	*.infra.ui.editor.reload
*.infra.core.editorsfactory	*.infra.ui.editorsfactory.anytype
*.infra.core.editorsfactory.anytype	*.infra.ui.editorsfactory
*.infra.core.extension.commands	*.infra.ui.extension.commands
*.infra.core.extension.diagrameditor	*.infra.ui.extension.diagrameditor
*.infra.core.lifecycleevents	*.infra.ui.lifecycleevents
*.infra.core.multidiagram.actionbarcontributor	*.infra.ui.multidiagram.actionbarcontributor

In the `org.eclipse.papyrus.infra.core.editor` package, several types were moved to a new `org.eclipse.papyrus.infra.ui.editor` package:

- `ContentProviderServiceFactory`
- `CoreMultiDiagramEditor`
- `DiSashModelManagerServiceFactory`
- `DiSashModelMngrServiceFactory`

- `IMultiDiagramEditor`
- `IPapyrusPageInput`
- `IReloadableEditor`
- `MultiDiagramEditorSelectionContext`
- `MultiDiagramPropertySheetPage`
- `PageIconRegistryServiceFactory`
- `PageMngrServiceFactory`
- `PapyrusPageInput`

And in the `org.eclipse.papyrus.infra.core.util` package, several individual types are moved to `org.eclipse.papyrus.infra.ui.util`:

- `DisplayUtils`
- `EditorUtils`
- `PapyrusImageUtils` — note that the icon resources are also moved
- `ServiceUtilsForActionHandlers`
- `ServiceUtilsForWorkbenchPage`

The service utilities API

The `AbstractServiceUtils` class and its subclass present some interesting use cases for the API migration. The `AbstractServiceUtils` class in the 1.x release provided the following methods that were accessible through all of the concrete utility subclasses:

- `getIPageMngr(T): IPageMngr`
- `getIPageManager(T): IPageManager`
- `getILifeCycleEventsProvider(T): ILifeCycleEventsProvider`
- `getISashWindowsContainer(T): ISashWindowsContainer`
- `getNestedActiveEditorPart(T): IEditorPart`

These methods were all convenient shortcuts and are simply deleted in the 2.0 API. There is no sensible way to move them because the types that exposed these methods must remain accessible in a headless context, but the APIs that they return are strictly UI-dependent. Clients must change to access these services using the standard `getService(Class<?>, T)` method. In the case of the nested active editor part, the replacement is

```
ServiceUtilsForXyz.getInstance().getService(
    ISashWindowsContainer.class, xyz).getActiveEditor();
```

Also, the `ServiceUtilsForActionHandlers` class is used by many clients to access the services in the currently active Papyrus editor, where the calling context doesn't actually know what that editor is. This worked by looking for the Platform UI's active editor and getting its service registry. That obviously doesn't work in the context of the now headless `org.eclipse.papyrus.infra.core` bundle, so a new OSGi service is defined that allows the core `AbstractServiceUtils` API to access this current-editor-context service registry. Clients that depend on this capability can now, if they no longer have access to the `ServiceUtilsForActionHandlers` API (which is deprecated anyways), get it thus:

```
ServiceUtils.getInstance().getServiceRegistry(null); // to get the registry
ServiceUtils.getInstance().getModelSet(null);        // To get the ModelSet
ServiceUtils.getInstance().getService(IPageManager.class, null); // Others
```

Bundle Re-exports Removed from the infra.core Bundle

Although it was in contravention of Papyrus project policy, the 1.x version of the `org.eclipse.papyrus.infra.core` bundle re-exported its dependency on the `org.eclipse.papyrus.infra.core.sasheditor.di` bundle for its DI/sash-windows model and other APIs. This is an overtly UI-dependent bundle, so it is no longer used by the `infra.core` bundle in Papyrus 2.0 and therefore is not re-exported. Clients that relied on this re-export will have to add it explicitly. Moreover, the EMF model that was provided by that bundle is moved to another new bundle, as discussed below.

