qftestJUI™

VERSION 1.08.4

DIDACTICIEL

Editeur : Quality First Software GmbH

Tulpenstr, 41 - DE-82538 Geretsried - Allemagne

www.qfs.de

Distributeur France : KAPITEC SOFTWARE S.A.S.

14, avenue Jean Bouin - 31620 Fronton - France

Tél. : +33-5 34 27 90 03 - Fax : +33-5 34 27 90 04

sales@kapitec.com - www.kapitec.com

CHAPITRE 1 - INTRODUCTION

Ce didacticiel est une introduction et une présentation à l'utilisation de l'outil de test d'IHM Java/Swing *qftestJUI*. Il vous présente les différents composants et fonctionnalités de *qftestJUI*, et il vous guide au travers des étapes nécessaires pour installer votre propre suite de test. Il contient également des rubriques plus avancées, telles que l'utilisation du débogueur de *qftestJUI*.

Ce didacticiel se développe et s'enrichit en même temps que *qftestJUI* avec l'ajout de nouveaux exemples dès lors que de nouvelles fonctionnalités sont disponibles. Pour cette raison, nous avons essayé de garder les chapitres de ce didacticiel indépendants les uns des autres, ainsi vous pouvez toujours revenir à un chapitre et travailler sur de nouveaux chapitres sans devoir repasser par les précédents.

Afin de rendre ce didacticiel toujours plus utile, nous avons besoin de votre aide. Il est toujours difficile pour les développeurs d'un produit logiciel de deviner quels sont les dispositifs importants pour les utilisateurs ; certains sont faciles à comprendre et à utiliser, et d'autres restent un mystère. C'est pourquoi nous vous encourageons vivement à nous donner votre avis sur les points que vous souhaiteriez voir approfondis dans le présent didacticiel. Bien évidemment, nous souhaitons également connaître les éventuels problèmes que vous rencontrez, que ce soit des exemples qui ne fonctionnent pas pour vous ou parce que vous avez manqué tel ou tel point. Avant toute chose ce sont vos commentaires qui permettent de déterminer la qualité de ce didacticiel. N'hésitez pas à nous adresser vos commentaires, vos souhaits ou toute anomalie détectée par courriel : qftestJUI-tutorial@qfs.de ou support@kapitec.com

CHAPITRE 2 - OPTIONS DEMO

2.1 Démarrage de qftestJUI et Chargement de la Suite de Test

Après avoir démarré *qftestJUI*, vous pouvez immédiatement utiliser notre premier exemple de suite de test. Sélectionnez le menu **File -> Open** puis le répertoire **qftestJUI-1.08.4/doc/tutorial** depuis votre répertoire d'installation *qftestJUI*. Cliquez sur le fichier **Options.qft**. qftestJUI charge alors la suite de test, comme montré ci-dessous :

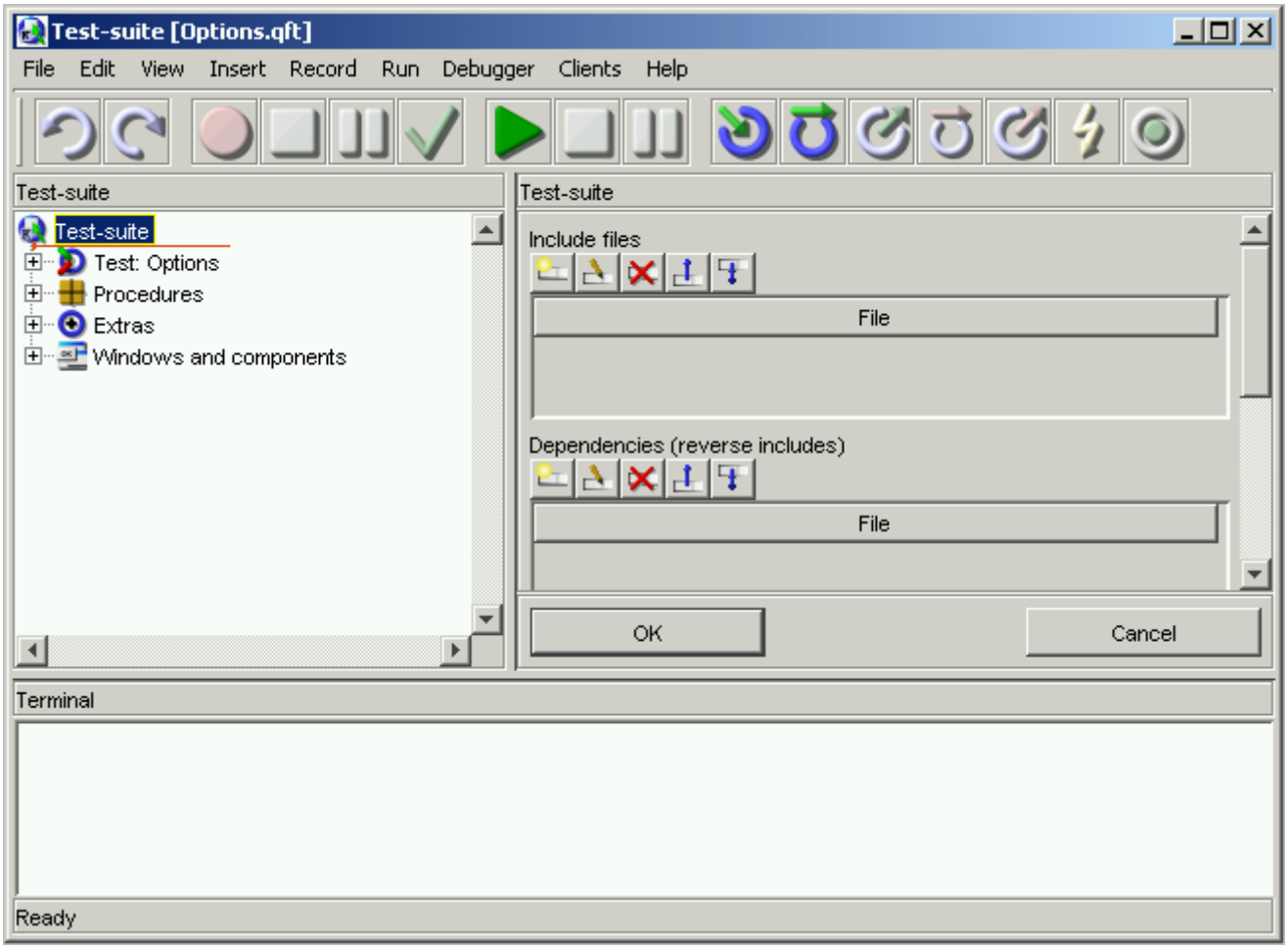

Figure 2.1 - Suite de Test Options.qft

La partie gauche de la fenêtre principale contient l'arborescence de la suite de test. La partie droite affiche les détails d'un nœud sélectionné. Si la vue Détails ne s'affiche pas, sélectionnez le menu **View -> Details**. Dans la zone inférieure de la fenêtre principale vous voyez la zone du Terminal, qui affiche les messages standards et les communications entre votre suite de test et l'application cliente que vous testez. Vous pouvez afficher ou non le Terminal en sélectionnant le menu **View -> Terminal -> Show**.

Avec la structure arborescente dans la fenêtre principale, vous pouvez naviguer et sélectionner des nœuds individuels de votre suite de test. Lorsqu'un nœud est sélectionné, ses propriétés s'affichent dans la zone Détails (partie droite de la fenêtre principale). Cliquez sur le nœud **Test: Options** pour ouvrir les nœuds du dessous. Vous verrez que ce nœud contient un nœud pour le **Setup**, trois nœuds test (un nœud test et deux nœuds séquences pour être précis), et un nœud pour le **Cleanup**.

