Part Handling in qooxdoo 3

Thomas Herchenröder
1&1 Internet AG

Parts

Parts are a means to *logically* partition a qooxdoo application, so that those parts can be loaded *incrementally* and *on demand*. Parts are defined through configuration, and the *Generator* distributes class code and resource information across multiple script files that are then retrieved via HTTP. The aim is to avoid loading of unnecessary code and data into the browser.

Concepts

The Generator collects *class* code into *scripts* (.js files). Scripts are grouped into *packages*. Scripts of the same package are always loaded together. Each *part* is implemented by a collection of packages, each package might be required by multiple parts. qooxdoo's *PartLoader* loads all packages for a part that has been required, but have not been loaded yet.

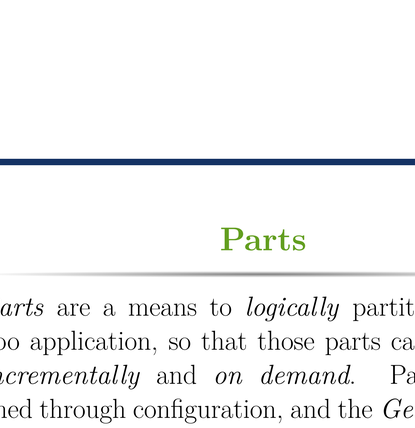

Figure 1: Relations between parts, packages, scripts and classes.

Configuring Parts

What to put in the *include* key of the part definitions. "include list" means all entries after glob expansion.

- the *boot* part should have the same include definition as the application
- include lists must be *free of overlaps*
- *load dependencies* of one part must not be in include lists of other parts
- don't define include lists along *physically boundaries* (name spaces, libraries, ...)
- don't define parts with *framework classes*

Some of these criteria a checked by the Generator.

2-Phase Package Calculation

Assign classes to packages in a 2-step process:

- **equivalence sets** group all classes that are required by the same set of parts
- **merging** (or collapsing) resolve smaller packages into larger ones

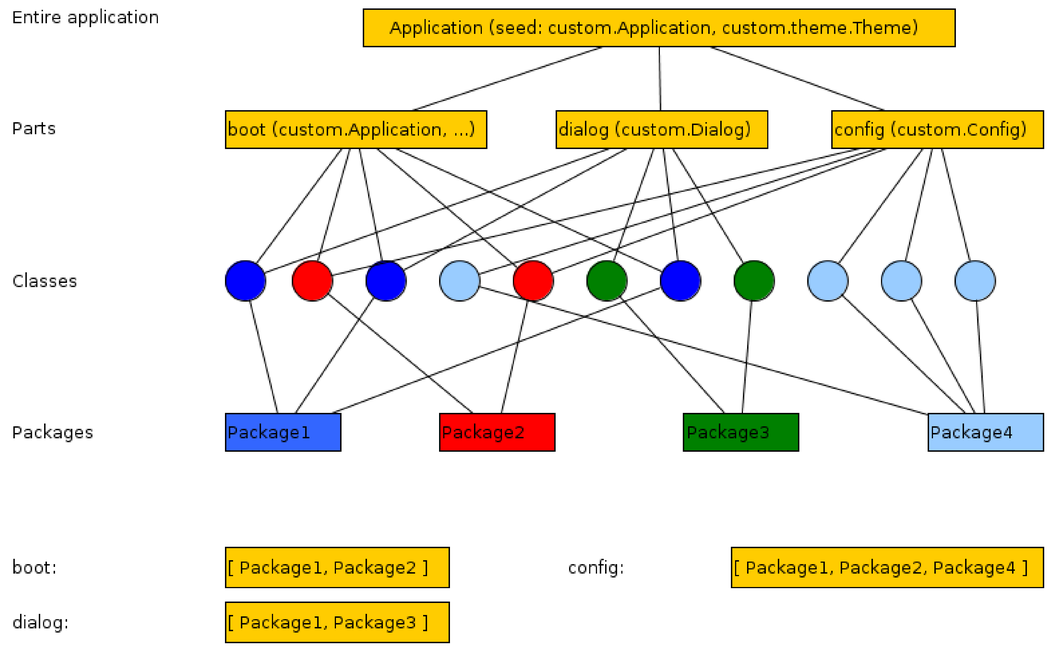

Figure 2: Mapping parts to classes, and classes to packages.

Part Assertions

- parts are **lazy run deps**
- each part is **self-contained** *wrt* load deps
- this is also true for *collapsePartsByOrder*, parts can be loaded out-of-order
- every class is only loaded **once**
- classes are **load-ordered** within a package, packages are load-ordered within a part

Equivalence sets

To construct the equivalence sets for the classes:

- calculate the class list for each part (starting from the part's *include* config)
- assigne each class the parts which require it
- group classes that are required by the same parts

If N is the number of defined parts then

$$max(count(equivalence\_sets(N))) = 2^N - 1$$

Package Dependencies

If classes $c1, c2$ and packages $p1, p2$, then

$$if\ c1 \in p1\ and\ c2 \in p2\ and\ depends(c1, c2) \Rightarrow depends(p1, p2)$$

Package Merging

Classes from the yielding package are added to the receiving package. The yielding package is removed. Let $p1$ be a package for merging into $p2$, $P(x)$ be the set of parts a package is used in, $Deps(x)$ be the depedencies of a class or package, then

- $P(p1) \subset P(p2)$ [$p2$ must at least be used where $p1$ is used]
- after the merge: $for\_each\_class\ c \in p1 : Deps(c) \subset p1\ and\ ordered(p1)$
- $replace(p1, p2)\ in\ all\ P(p1)$ [use $p2$ in all parts where $p1$ was used]

This is done with *names*, no code is shuffled around. Merging can be more aggressive if *collapsePartsByOrder* is given in config.

Package calculation log (Feedreader):

```
>> Assembling parts - part addfeed -
Part #addfeed depends on 245 classes -
part boot - Part #boot depends on 364
classes - part settings - Part #settings
depends on 274 classes
>> Package summary : 6 packages
- Package #7 contains 228 classes
- [<Class:qx.module.Animation>,
<Class:qx.ui.layout.Atom>,
<Class:qx.bom.clien - Package #7
depends on these packages: [] -
Package #6 contains 23 classes
- [<Class:qx.io.PartLoader>,
<Class:qx.bom.request.Script>,
<Class:qx.io.part. - Package #6
depends on these packages: ['7']
- Package #5 contains 4 classes -
[<Class:qx.ui.form.IColorForm>,
<Class:qx.ui.form.Resetter>,
<Class:qx.ui.fo - Package #5
depends on these packages: ['7']
- Package #4 contains 19 classes
- [<Class:qx.ui.form.RadioGroup>,
<Class:qx.ui.form.RadioButton>,
```

References

[1] "Parts and Packages Overview", qooxdoo Manual, http://bit.ly/1hL6cqU

[2] "Using Parts", qooxdoo Manual, http://bit.ly/17y8MMH

[3] "Generator Config Keys - packages", qooxdoo Manual, http://bit.ly/177sNXW

[4] qx.io.PartLoader, qooxdoo API, http://bit.ly/1cvQM2Q