section *ModuleManagerTest* **parents** *ModuleManager*

This specification defines several unit tests for the Module Manager.

We start by defining some members of the given sets, so that we can use meaningful names within our examples.

$$
\begin{array}{l}
networkReaderClass == 10 \\
pvReaderClass == 11 \\
pvAssetFactory == 12 \\
pvAllocationReader == 13 \\
pvCharacteristicsReader == 14 \\
networkAssetFactory == 15 \\
pvMinorParameterReader == 16 \\
pvExactAllocationReader == 17 \\
pvAssetSpecificReader == 18 \\
pvAssetCommonReader == 19 \\
extAssetFactory == 20 \\
extAgentFactory == 21 \\
extDataProvider == 22 \\
dataIdNetworkData == 30 \\
dataIdBillingData == 31 \\
dataIdPvAssetDataAllocMinor == 32 \\
dataIdPvAssetDataAllocExact == 33 \\
dataIdPvSystems == 34 \\
dataIdPvCharacteristics == 35 \\
dataIdPvExactAllocation == 36 \\
pathTownsvilleNetwork == 40 \\
pathTownsvilleBillingData == 41 \\
pathMinorParameterCsv == 42 \\
pathExactPvAllocationCsv == 43 \\
pathTownsvillePvInstallsCsv == 44 \\
pathCommonPvCsv == 45 \\
contributorPvAsset == 50 \\
contributorPvAssetReader == 51
\end{array}
$$

First we define an example factory (for PhotoVoltaic Solar Panels), with a compulsory property called *pvCharacteristics* that takes any data provider that subclasses *pvCharacteristicsReader* (these data providers provide a list of typical PV systems), and an optional getter-setter called *exactAllocation* that takes a data provider of type *pvExactAllocationReader* (this is used when precise information is available about which houses have PV systems, and the details of those systems).

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{pvCharacteristics}}\\
\hline
SetterGetter \\
\hline
name = dataIdPvCharacteristics \\
argType = pvCharacteristicsReader \\
value = nullClass \\
optional = NO \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{exactAllocation}}\\
\hline
SetterGetter \\
\hline
name = dataIdPvExactAllocation \\
argType = pvExactAllocationReader \\
value = nullClass \\
optional = YES \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{PvAssetFactory}}\\
\hline
AssetFactory \\
\hline
className = pvAssetFactory \\
consumes = \{\} \\
prior = \{\} \\
methods = pvCharacteristics \cup exactAllocation \\
\hline
\end{array}$$

We also prove that these two getter-setters and the factory are correctly and unambiguously defined (we have not defined any contradictory properties, or left any properties unspecified), as follows:

theorem pvCharacteristicsIsValid
$\vdash ?\ \#\ pvCharacteristics = 1$

theorem exactAllocationIsValid
$\vdash ?\ \#\ exactAllocation = 1$

theorem PvAssetReaderFactoryIsValid
$\vdash ?\ \#\ PvAssetFactory = 1$

However, when we ask ZLive to prove these conjectures, the first two fail, because we specified a Java class name rather than a Java type for 'argType'.

This is a minor error that is picked up by the Z typechecker when we use separate given types for each kind of value, or is picked up by the above conjectures when we just use integers for animation purposes. Reflecting on these failures led us to revise our model so that the *value* and *argType* are now both Java class names, representing the actual and expected types of parameter, respectively.

Next we define an example plugin module.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{EgModulePvAsset}} \\
\hline
\textit{Module} \\
\hline
\textit{contributor} = \textit{contributorPvAsset} \\
\textit{classes} = \textit{PvAssetFactory} \\
\hline
\end{array}
$$

theorem EgModulePvAssetIsValid
$\vdash ? \; \# \, \textit{EgModulePvAsset} = 1$

Now we test each of the four cases of the *SatisfySetterMethod*.

1. When there are no matching data providers and the setter is compulsory:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{testNoMatch}} \\
\hline
\textit{SatisfySetterMethod} \\
\hline
m \in \textit{pvCharacteristics} \\
\textit{dataProviders}? = \{\} \\
\textit{outputid} = \{\} \\
\hline
\end{array}
$$

Animating this test in ZLive gives the following (unique) result – note that the output *value*! is set to *dataProviderError* (8):

$1 : \langle\!| m == \langle\!| \textit{name} == 35, \textit{argType} == 14, \textit{value} == 9, \textit{optional} == 0 |\!\rangle,$
$m' == \langle\!| \textit{name} == 35, \textit{argType} == 14, \textit{value} == 8, \textit{optional} == 0 |\!\rangle,$
$\textit{dataProviders}? == \{\}, \textit{outputid} == \{\}, \textit{matching}! == \{\}, \textit{value}! == 8 |\!\rangle$