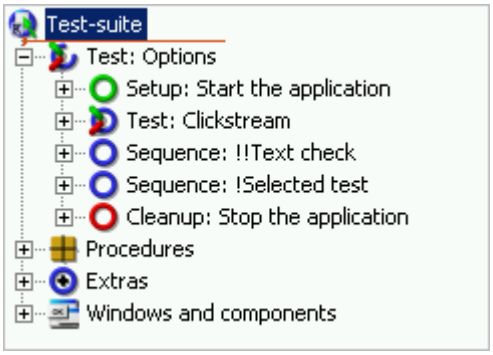

Figure 2.2 - Contenu du Nœud Test: Options

2.2 Démarrage de l'Application

La première étape consiste à examiner le nœud **Setup**. Ouvrez maintenant ce nœud afin de voir son contenu comme montré ci-dessous :

Figure 2.3 - Nœud Setup

Dans le nœud Setup, vous voyez alors trois nœuds fils :

- **Start Java SUT client** - démarre l'application pour le client que vous allez tester, appelé le SUT (*System Under Test*).
- **Wait for client to connect** - attend que la nouvelle Machine Virtuelle Java (JVM), avec laquelle votre SUT s'exécutera, s'attache à *qftestJUI*.
- **Wait for component** - attend l'apparition d'un certain composant du SUT.

Pour cet exemple, nous attendons la fenêtre principale du SUT. Ne vous inquiétez pas si cette étape ne vous semble pas complètement claire, nous apporterons ultérieurement des explications sur les composants et leurs significations.

Maintenant cliquez sur le nœud **Start Java SUT client** de sorte qu'il soit sélectionné comme montré ci-dessous :

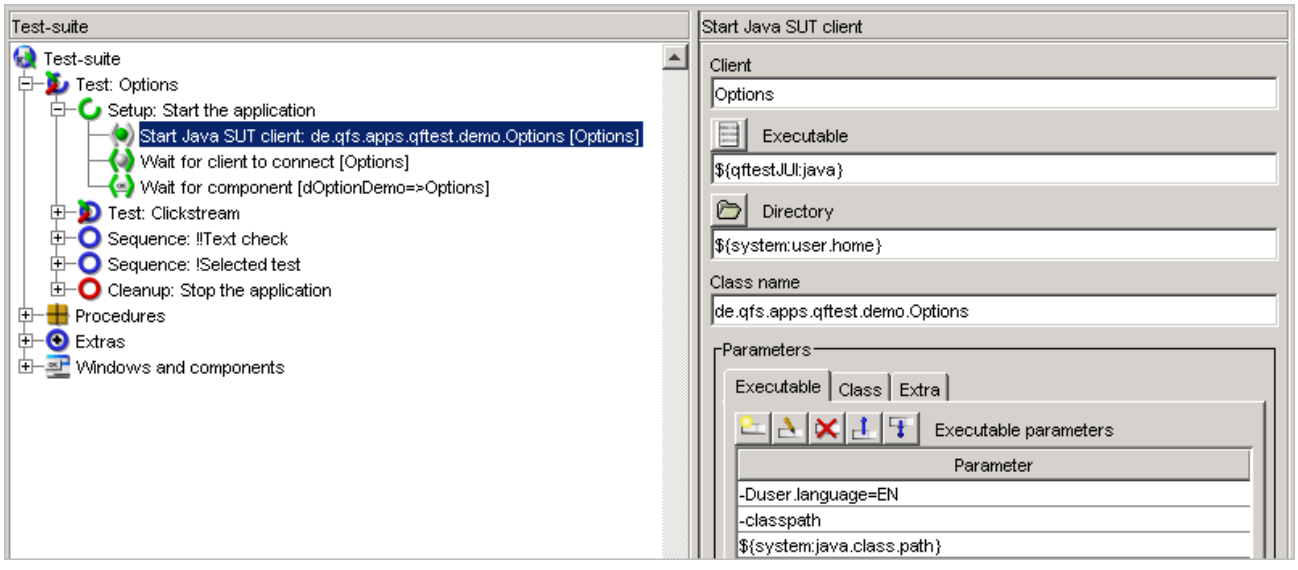

Figure 2.4 - Le Nœud Start Java SUT Client

Regardons plus en détail les propriétés de ce nœud, telles q'elles sont affichées dans la fenêtre Détails (côté droit).

- Dans le premier champ l'étiquette **Client** est un nom sans équivoque pour l'application SUT. Vous pouvez donner au SUT le nom que vous souhaitez, mais ce même nom sera utilisé comme référence au SUT dans tous les autres nœuds. En d'autres termes, ce nom doit systématiquement être utilisé.
- **Executable** est le deuxième champ, dans lequel le nom de l'exécutable du programme est spécifié. Pour notre exemple, la variable **${qftestJUI:java}** a été donnée. Cette variable correspond au programme Java que vous avez choisi à l'installation de *qftestJUI*. Sinon vous pouvez rentrer manuellement le chemin d'accès à votre JVM, avec l'exécutable Java. Avec le bouton **Select executable file**  vous pouvez entrer dans un exécutable avec un navigateur standard de fichiers.
- Le champ **Directory** indique le répertoire de travail pour le SUT une fois démarré.
- Dans le quatrième champ, l'étiquette **Class name** spécifie la classe Java utilisée pour implémenter le SUT. La fonction **main()**, qui sera appelée par Java au démarrage du SUT, doit être dans cette classe.
- Dans la section **Parameters** vous voyez les arguments en ligne de commande pour le SUT. Dans cet exemple, nous spécifions les arguments pour paramétrer la langue de l'application à Anglais. De plus, le classpath est paramétré correctement, ce qui est nécessaire parce qu'une classe de *qftestJUI* est utilisée comme SUT.
- Les autres champs de propriétés peuvent être laissés vides et ils ne sont pas requis pour notre exemple. Toutefois vous remarquerez qu'un commentaire a été ajouté (dans le dernier champ). Tous les nœuds ont un champ commentaire facultatif, utile au développeur-testeur pour documenter l'objet d'un nœud.

Nous sommes maintenant prêts pour démarrer le SUT. Cliquez à nouveau sur le nœud **Setup** (pour qu'il soit sélectionné), mais encore ouvert (les nœuds fils doivent être visibles). Maintenant cliquez sur le bouton **Replay** (rejeu)  . Ce bouton permet d'exécuter le nœud sélectionné, à savoir **Setup** dans notre cas.

Quand un nœud s'exécute, *qftestJUI* utilise la structure arborescente pour vous montrer ce qui se passe. Un nœud actif est marqué avec un pointeur en forme de flèche **->** qui indique quel est le nœud est en exécution. Vous voyez également des messages provenant du SUT client dans la zone du Terminal.

Lorsque la séquence **Setup** est effectuée, notre application de démonstration **Options Demo** apparaît à votre écran. C'est le SUT, entièrement sous le contrôle de l'œil vigilant de *qftestJUI*. Si vous ne voyez pas **Options Demo** à votre écran, merci de vérifier si cette fenêtre n'est pas cachée derrière la fenêtre principale de *qftestJUI*. Cela peut se produire quand la fenêtre de *qftestJUI* est maximisée.

