16

Ouvrir l'application vers l'extérieur

Lors du développement d'une application, il est très fréquent d'accéder à des ressources externes à l'application elle-même. Les objectifs d'une telle manœuvre sont nombreux. Il peut s'agir de fournir à l'utilisateur un contenu dynamique, par exemple issu de services web, de sauvegarder ses préférences dans une base de données locale de sorte à les lui restaurer lors de sa prochaine session ou même d'accéder à des informations plus spécifiques situées sur une base de données non pas locale mais distante.

Attention › *Ce chapitre requiert l'étude préalable des chapitres précédents traitant la programmation d'interfaces Qt Quick avec JavaScript ainsi que l'association entre Qt Quick et le C++.*

16.1. Ouvrir l'application vers le réseau

Dans cette section, nous allons voir comment le QML permet les interactions implicites et explicites entre une application et le réseau et comment Qt Quick gère ces interactions. Afin d'illustrer la notion, des exemples variés seront donnés, accompagnés par l'implémentation d'un cas de figure assez répandu de nos jours : la communication entièrement en QML et en JavaScript entre une application et un service web.

La transparence réseau de Qt Quick

Certains composants QML mettent à disposition des propriétés et/ou des fonctions prenant en paramètre une URL. Par le biais de sa transparence réseau, le QML permet au développeur de ne pas à avoir à se préoccuper de l'emplacement du contenu auquel il souhaite accéder, ce qui lui permet notamment de ne pas avoir à faire de distinction entre un contenu local et un contenu distant.

```
Image {
    id: img1
    source: "logo.png" ❶
    anchors {
        top: parent.top
        horizontalCenter: parent.horizontalCenter
    }
}
Image {
```

```
    source: "http://www.exemple.com/logo.png" ❷
    anchors {
        top: img1.bottom
        horizontalCenter: parent.horizontalCenter
    }
}
```

❶ Associe une URL relative à la propriété `source` d'un premier composant Image.

❷ Associe une URL absolue à la propriété `source` d'un second composant Image.

La résolution des URL relatives est effectuée selon l'emplacement de la source du fichier QML contenant l'URL en question. Ainsi, si le fichier QML contenant l'exemple précédent est situé à l'adresse `http://www.exemple.com/Frame.qml`, alors l'URL relative `logo.png` va avoir pour URL finale `http://www.exemple.com/logo.png`. De même, s'il est situé sur `/home/Frame.qml`, alors l'URL finale va être `/home/logo.png`.

Les URL ne sont pas les seuls éléments à être affectés par la transparence réseau intégrée avec le QML. En effet, les directives `import` sont également affectées de façon identique lorsqu'elles sont associées à un espace de noms par la biais du mot clé `as` (voir Démarrer en QML). Sans ce mot clé, `import` ne prend effet que pour les ressources situées en local.

```
import "components" ❶
import "components" as Cmp ❷
import "script.js" as Scr ❸
```

❶ Import d'un dossier `components` dans le cadre d'une utilisation locale seulement.

❷ Import d'un dossier `components` pouvant profiter de la transparence réseau.

❸ Import d'un script `script.js` pouvant profiter de la transparence réseau.

Le système de ressources de Qt permet au développeur de compiler les sources QML directement dans l'exécutable et d'y accéder par le biais du schéma `qrc`. Du point de vue de la transparence réseau, l'accès au contenu par le biais d'URL relatives est fait au même titre qu'une ressource située dans le système de fichiers.

```
Loader {
    source: "qrc:/frames/Frame.qml"
    anchors.fill: parent
}

// Frame.qml
```

```
import "../components" as Cmp

Cmp.Button {
    icon: "../images/back.png"
}
```

Note › *Dans cet exemple, le fichier `Frame.qml` étant chargé depuis le système de ressources, l'URL relative de la propriété `icon` du bouton s'y trouvant va avoir pour URL finale `qrc:/images/back.png`. Quant au bouton, il va être chargé depuis `qrc:/components/Button.qml`.*

Chargement progressif des ressources

Quand une application charge des éléments depuis le réseau, de multiples cas de figure peuvent survenir. Est-ce que la ressource a fini d'être récupérée ? Est-ce qu'une erreur est survenue lors du chargement ? Afin de gérer ce genre d'interrogation, QML associe des propriétés `status` et `progress` aux composants permettant de charger des ressources qui informent de l'état et de l'avancée du chargement de celles-ci, à l'image d'un navigateur qui charge progressivement une page web et informe de la progression.

La propriété `status` est une énumération informant de l'état du chargement. Le Tableau 16.1 récapitule les valeurs qu'elle peut prendre.