2. When there are no matching data providers and the setter is optional:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{testNoMatchOptional}} \\
\hline
\textit{SatisfySetterMethod} \\
\hline
m \in \textit{exactAllocation} \\
\textit{dataProviders}? = \{\} \\
\textit{outputid} = \{\} \\
\hline
\end{array}
$$

ZLive gives:

$$1 : \langle\!| m == \langle\!| name == 36, argType == 17, value == 9, optional == 1 |\!\rangle,$$
$$m' == \langle\!| name == 36, argType == 17, value == 9, optional == 1 |\!\rangle,$$
$$dataProviders? == \{\}, outputid == \{\}, matching! == \{\}, value! == 9 |\!\rangle$$

This leaves the $m.value$ field unchanged (*nullClass*) and returns *nullClass* in $value!$ as well, which indicates that no error has occurred.

3. When there are several matching data providers, ZLive reports a unique solution, with the $value!$ output set to *dataProviderError* (8):

$$\begin{array}{l}
\textit{testSeveralMatch} \\
\hline
\textit{SatisfySetterMethod} \\
\hline
m \in pvCharacteristics \\
dataProviders? = \{pvCharacteristicsReader, pvExactAllocationReader\} \\
outputid = \{pvCharacteristicsReader \mapsto dataIdPvCharacteristics, \\
\qquad pvExactAllocationReader \mapsto dataIdPvCharacteristics\}
\end{array}$$

ZLive gives:

$$1 : \langle\!| m == \langle\!| name == 35, argType == 14, value == 9, optional == 0 |\!\rangle,$$
$$m' == \langle\!| name == 35, argType == 14, value == 8, optional == 0 |\!\rangle,$$
$$dataProviders? == \{14, 17\}, outputid == \{(14, 35), (17, 35)\},$$
$$matching! == \{14, 17\}, value! == 8 |\!\rangle$$

4. Finally, we test the sweet path, where there is exactly one match.

$$\begin{array}{l}
\textit{testUniqMatch} \\
\hline
\textit{SatisfySetterMethod} \\
\hline
m \in pvCharacteristics \\
dataProviders? = \{pvCharacteristicsReader, pvExactAllocationReader\} \\
outputid = \{pvCharacteristicsReader \mapsto dataIdPvCharacteristics, \\
\qquad pvExactAllocationReader \mapsto dataIdPvExactAllocation\}
\end{array}$$

In this case, ZLive updates $m'.value$ to the unique matching data provider class (*pvCharacteristicsReader*), and also returns that class in $value!$.

ZLive returns:

$$1 : \langle\!| m == \langle\!| name == 35, argType == 14, value == 9, optional == 0 |\!\rangle,$$
$$m' == \langle\!| name == 35, argType == 14, value == 14, optional == 0 |\!\rangle,$$
$$dataProviders? == \{14, 17\}, outputid == \{(14, 35), (17, 36)\},$$
$$matching! == \{14\}, value! == 14 |\!\rangle$$

This testing approach uses the set-oriented nature of Z to check that each of the four test cases is correctly defined, with no inconsistent values, no missing/unspecified values, and returns a unique solution (a singleton set) that contains the expected results. This gives us strong confidence that the specified operation is correct, and that it has the four behaviours that we desire.

The next step is to test the lifted operation *SatisfyFactoryInputs* that sets all the properties of a whole factory. To do this, we need an example instance of a *ModuleManager* that contains some *Module* objects.

$$\begin{array}{|l}\hline \multicolumn{1}{l}{\mathit{ExamplePvModule}} \\ \mathit{Module} \\ \hline contributor = contributorPvAsset \\ classes = PvAssetFactory \\ \hline \end{array}$$

$$\begin{array}{|l}\hline \multicolumn{1}{l}{\mathit{ExampleModuleManager}} \\ \mathit{ModuleManager} \\ \hline modules = ExamplePvModule \\ \hline \end{array}$$

As usual, we check that these examples are uniquely defined.

theorem ExamplePvModuleIsValid
$\vdash ?\ \# ExamplePvModule = 1$

theorem ExampleModuleManagerIsValid
$\vdash ?\ \# ExampleModuleManager = 1$

$$\begin{array}{|l}\hline \multicolumn{1}{l}{\mathit{testSatisfyFactoryInputs1}} \\ \mathit{SatisfyFactoryInputs} \\ \mathit{ExampleModuleManager} \\ \hline factory? \in PvAssetFactory \\ dataProviders? = \{pvCharacteristicsReader, pvExactAllocationReader\} \\ outputid = \{pvCharacteristicsReader \mapsto dataIdPvCharacteristics, \\ \qquad\qquad pvExactAllocationReader \mapsto dataIdPvExactAllocation\} \\ \hline \end{array}$$