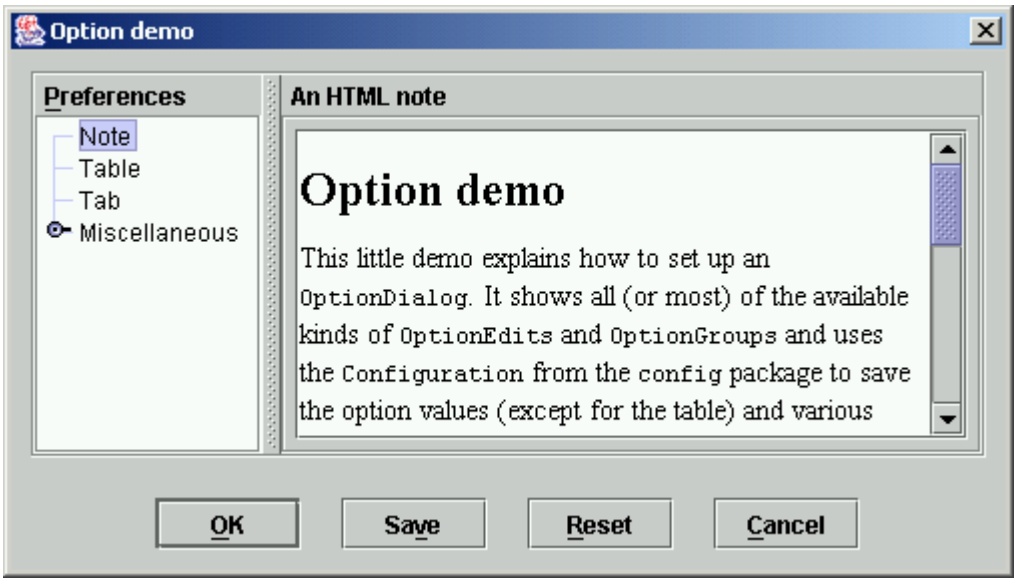

Figure 2.5 - Options Demo

Ceci est un bref résumé de la séquence de démarrage dans le nœud Setup :

- **Start Java SUT client** démarre l'application.
- **Wait for the client to connect** attend que la Machine Virtuelle Java client se connecte à *qftestJUI*, ou que la temporisation soit atteinte.
- **Wait for component** attend que la fenêtre principale du SUT apparaisse. C'est la dernière étape de la séquence de démarrage et cela permet de s'assurer que le SUT est prêt à être testé.

Remarque : Si Options Demo n'est pas visible à votre écran ou apparaît pendant un court moment, c'est que cette fenêtre est probablement couverte (cachée) par la fenêtre principale de *qftestJUI*. La meilleure solution consiste à placer ces fenêtres côte à côte de manière à pouvoir toutes les voir en même temps.

2.3 Test Flux de Saisie

La tâche suivante consiste à jeter un premier coup d'œil sur la façon dont un test pour le SUT est structuré dans *qftestJUI*. Le nœud test **Clickstream** (*Flux de Saisie*) contient plusieurs séquences d'opérations qui exécutent des fonctions dans le SUT, telles que les clics souris et de la saisie de champs textes. Au travers de ce test, nous illustrons plusieurs fonctionnalités simples, mais très importantes de *qftestJUI*.

Lorsque vous ouvrez le nœud test **Clickstream**, vous voyez les deux séquences comme ci-dessous :

Figure 2.6 - Nœud-Test Clickstream

Pour exécuter le test, sélectionnez le nœud **Test: Clickstream** et cliquez sur le bouton de rejeu  . Les séquences de test seront alors exécutées directement dans le SUT, comme vous pouvez le voir pendant l'exécution.

Pour vous aider à comprendre ce que vous venez de voir, jetons un coup d'œil à la séquence : **Table.**

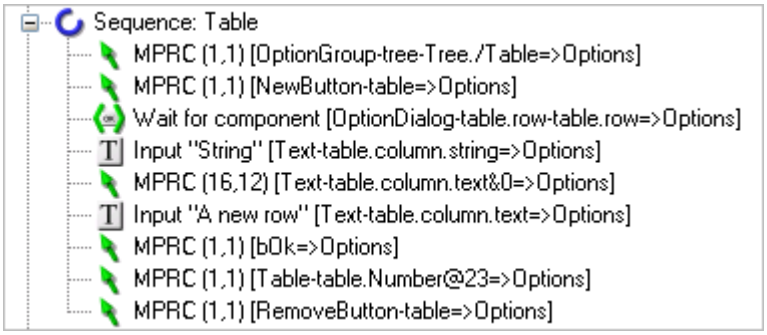

Figure 2.7 - Séquence Test Table

Ici nous voyons que la séquence démarre avec plusieurs clics souris, indiqués par les étiquettes des nœuds avec l'acronyme **MPRC**. *Moved-Pressed-Released-Clicked* est un résumé des événements réels qui se produisent quand vous cliquez sur le bouton de la souris. A côté de l'étiquette MPRC vous voyez les coordonnées réelles utilisées pour implémenter le clic souris relatif au composant indiqué entre crochets après les coordonnées. Le premier nœud **MPRC** est, par exemple, un clic sur l'élément de la structure arborescente du SUT ayant pour étiquette **Table**.

Après les quelques premiers clics souris dans la séquence, l'exécution du nœud **Wait for component** provoque une attente jusqu'à l'apparition de la boîte de dialogue d'édition du SUT. Une fois que le dialogue apparaît, l'exécution de la séquence continue, dans laquelle le nœud **Input** est activé afin de saisir une chaîne de caractères dans un champ du dialogue. Les

nœuds restants de la séquence ont des fonctions similaires à celles mentionnées ci-dessus et devraient vous sembler plus clairs à partir de leurs descriptions.

Si vous êtes curieux, ouvrez également l'autre séquence (**Tab**). Dans cette séquence, il y a des nœuds similaires à ceux que vous avez vu dans **Table** avec, en plus, un nœud utilisé pour le contrôle du clavier.

2.4 Quelques Astuces

Nous faisons une courte pause ici pour vous donner quelques astuces qui s'avèreront utiles pour la suite du didacticiel.

Lorsque vous travaillez avec de nombreux nœuds dans *qftestJU*I vous pouvez, par exemple, avoir besoin d'aide pour vous rappeler la raison pour laquelle ils sont utilisés. Pour répondre à ce besoin, *qftestJUI* propose une aide contextuelle, accessible par un simple clic souris. Déplacez le pointeur de la souris sur un élément pour lequel vous voulez de l'aide, et faîtes un clic droit. Dans le menu contextuel qui s'affiche, vous voyez une entrée **"What's this?"** En sélectionnant cette option, la section appropriée du Manuel Utilisateur de *qftestJUI* s'affiche alors dans votre navigateur Web par défaut.

Nous vous précisons également qu'il existe une version du Manuel Utilisateur au format PDF, disponible dans le répertoire d'installation de *qftestJUI*.

2.5 Contrôle de Texte

Un des concepts les plus importants dans *qftestJUI* est celui du contrôle, c'est-à-dire une requête de certains éléments dans le SUT. Un **Contrôle de Texte** (*Text Check*) interroge sur l'existence ou l'absence de texte dans un composant du SUT, tel qu'un champ texte.

Par exemple, faisons un contrôle sur un champ dans le client SUT. Affichez la fenêtre **Options Demo** et cliquez sur le nœud **Miscellaneous** listé dans la structure arborescente pour que le nœud s'ouvre et affiche ses nœuds fils. Parmi les nœuds fils, vous voyez le nœud **Numbers**. Cliquez sur ce nœud pour afficher sur le côté une nouvelle fenêtre qui inclut le champ texte portant l'étiquette **May be negative**. Pour notre exemple, nous allons faire un contrôle sur ce champ.

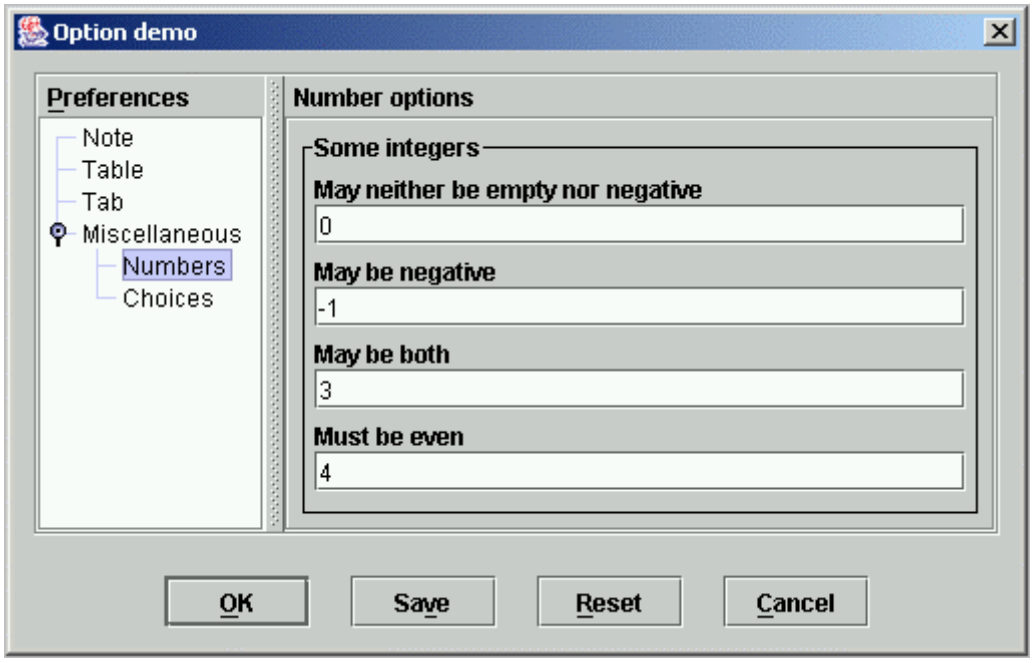

Figure 2.8 - Champ Texte à Contrôler

Maintenant retournez à la suite de test et ouvrez le nœud **Text check**. A l'intérieur, vous voyez la séquence de test suivante :