Tableau 16.1 : *Valeurs possibles de la propriété `status`*

Valeur	Description
`Null`	La ressource n'a pas été définie.
`Ready`	La ressource a été chargée avec succès.
`Loading`	La ressource est en cours de chargement.
`Error`	La ressource n'a pas pu être chargée à cause d'une erreur.

Note › *Certains composants tels que `Component` mettent à disposition une méthode `errorString()` détaillant dans la chaîne de caractères retournée l'erreur ayant pu se produire dans le cas de la valeur `Error`. Par exemple :*

```
var component = null;
function doSomething() {
    component = Qt.createComponent("http://invalid.com/Invalid.qml");
    if (component.status === Component.Loading) {
        component.statusChanged.connect(somethingFinished);
    } else {
        somethingFinished();
    }
}
function somethingFinished() {
```

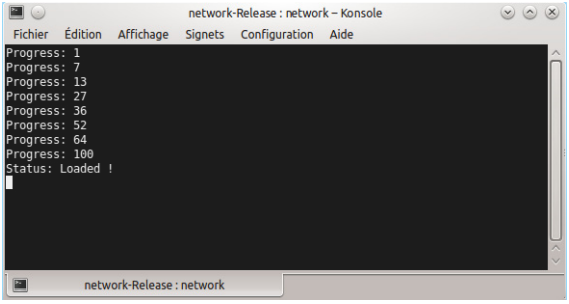
```
    if (component.status !== Component.Ready) {
        console.log(component.errorString());
    }
}
```

Quant à la propriété `progress`, elle représente un nombre réel de 0.0 à 1.0 permettant de décrire l'avancée du chargement.

Les Exemple 16.1 et Exemple 16.2 établissent un parallèle entre l'état et l'avancée du chargement d'un composant tel qu'Image et d'une page web.

Exemple 16.1 : *État et avancée du chargement d'un composant Image*

```
Image {
    source: "http://www.d-booker.fr/img/logo.jpg"
    onProgressChanged: console.log("Progress: " +
 Math.floor(progress * 100))
    onStatusChanged: {
        var baseMsg = "Status: ";
        if (status === Image.Error) {
            console.log(baseMsg + "Image.Error");
        } else if (status === Image.Ready) {
            console.log(baseMsg + "Loaded !");
        }
    }
}
```

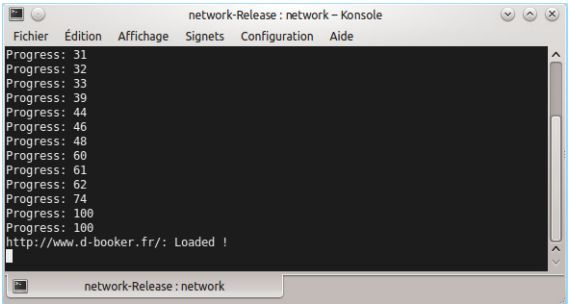


Sortie standard obtenue lors du chargement d'un composant Image par transparence réseau

Exemple 16.2 : *État et avancée du chargement d'un composant WebView*

```
import QtQuick 2.0
import QtWebKit 3.0
```

```
WebView {
    url: "http://d-booker.fr"
    onNavigationRequested: {
        var schema = /^(https|http):\/\/(.+\.)?d-booker.fr/;
        if (schema.test(request.url)) {
            request.action = WebView.AcceptRequest;
        } else {
            request.action = WebView.IgnoreRequest;
        }
    }
    onLoadProgressChanged: console.log("Progress: " + loadProgress)
    onLoadingChanged: {
        var baseMsg = loadRequest.url + ": ";
        if (loadRequest.status === WebView.LoadFailedStatus) {
            console.log(baseMsg + loadRequest.errorString);
        } else if (loadRequest.status ===
 WebView.LoadSucceededStatus) {
            console.log(baseMsg + "Loaded !");
        }
    }
}
```



Sortie standard obtenue lors du chargement progressif d'une page dans un WebView

Note *› Bien qu'un parallèle puisse être établi entre les propriétés* `status` *et* `loadStatus` *ainsi que les propriétés* `progress` *et* `loadProgress` *des composants Image et WebView, ces propriétés sont bien distinctes dans le sens où le composant WebView présente des propriétés qui lui sont propres et qui ne s'utilisent pas nécessairement de la même manière que les propriétés* `status` *et* `progress` *des composants permettant de charger des ressources externes.*

Le chargement progressif soulève une dernière question : qu'en est-il de l'aspect synchrone ou asynchrone du chargement d'une ressource ?

Afin de ne pas geler l'interface utilisateur, QML privilégie le chargement des ressources de manière asynchrone, notamment en faisant en sorte que toutes les ressources chargées depuis le réseau soient chargées par le biais d'un thread séparé. Cependant, dans