Part Handling in qooxdoo

Thomas Herchenröder, 1&1 Internet AG (November 20, 2013)

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.

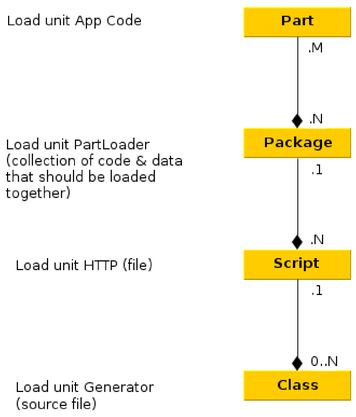

Figure 1: Relations between parts, packages, scripts and classes

Configuring Parts

Specifying 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 *physical boundaries* (name spaces, libraries, ...)
- don't define parts with *framework classes*

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
  - by order (*collapse groups*)
  - by size

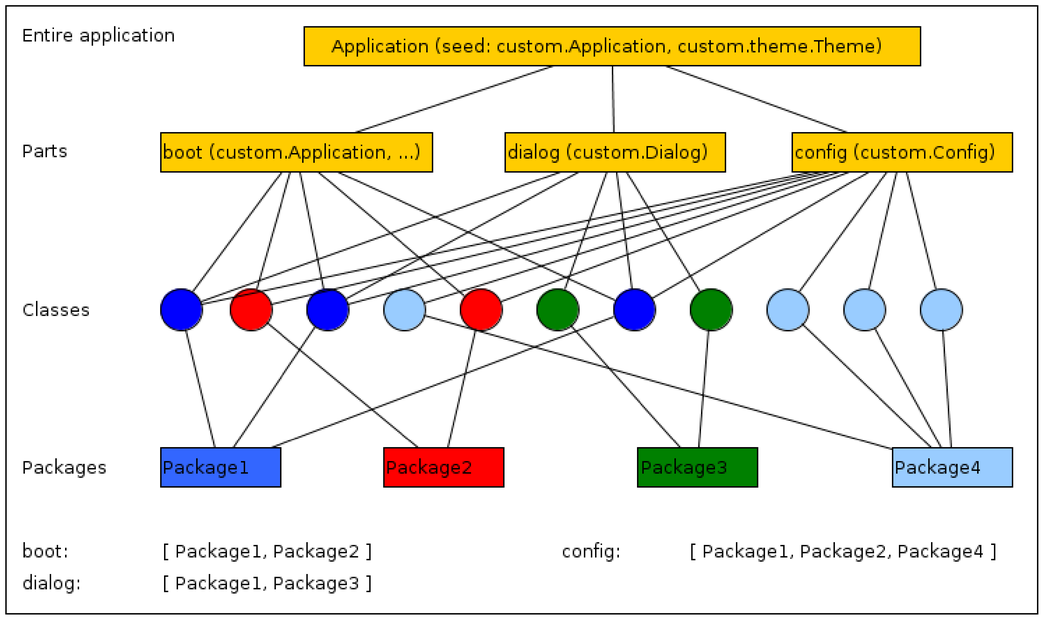

Figure 2: Mapping parts to classes, and classes to packages

Part Assertions

- parts are **lazy run deps**
- each part is **self-contained** *wrt* load deps
- this is also true for *expected-load-order*, 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:

- **part class list**: calculate the class list starting from part's *include*, skipping other seeds
- **class labeling**: assign each class the parts which require it
- **classify**: group classes that are required by the same parts

If N is the number of defined parts then

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

References

[1] "Parts and Packages Overview", qooxdoo Manual, http://manual.qooxdoo.org/3.0.x/pages/development/parts_overview.html

[2] "Using Parts", qooxdoo Manual, http://manual.qooxdoo.org/3.0.x/pages/development/parts_using.html

[3] "Generator Config Keys - packages", qooxdoo Manual, http://manual.qooxdoo.org/3.0.x/pages/tool/generator/generator_config_ref.html#package

[4] qx.io.PartLoader, qooxdoo API, http://demo.qooxdoo.org/3.0.x/apiviewer/index.html#qx.io.PartLoader

Package Dependencies

Classes c_1, c_2, packages p, p_1, p_2, part P_1, then

- $if\ c_1 \in p_1\ and\ c_2 \in p_2\ and\ depends(c_1, c_2) \Rightarrow depends(p_1, p_2)$
- $ordered(P_1) \Leftrightarrow ordered(Packages(P_1))\ and\ \forall p \in Packages(P_1) : ordered(p)$

Package Merging

Let p_1 be a package for merging into p_2, $Parts(x)$ be the set of parts a package is used in, $Deps(x)$ be the load depedencies of a class or package, then

- Classes(p_1) go into p_2, p_1 is removed
- $Parts(p_1) \subset Parts(p_2)$ [p_2 must at least be used where p_1 is used]
- after the merge: $\forall P \in Parts(p_2) : ordered(P)$
- $\forall P \in Parts(p_2) : Deps(p_1) \subset P$ [dependencies of p_1 must be fulfilled wherever p_2 is used]
- *expected-load-order* allows more aggressive merging
- some classes will be loaded where not needed
- the *parts verifier* checks these constraints

```
>>> Creating part structures...
  - Part #addfeed => 1
  - Part #boot => 2
  - Part #settings => 4
>>> 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.
  - Package #7 depends on these packages: []
  - Package #6 contains 23 classes
  - [<Class:qx.io.PartLoader>, <Class:qx.bom.request.Script>, <Class:qx
  - Package #6 depends on these packages: ['7']
  - ...
>>> Part summary : 3 parts
  - Part #addfeed packages(3): #7, #5, #1
  - Part #boot packages(3): #7, #6, #2
  - Part #settings packages(4): #7, #5, #6, #4
  - Total of packages used in parts: 10
>>> Collapsing parts
>>> Collapsing parts by collapse order...
  - Collapse group 0 [u'boot']
    - collapsing unique packages...
      - Search a target package for package #2
        - Merge package #2 into #6
          - Adding packages dependencies to target package: ['7']
          - Target package #6 now depends on: ['7']
      - ...
>>> Collapsing parts by package sizes...
  - ...
>>> Package summary : 4 packages
  - Package #7 contains 364 classes
  - [<Class:qx.module.Animation>, <Class:qx.ui.layout.Atom>, <Class:qx.
  - Package #7 depends on these packages: []
  - Package #5 contains 4 classes
  - [<Class:qx.ui.form.IColorForm>, <Class:qx.ui.form.Resetter>, <Class
  - Package #5 depends on these packages: ['7']
  - ...
>>> Part summary : 3 parts
  - Part #addfeed packages(3): #7, #5, #1
  - Part #boot packages(1): #7
  - Part #settings packages(3): #7, #5, #4
  - Total of packages used in parts: 7
```

Figure 3: Part creation verbose output (excerpt)