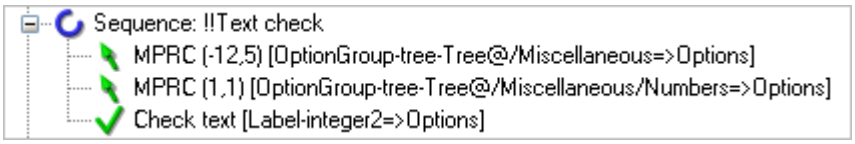

Figure 2.9 - Séquence Text Check

Vous devez vous demander pourquoi il y a deux points d'exclamation placés devant le nom du nœud de la séquence **Text check**. Ces marques permettent de marquer les résultats, une fois exécutés, à des fins de documentation. Ces résultats finiront dans un rapport automatiquement créé au format HTML. Ce sujet sera couvert dans un autre chapitre.

Une fois le nœud séquence **Check text** sélectionné, cliquez sur le bouton de rejeu pour l'exécuter. Une fois la séquence terminée, une boîte de dialogue apparaît pour indiquer qu'une erreur s'est produite :

Figure 2.10 - Erreur dans le Nœud Check text

Que s'est-il passé ? Quand de tels incidents se produisent, l'utilisation du run-log de *qftestJUI* peut s'avérer très utile pour le diagnostic. Sélectionnez le menu **Run -> Run log** pour afficher le run-log. Une nouvelle fenêtre apparaît alors affichant l'enregistrement du contenu de la séquence que vous venez d'exécuter :

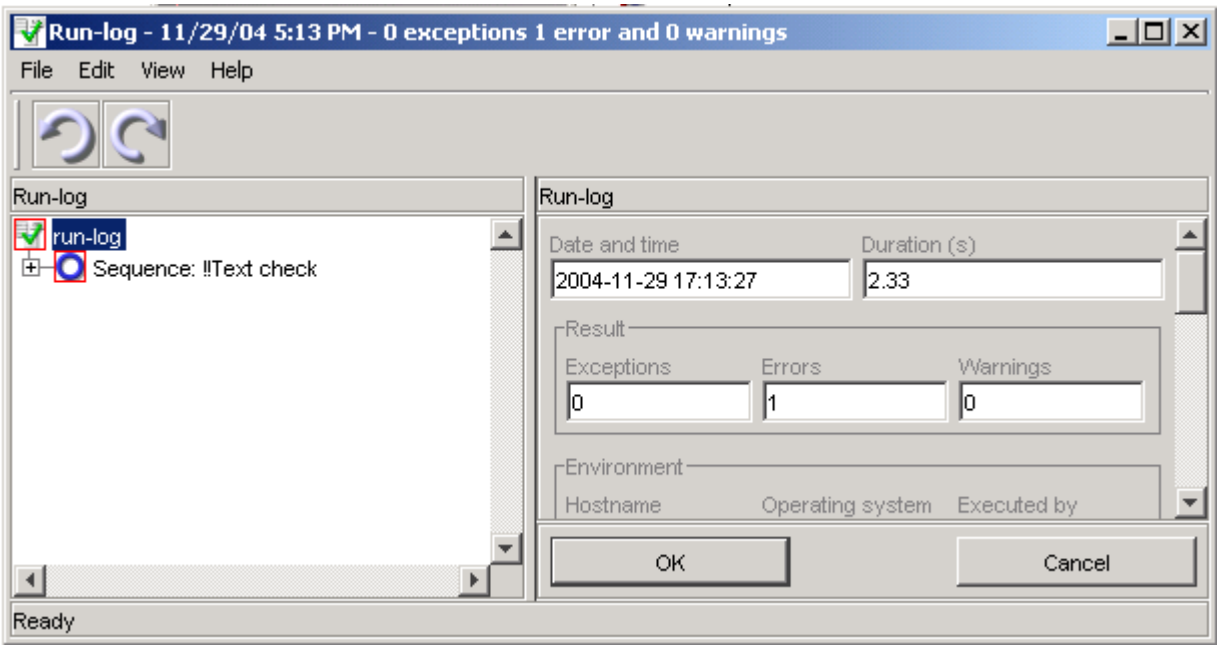

Figure 2.11 - Run-Log pour la Séquence Text check

Le run-log est semblable en structure à l'arbre de la vue Suite de Test avec laquelle vous vous êtes familiarisé(e). L'arbre sur le côté gauche représente les événements journalisés pour l'exécution du test, avec le premier événement en haut et le dernier en bas. Quand vous cliquez sur l'un des nœuds événements sur le côté gauche, les propriétés de l'événement s'affichent sur la droite.

Alors que vous ouvrez les nœuds dans la structure arborescente sur la gauche, vous remarquez immédiatement que certains nœuds sont entourés par un cadre rouge. Comme vous vous en doutez, c'est une indication de *qftestJUI* concernant un problème s'étant produit dans un nœud fils. Si vous gardez ouverts les nœuds rouges, vous arriverez finalement au nœud qui pose problème. Un moyen plus facile pour trouver une erreur consiste à utiliser simplement le menu **Edit -> Find next error** à partir du menu run-log. Cela vous amènera à :

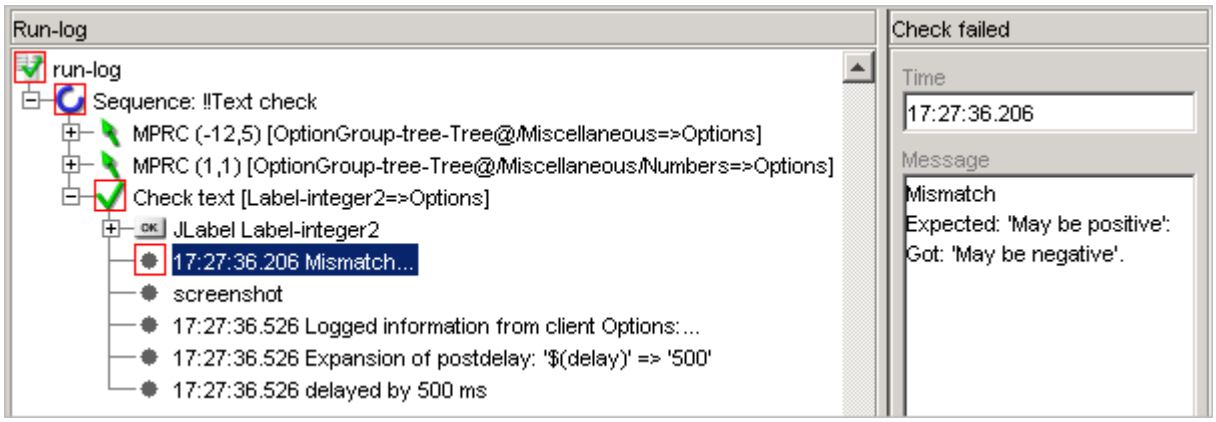

Figure 2.12 - Diagnostic de l'Erreur dans la Séquence Check text

Lorsque vous cliquez sur le nœud rouge, vous voyez dans les détails de ce nœud qu'il y a une différence entre le texte attendu dans notre champ SUT et celui trouvé. L'étape suivante consiste à aller au nœud correspondant dans la suite de test et à le corriger. Assurez-vous que le nœud approprié est sélectionné dans le run-log, et utilisez le menu **Edit -> Find node in test-suite** ou tapez le raccourci clavier **Ctrl+T**.