APIs Moved out of the infra.emf Bundle

A few UI-dependent packages are moved from in their entirety from the `org.eclipse.papyrus.infra.emf` bundle to a new `org.eclipse.papyrus.infra.ui.emf` bundle (`*` replaces the prefix `org.eclipse.papyrus`):

Old Package	New Package
*.infra.emf.databinding	*.infra.ui.emf.databinding
*.infra.emf.dialog	*.infra.ui.emf.dialog
*.infra.emf.providers	*.infra.ui.emf.providers
*.infra.emf.providers.strategy	*.infra.ui.emf.providers.strategy

One class is moved from the `org.eclipse.papyrus.infra.emf.adapters` package to `org.eclipse.papyrus.infra.`

- `EObjectAdapterFactory`

One class is moved from `org.eclipse.papyrus.infra.emf.utils` to `org.eclipse.papyrus.infra.emf.utils`:

- `ProviderHelper`

In fact, the `org.eclipse.papyrus.infra.ui.emf.utils.ProviderHelper` class now also has a `getCustomizationManager()` method that should now be used instead of the same method on the `org.eclipse.papyrus.infra.emf.Activator` class to access the EMF Facet customization manager.

One class is moved from the `org.eclipse.papyrus.infra.emf.utils` package to `org.eclipse.papyrus.infra.ui.` in the `org.eclipse.papyrus.infra.ui` bundle:

- `EMFStringValueConverter`

and several others are moved to the `org.eclipse.papyrus.infra.ui.util` package because they have nothing to do with the EMF component, specifically:

- `ServiceUtilsForHandlers`
- `ServiceUtilsForIEvaluationContext`
- `ServiceUtilsForSelection`

Finally, note that, although the `org.eclipse.papyrus.infra.emf` bundle no longer has any UI dependencies in version 2.0, the `EMFHelper::getEObject(Object)` utility API still can unwrap selections from the

Model Explorer and other views based on the EMF Facet tree model nodes that encapsulate the business objects being presented.

APIs Moved out of the infra.core.sasheditor.di Bundle

The `infra.core.sasheditor.di` bundle in the 1.x version performs two distinct functions: defining an EMF model for the editor layout in the `*.di` (and also `*.sash`) resource, and implementing a sash-windows content provider (from the `infra.core.sasheditor` bundle) based on this DI model. Unfortunately, the latter adds a UI dependency to the bundle, so that headless code cannot make use of the DI model API.

The DI model packages are moved into their own bundle, better aligning with the usual EMF code generation pattern. The following packages are now provided by the `org.eclipse.papyrus.infra.sashwindows.di` bundle:

- `org.eclipse.papyrus.infra.core.sashwindows.di`
- `org.eclipse.papyrus.infra.core.sashwindows.di.exception`
- `org.eclipse.papyrus.infra.core.sashwindows.di.impl`
- `org.eclipse.papyrus.infra.core.sashwindows.di.util`

This new bundle is now a dependency of the `infra.core` bundle, which continues to use the DI model API. However, it is not re-exported as the `infra.core.sasheditor.di` dependency was in the 1.x version, so clients of the DI model API that got it “for free” from `infra.core` must now add an explicit dependency on `infra.core.sashwindows.di`.

Similarly, the `IPageManager` API does still need to be accessible by client code in a headless context in order to maintain the integrity of the DI/Sash model references to pages, especially when the notation views that they reference are deleted. This is necessary whether they are presented in an editor or not. A part of this is the particles of edit advice that clean up page-references to deleted objects. Accordingly, the following classes are moved from the `org.eclipse.papyrus.infra.core.sasheditor.di` bundle to `org.eclipse.papyrus.infra.core.sashwindows.di` along with the DI model, itself:

Old Type	New Type
*.sasheditor.contentprovider.IPageManager	*.sashwindows.di.service.IPageManager
*.sasheditor.contentprovider.service.ILocalPageService	*.sashwindows.di.service.ILocalPageService
*.sasheditor.contentprovider.service.AbstractLocalPageService	*.sashwindows.di.service.AbstractLocalPageService