Un autre dispositif utile pour le diagnostic d'erreur est l'arbre de nœud screenshot dans le run-log. Ses détails incluent une capture d'écran complète prise au moment où l'erreur s'est produite. Etre en mesure de voir l'état du SUT à ce moment est d'une aide très précieuse pour déterminer la cause de l'erreur. La figure 2.13 montre une capture d'écran d'un nœud avec situation de l'erreur.

Remarque : Les informations réunies lors de l'exécution d'un test long peuvent être gourmandes en mémoire. Par conséquent *qftestJUI* est configuré pour créer un run-log compact. Seuls les 100 derniers nœuds de protocole sont conservés, le reste est jeté, excepté pour les informations pertinentes pour la génération de rapports et pour le diagnostic d'erreur qui sont toujours conservées. Cette fonctionnalité est configurable via l'option **Create compact run-log** accessible par le menu **Edit -> Options -> Run-logs -> Content**. Le type de run-log est également montré dans son nœud racine.

Exercice de base : retournez dans votre suite de test et modifiez le texte attendu dans le nœud **Check text**. Maintenant si vous exécutez la séquence, aucune erreur ne se produira.

2.6 Contrôle d'un Bouton Radio

La troisième séquence de test **Selected test** réalise des requêtes de l'état d'un bouton radio spécifique dans le SUT pour déterminer si ce bouton radio est sélectionné. Dans la section **Choices** de la zone **Miscellaneous** dans l'application **Options**

Demo, vous pouvez sélectionner interactivement n'importe lequel des quatre boutons radio. Nous utiliserons ces boutons radio pour implémenter un test qui inclut un contrôle réussi, ainsi qu'un contrôle qui génère une erreur.

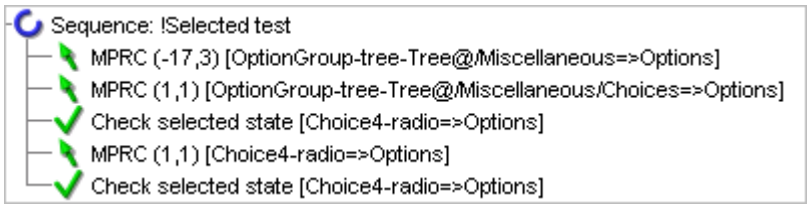

Figure 2.14 - Nœud Selected test

Le nœud **Selected test** contient trois événements MPRC et deux contrôles. Les deux premiers MPRC permettent aux nœuds dans la structure arborescente du SUT de s'ouvrir jusqu'à ce que le nœud **Choices** soit visible et ses propriétés affichées sur la droite. Ensuite, vous voyez un nœud de contrôle **Check selected state** qui interroge l'état du quatrième bouton radio pour déterminer qu'il n'est pas sélectionné. Le troisième événement MPRC effectue un clic sur ce quatrième bouton radio qui est maintenant sélectionné. Finalement, un autre contrôle **Check selected state** est effectué. Cependant, maintenant que le bouton radio est sélectionné, vous voyez que ce dernier contrôle génère une erreur.

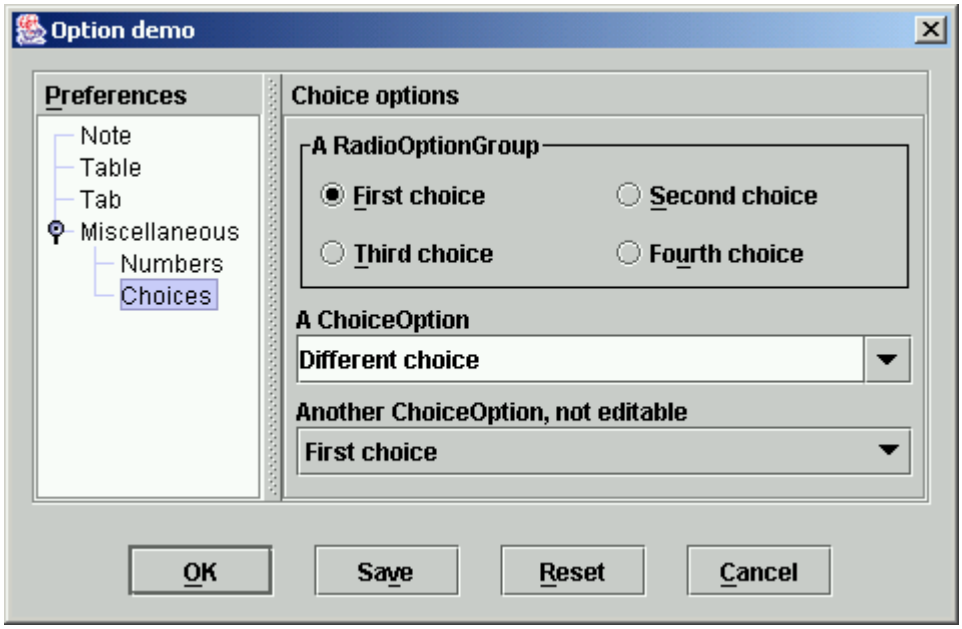

Figure 2.15 - Boutons Radio dans Options Demo

Essayez maintenant d'exécuter la séquence Selected test afin de voir l'exécution réelle de la description fournie. Après l'exécution de la séquence test, vous pouvez vouloir fermer les nœuds dans le SUT, de sorte qu'il soit de retour à son état original (nœud **Miscellaneous** plus visible).

2.7 Arrêt de l'Application

La fonction de la séquence **Cleanup** (épuration / suppression) est importante à comprendre. Un nœud **Cleanup** sera exécuté après chaque test ou séquence localisé au même niveau de la structure arborescente, tout comme la séquence **Setup** sera exécutée avant chaque test ou séquence. Le nœud **Cleanup** permet de laisser le SUT dans un état spécifique ou "propre'" afin que des tests ultérieurs puissent être exécutés de manière déterministe.

Cependant veuillez noter que le nœud **Cleanup**, tout comme le nœud **Setup**, sera seulement exécuté lorsque le nœud de plus haut niveau est exécuté, ou lorsque vous l'exécutez explicitement. Dans les sections précédentes, vous avez sélectionné et effectué un test ou une séquence spécifique (comme par exemple **Clickstream**), qui n'entraîne pas l'exécution des nœuds **Setup** ou **Cleanup**. Dans la section qui suit, vous allez voir visuellement comment cela fonctionne quand vous exécutez le nœud de haut niveau **Test: Options**.

Ouvrez maintenant le nœud **Cleanup** pour voir ses nœuds fils. Pour ce nœud, nous essayons d'arrêter et de fermer l'application SUT d'une manière ordonnée en cliquant le bouton **Cancel** du SUT. Au cas où le résultat attendu n'est pas obtenu (le SUT ne se ferme pas), une tentative plus agressive est alors faîte.

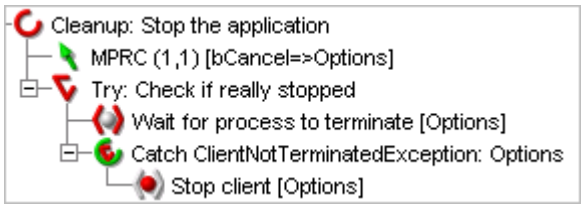

Figure 2.16 - La Séquence Cleanup

Notre séquence **Cleanup** est organisée comme suit :

- Le premier nœud **MPRC** clique le bouton **Cancel** du SUT.
- Les nœuds restants sont construits autour d'une construction try/catch, qui est utilisée pour gérer les cas erronés du SUT en échec qui s'arrête quand **Cancel** est cliqué. Le premier nœud dans le bloc try/catch effectue un contrôle **Wait for client to terminate**.
- Dans le cas où le client SUT échoue pour terminer dans la période de temporisation donnée, une exception est générée - **ClientNotTerminatedException**, qui sera alors capturée dans le nœud Catch.
- Si l'exception est en effet générée, un nœud **Stop client** est alors exécuté sous le nœud Catch. Le nœud **Stop client** a pour effet de terminer le processus en cours exécutant l'application SUT.

Ce que nous avons mis en application est un nœud **cleanup** "défensif" qui essaie en premier de stopper l'application normalement, et seulement si cet échec arrête le processus.

2.8 Test Complet

Les sections précédentes ont eu pour objectif de vous amener étape par étape aux différents éléments de notre suite de test exemple. Désormais, vous êtes prêt(e) à exécuter le test entier d'un seul trait.

Premièrement, fermez **Options Demo** au cas où il serait toujours en exécution. Vous pouvez le faire en exécutant le nœud **Cleanup** décrit précédemment. Alors sélectionnez le nœud **Test: Options** et exécutez-le avec le bouton de rejeu. La suite de test complète durera quelques minutes, en raison des délais que nous avons construits dans certains nœuds de la suite de test, aussi vous pouvez mieux suivre ce qui se passe. Si vous regardez les détails du nœud **Test: Options**, vous verrez une variable **delay** définie dans la section **Variables**. Vous pouvez changer cette valeur si vous voulez réduire ou augmenter le temps total d'exécution.

Quand les tests sont achevés, vous devez finir avec deux échecs, les deux dont nous avons parlé dans les sections précédentes. Ouvrez à nouveau le run-log pour l'étudier :

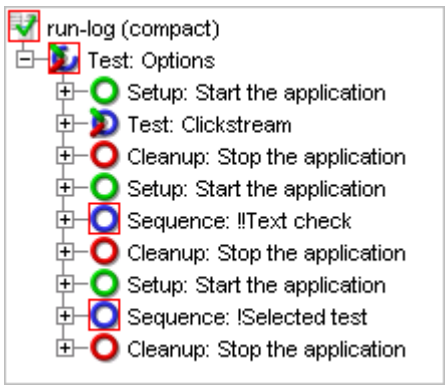

Figure 2.17 - Run-Log pour les Tests Exécutés

Ici vous verrez le processus que nous avons essayé d'expliquer dans la section précédente concernant l'exécution des nœuds **Setup** et **Cleanup** avant et après chaque test et séquence de manière explicite.

2.9 Génération de Rapports

Dans le domaine de l'Assurance Qualité, la documentation et l'archivage de tests sont d'une très grande importance. A cet effet, *qftestJUI* offre une fonctionnalité de génération automatique de rapports. Maintenant que vous avez effectué une exécution d'un test complet, c'est le moment de vous montrer ces possibilités.

Assurez-vous que le run-log de votre test est ouvert. Maintenant utilisez l'option **File -> Create HTML/XML report** du menu **run-log** pour afficher la fenêtre de dialogue utilisée pour spécifier la nature du rapport voulu.

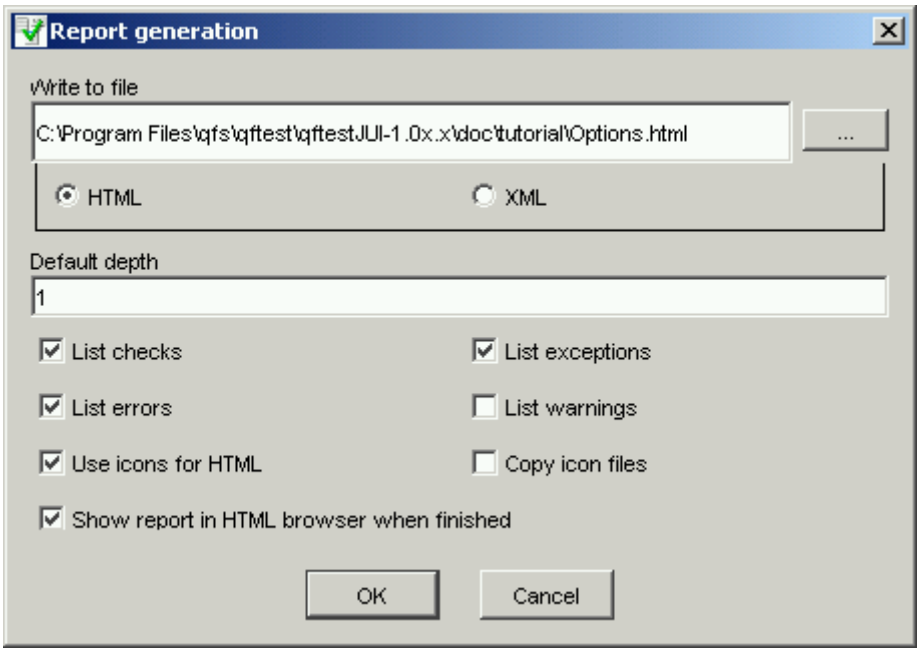

Figure 2.18 - Propriétés de Génération de Rapport

Dans le premier champ, vous pouvez spécifier le nom du fichier du rapport. Ensuite, vous décidez du type de rapport que vous voulez. *qftestJUI* offre deux types de rapports : format HTML ou XML. Un rapport XML est utile si vous avez écrit vos propres feuilles de styles XSLT, pour mettre en forme le rapport dans le style que vous préférez.

Le champ **Default depth** spécifie la profondeur par défaut de la génération de rapports. C'est une nomenclature qui décrit simplement la profondeur de la structure arborescente de la suite de test que vous voulez dans votre rapport. Avec une profondeur par défaut à '**1**' votre rapport contiendra uniquement les séquences et les tests de la suite de test du niveau le plus élevé, et ne contiendra donc pas les nœuds fils sous-jacents, tels que les événements MPRC.

Nous devons nous référer aux points d'exclamation ("**!**") utilisés dans les sections précédentes. Ils sont utilisés pour marquer spécifiquement le nœud en question à des fins de documentations. Un nœud marqué avec un point d'exclamation de cette manière sera toujours inclus dans le rapport généré, indépendamment de la profondeur par défaut spécifié.

Essayez de générer un rapport HTML simple à partir des résultats du dernier test exécuté. Remplissez les champs que nous avons mentionné ci-dessus et confirmez la génération en cliquant sur le bouton **OK**. Le rapport sera alors généré et apparaîtra dans votre navigateur comme le montre la figure 2.19.

Le rapport commence avec une référence aux données du run-log associé, suivie d'une série d'informations sur votre système. Ensuite vous voyez les résultats du test, dont l'exécution totale, tout comme le comptage des erreurs et des exceptions qui se sont produites. Dans la figure 2.19, vous pouvez voir que deux erreurs se sont produites dans le test.

Après ce résumé, vous voyez les détails spécifiques aux nœuds de test et de séquence qui ont été exécutés. Cela vous aide à déterminer exactement où une erreur spécifique s'est produite. A la fin du rapport, vous voyez une courte description des erreurs rencontrées.

Remarque : Les icônes arbre-nœud au sein du rapport généré sont affichées pour une meilleure orientation et compréhension des relations entre le rapport et la suite de test. Mais elles n'ont aucune autre fonction, par exemple ouvrir/fermer les nœuds de l'arbre.

Le dispositif de génération de rapports s'avèrera très utile au fur et à mesure du développement de votre suite de test. Essayez d'ajuster certains des paramètres de la fenêtre de dialogue de génération de rapport pour obtenir le niveau et la profondeur des détails que vous souhaitez afficher.