Along with these, the `IPageUtils::getMemoizedCloseAllPagesCommand()` method is moved to the `org.eclipse.papyrus.infra.core.sashwindows.di.util.DiUtils` class. Also, the `org.eclipse.papyrus.infra.core.sasheditor.contentprovider.PageMngr` class, which implemented the deprecated `IPageMngr` interface, is moved to `org.eclipse.papyrus.infra.core.sashwindows.di.servi` as a headless implementation of the `IPageManager` service. It includes support for the legacy `PageList` object in the DI model and is specialized, as the `PageMngr` was previously, by the UI-based `PageManagerImpl` in the `org.eclipse.papyrus.infra.core.sasheditor.di` bundle.

API Moved out of the infra.constraints Bundle

The `org.eclipse.papyrus.infra.constraints.providers.ConstraintTypeContentProvider` class is moved to a new bundle `infra.constraints.ui` as `org.eclipse.papyrus.infra.constraints.ui.provider`

More significantly, the signatures of API methods that accepted `ISelection` or `IStructuredSelection` (which are JFace UI types) now accept more plastic `Object` and `Collection<?>` parameters, respectively:

- `IConstraintEngine::getDisplayUnits(Object selection)` (and hence the same in the `DefaultConstraintEngine` class) no longer requires an `ISelection`
- `Constraint::match(Collection<?> selection)` (and hence the same in `AbstractConstraint` and `CompoundConstraint`) no longer requires an `IStructuredSelection`

Note that the default implementation of the `getDisplayUnits(Object)` API accepts arguments in a variety of shapes, from which it tries to get or create a collection to pass along to its constraints:

- incoming `null`s are coerced to empty collections
- incoming collections are taken as they are
- incoming objects that offer a no-argument collection coercion method such as `asSet()` or `toList()` or similar (which includes the `IStructuredSelection` type) will be converted to a collection via that method
- other objects will be wrapped in a singleton collection

APIs Moved out of the infra.extendedtypes Bundle

An UI-dependent package defining APIs for action providers is moved from the `infra.extendedtypes` bundle to a new `org.eclipse.papyrus.infra.extendedtypes.ui` bundle:

Old Package	New Package
*.infra.extendedtypes.providers	*.infra.extendedtypes.ui.providers

This includes all of:

- the `ExtendedElementTypeActionService` GMF-style service class
- the `IExtendedElementTypeActionProvider` interface implemented by plug-ins contributing actions
- the `extendedElementTypeActionProvider` extension point on which action providers are registered is moved into the new UI bundle's namespace

APIs Moved out of the infra.services.edit Bundle

The `ElementTypeValidator` class is moved from the `org.eclipse.papyrus.infra.services.edit.utils` package to the `org.eclipse.papyrus.infra.services.ui.dialogs` package in a new `org.eclipse.papyrus.infr` bundle.

APIs Moved out of the infra.onefile Bundle

Several UI-dependent packages are moved out of the `infra.onefile` bundle into a new `org.eclipse.papyrus.infra.o` bundle. However, most of these contained only internal APIs and were not exported at all, so clients should not be affected.

One package is partially moved:

Old Package	New Package
*.infra.onefile.providers	*.infra.onefile.ui.providers

The following types in the `onefile.providers` package were moved to the new `onefile.ui.providers` package:

- `PapyrusContentProvider`
- `PapyrusLabelProvider`
- `PapyrusViewerSorter`

The `OneFileModelProvider` class remains in the headless `infra.onefile` bundle and other classes are now internal in the UI bundle because they were not public API:

- `CopyToClipboardAction`
- `OneFileDecorator`
- `PapyrusEditActionProvider`
- `PapyrusModelActionProvider`
- `SubresourceFileActionProvider`

Finally, some APIs are changed on the sub-type level:

- the `IPapyrusElement::getImage()` method is removed. This is simply incompatible with a headless execution environment and is an incorrect placement of the responsibility, anyways, which in Eclipse is served by label providers. Thus, the implementations of this API are now provided exclusively by the `PapyrusLabelProvider` class in the UI bundle
- several API methods are moved from the `OneFileUtils` class into a new class `org.eclipse.papyrus.infra.onef`
  - `getActivePage()`
  - `getEditorID(IEditorInput)`
  - `isOpenInEditor(Object)`
  - `openInEditor(Object, boolean)`

  It is doubtful that any of these should have been used by clients in the first place, as they are all redundant with similar utilities provided now by the `infra.ui` bundle (formerly by the `infra.core` bundle).

GEF 3 Dependencies

To support the introduction of diagrams based on the new GEF 4 API, dependencies on the GEF 3 API need to be isolated as much as possible from the core GMF-based diagram infrastructure in Papyrus. Also, because GEF implies a UI dependency, dependencies on GEF APIs in bundles that should be headless also need to be refactored.

Accordingly, some GEF-related refactorings in the Infra Layer may need to be accounted for by clients.

APIs Moved out of the infra.core.sasheditor Bndle

The `MultiDiagramEditorGefDelegate` class is moved from the `org.eclipse.papyrus.infra.core.sasheditor.ed` package to the `org.eclipse.papyrus.infra.gmfdiag.gef.internal.editor` package in a new `org.eclipse.papyrus.infra.gmfdiag.gef` bundle. Note that this API is now internal.

Also, the `CoreMultiDiagramEditor` class, which itself was moved from the `infra.core` bundle to `infra.ui`, has had its `gefAdaptor` field removed because the GEF dependency in the `infra.ui` bundle would be illegal. Accordingly, this is now implemented as an external adapter via Eclipse Platform's adapter registry: the `CoreMultiDiagramEditorAdapterFactory` class in the `infra.gmfdiag.gef` bundle provides the `ActionRegistry` adapter via the `MultiDiagramEditorGefDelegate`.

UML Dependencies

New APIs to Generalize UML-specific Patterns

Replacing Usage of UmlModel

The `org.eclipse.papyrus.uml.tools.utils.UmlModel` API (often via `UmlUtils` in the same package) is commonly used in the 1.x releases to access the "semantic model", being the model content that it is the user's intent to edit. This naturally assumes UML content to the exclusion of any other (the Papyrus vision being broader than UML). Several new and updated APIs are now available for a more generic access to the model content:

- in the *Language Service*, an `ILanguage` can now be associated with an `IModel` that provides access to its semantic content in the `ModelSet` via a new `<modelBinding>` element in the `org.eclipse.papyrus.infra.core.languages` extension point. The `UmlModel` is thus associated with the UML language. The `ILanguage` interface has a new `getModel(ModelSet) : IModel` API to get the associated language, if there is one. Similarly, the `ILanguageService` has a static convenience method `getLanguageModels(ModelSet)` for obtaining all of the semantic models in a model set that have content
- the `IEMFModel` interface extending `IModel` has a new `getRootElements() : Iterable<? extends EObject>` API providing the root semantic model elements. Thus, for any language models (per above) that implement this interface, they can now provide the actual root semantic model elements that the user is editing. The `UmlModel` implementation of this API provides the top-most UML elements, excluding stereotype applications or other foreign-schema resource contents
- the *Semantic Service* now more accurately provides the root semantic model elements, according to the aforementioned language models, instead of all contents of all resources in the set. Also, the `SemanticService::getSemanticIModels()` API now is implemented, providing the language models as above

Replacing Usage of SemanticUMLContentProvider

Several generic UI components, such as element-chooser dialogs and even the *Model Explorer* view, have a need to present the semantic model content from the `ModelSet` to the user. Thus, it was often necessary to construct a `SemanticUMLContentProvider` as content provider for a `TreeViewer`. The most appropriate `ITreeContentProvider`s for presentation of model content can now be obtained via the semantic `IModel`s discussed in the previous section:

- a new `org.eclipse.papyrus.infra.ui.providers.ISemanticContentProviderFactory` interface creates tree-content providers conforming to the specialized Papyrus protocols required for various use cases: `IStaticContentProvider`, `IHierarchiveContentProvider`, and `IAdaptableContentProvider`. The provider factories are self-composable; a composite factory yields composite content providers
- the `IModel` interface now extends `IAdaptable`, to provide for adapters of `ISemanticContentProviderFactory` type. Thus, for any semantic model (as above) that has a content-provider factory adapter, a suitable tree content provider can be obtained. By composing the factories for all semantic models in the model-set, all potentially heterogeneous semantic content can be presented to the user in a unified view (for example, if there is Ecore content in addition to UML content in the model-set)
- in the 2.0 release, a provider factory adapter is supplied for the `UmlModel` that creates the `SemanticUMLContentProvider` used previously

Replacing Usage UML-specific XML Enablement Expression Definitions

Several menu/toolbar contributions in Papyrus are guarded with enablement expressions that are UML-specific, that should be more generally applicable to Papyrus editors on any kind of semantic model. Consider replacing expression definitions as follows

Old Definition/Property	New Definition/Property
*.uml.diagram.common.IsPapyrusActiveWithUMLModel	*.ui.semanticModelActive

Note that the "old definitions" in the left column of the table above are still defined and usable in the appropriate circumstances.

Extension Identifiers Moved out of the uml.diagram namespace

The `org.eclipse.papyrus.uml.diagram.ui.toolbar` contribution to the Platform toolbars is defined in the `org.eclipse.papyrus.infra.ui` bundle. Accordingly, its identifier is changed to `org.eclipse.papyrus.ui.toolbar` to better reflect its role as **the** Papyrus toolbar.

Views Dependencies

Several of the Infra Layer bundles (not only diagram infrastructure but also the infrastructure core, including bundles that should be headless) have dependencies in the 1.x releases on the Properties Model from the `org.eclipse.papyrus.views.properties.model` bundle and also its `.edit` counterpart for bundles that extend/re-used the properties model. This dependency is a violation of the strictly acyclic dependencies between Papyrus layers.

Accordingly, these model bundles are refactored into the Infra Layer. The bundle names are changed according to the following table

Old Bundle	New Bundle
*.views.properties.model	*.infra.properties
*.views.properties.model.edit	*.infra.properties.edit
*.views.properties.model.editor	*.infra.properties.editor

Along with this, the model API packages are correspondingly renamed:

Old Package	New Package
*.views.properties.contexts	*.infra.properties.contexts
*.views.properties.contexts.impl	*.infra.properties.contexts.impl
*.views.properties.contexts.util	*.infra.properties.contexts.util
*.views.properties.environment	*.infra.properties.environment
*.views.properties.environment.impl	*.infra.properties.environment.impl
*.views.properties.environment.util	*.infra.properties.environment.util
*.views.properties.ui	*.infra.properties.ui
*.views.properties.ui.impl	*.infra.properties.ui.impl
*.views.properties.ui.util	*.infra.properties.ui.util

as well as a non-model package that is serves persistence of context and environment models:

Old Package	New Package
*.views.properties.catalog	*.infra.properties.catalog

This moved package includes the `PropertiesCatalog` EMF resource implementation and the `PropertiesURIHandler` that implements its `ppe:` URI scheme.

Extension points are moved or split:

- the `org.eclipse.papyrus.views.properties.environment` point is renamed `org.eclipse.papyrus.infra` (note the plural, now) and is otherwise unchanged
- the `org.eclipse.papyrus.properties.context` point is split into two. A new `org.eclipse.papyrus.infra.p` point (note the plural, now) defines the `<context>` element for registration of context models. The old extension point is retained, minus the `<context>` element, for association of contexts with preference pages via the `<preferencePageBinding>` element