CHAPITRE 3 - CREATION D'UNE SUITE DE TEST

3.1 Introduction

Ce chapitre décrit les différentes étapes nécessaires pour vous guider dans la création de votre première suite de test. L'application à tester est **FileChooserDemo** incluse dans le kit de développement logiciel Java. Cela est également inclus dans la distribution de qftestJUI avec le code source, dans le répertoire **qftestJUI-1.08.4/doc/tutorial/FileChooserDemo**

L'application **FileChooserDemo** est un très bon exemple sur la manière de personnaliser la classe **FileChooser** dans Swing, et de créer vos propres fichiers de dialogue Open, **Save** et personnalisés pour votre application. Nous l'utilisons comme le SUT (*System Under Test*) pour la création de votre propre suite de test.

Ce chapitre du didacticiel a été conçu de sorte à illustrer les concepts suivants :

- Démarrage d'un SUT dans qftestJUI
- Enregistrement de composants et organisation en des tests simples
- Création de tests vérifiant les dépendances des fonctions dans le SUT

3.2 Démarrage de l'Application

Ouvrez une nouvelle (vide) suite de test dans qftestJUI avec le menu **File -> New Suite**. Si vous venez juste de démarrer *qftestJUI*, vous verrez que vous avez déjà une suite de test vide sous vos yeux.

Assurez-vous que la vue détaillée est activée (menu **View -> Show Details**). Vous voyez maintenant la fenêtre divisée en deux avec un arbre du côté gauche et un volet du côté droit, représentant les détails du nœud sélectionné. L'arbre vous permet de naviguer parmi les nœuds de la suite de test ; lorsque vous sélectionnez un nœud, ses propriétés s'affichent sur le volet de droite.

Pour commencer vous devez démarrer votre application. Comme mentionné précédemment, l'application à tester est **FileChooserDemo** du kit de développement logiciel Java.

Pour cet exemple, nous commençons par insérer des éléments dans le nœud **Extras** de la suite de test. Cette zone est une sorte de bloc-notes dans laquelle nous pouvons expérimenter certains concepts avant de les placer dans la structure correcte de la suite de test. Quand vous ouvrez la branche du nœud **Extras**, vous voyez apparaître une fine ligne rouge juste après. Nous nous y référons comme un "marqueur d'insertion". Le but étant de vous faire savoir où le prochain nœud sera inséré si vous sélectionnez un élément depuis le menu **Insert**.

Remarque : Avant que vous n'insériez réellement le nœud pour démarrer le SUT, voici quelques explications sur le comportement de *qftestJUI* lors de l'insertion de nœuds à la position du marqueur d'insertion. En fait, *qftestJUI* vous laisse uniquement choisir des nœuds qui sont "autorisés" à être insérés à la position courante du marqueur. Aussi si par exemple vous sélectionnez le nœud racine de votre suite de test (en laissant la branche du nœud ouverte) et vous essayez d'insérer un nœud via le menu **Insert**, tous les choix sont désactivés mais seul le nœud **Test** de **Sequences nodes** est sélectionnable. Il y a d'autres restrictions avec d'autres types de nœuds (merci de vous référer au ***Manuel Utilisateur***). Parfois vous serez surpris de voir qu'un nœud attendu ne peut pas être inséré, mais dans la plupart des cas cela est dû à une mauvaise position du marqueur d'insertion, parce que vous n'avez pas ouvert la branche du nœud parent pour insérer le nouveau nœud comme fils.

Pour démarrer l'application, utilisez le nœud **Start Java SUT client**. La capture d'écran suivante vous montre comment utiliser les menus contextuels, accessibles par un clic souris droit, pour insérer le nœud.

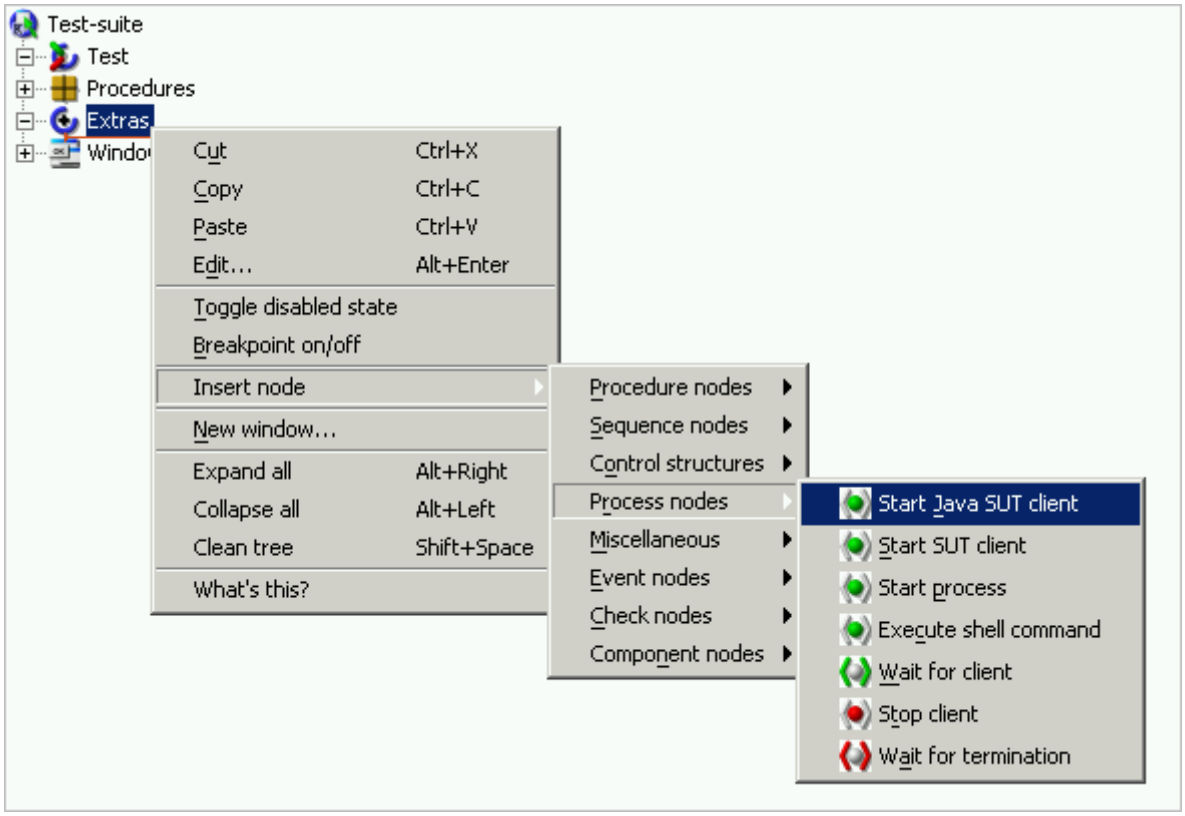

Figure 3.1 - Insertion du Nœud Start Java SUT Client

Après la sélection du nœud à insérer, une fenêtre de dialogue s'affiche, vous permettant de renseigner les propriétés du nœud :

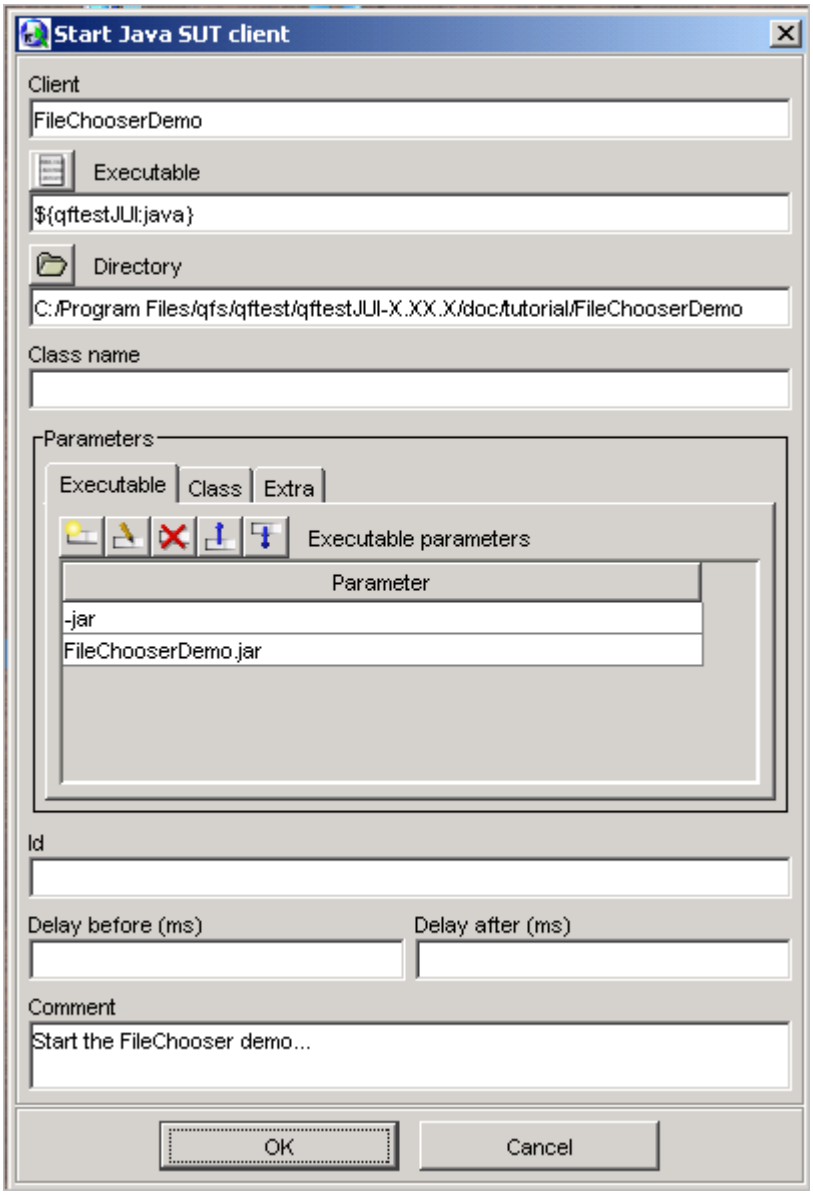

Figure 3.2 - Propriétés du Nœud Start Java SUT Client

Les propriétés pour ce nœud sont décrites ci-après :

- Dans le champ **Client** vous entrez le nom pour le SUT qui sera utilisé comme référence à l'application par qftestJUI. Vous pouvez donner au SUT le nom que vous voulez, mais le même nom sera utilisé pour référencer le SUT dans d'autres nœuds, aussi il vous faut constamment utiliser le nom choisi.
- Le second champ spécifie le programme exécutable qui sera utilisé pour implémenter le SUT. Pour notre exemple, la variable **${qftestJUI:java}** a été donnée, qui déclare de façon absolue que la Machine Virtuelle Java utilisée pour démarrer *qftestJUI* est la même que celle utilisée pour démarrer le SUT.
- Le champ **Directory** détermine le répertoire personnel dans lequel le SUT sera démarré. Vous pouvez changer le répertoire en utilisant le bouton de sélection de répertoire (**Select directory**)  ou en le saisissant manuellement.
- Le quatrième champ vous présente la classe de démarrage de votre application Java. Pour notre exemple, ce champ ne nécessite pas d'être renseigné, dans la mesure où le programme est dans une archive jar.
- Le démarrage de notre application sera implémenté avec la commande Java jar **FileChooserDemo.jar**. Aussi nous avons simplement besoin d'entrer cette commande comme un élément dans le champ **Parameters**. Toute entrée dans la table **Parameters** sera passée individuellement dans Java lors de l'exécution, aussi nous devons séparer les entrées. Les entrées peuvent être faîtes avec le bouton d'ajout de nouvelle ligne (**Add new row**)  .
- Les champs restants de la boîte de dialogue des propriétés ne sont pas requis pour cet exemple. Le champ **Comment** en bas de la boîte de dialogue est utile, mais toujours facultatif.

Cliquez sur **OK** pour confirmer vos saisies dans la boîte de dialogue. Si tout marche correctement, vous voyez alors le nouveau nœud apparaître après le marqueur d'insertion dans la suite de test. Cliquez sur ce nouveau nœud (sélectionnez-le) et exécutez-le avec le bouton de rejeu  . Vous verrez ensuite l'application **FileChooserDemo** apparaître sur votre écran :

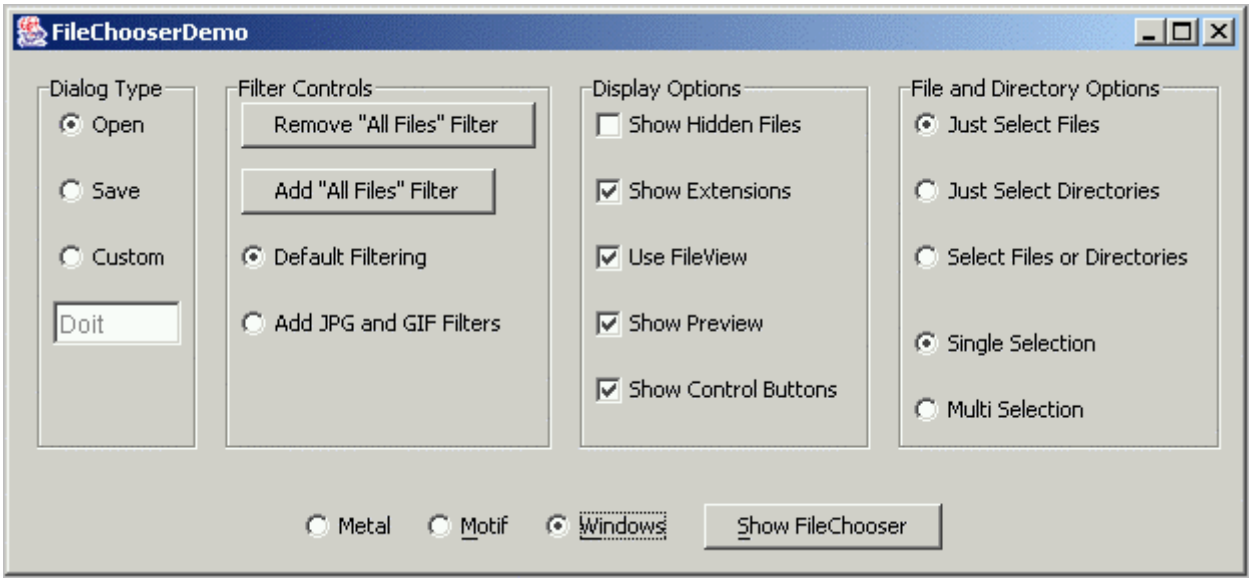

Figure 3.3 - Fenêtre FileChooserDemo

Si l'application **FileChooserDemo** ne s'affiche pas au bout de quelques secondes, revenez aux étapes précédentes pour vous assurer que votre nœud **Start Java SUT client** a été correctement renseigné. Vous pouvez également jeter un coup d'œil au Terminal, qui fournit des messages utiles au cas où quelque chose ne démarrerait pas.

3.3 Ajout d'un Test Flux de Saisie

Maintenant vous êtes prêt(e) à ajouter un **Test Clickstream** simple. Appuyez sur le bouton d'enregistrement (*Record*)  et passez sur la fenêtre de l'application du SUT. A partir de maintenant chaque action souris et clavier effectuée dans la fenêtre du SUT sera enregistrée. Cliquez sur différents boutons et revenez à la fenêtre de la suite de test dans *qftestJUI*. Appuyez sur le bouton d'arrêt  . Vous retrouverez la séquence enregistrée placée sous le nœud **Extras** sur le côté gauche de la fenêtre principale, comme montré ci-dessous :

Figure 3.4 - Flux de Saisie enregistré

Le nom de la séquence enregistrée sera entré de manière standard par *qftestJUI* tout comme le temps et la date d'enregistrement. Vous pouvez changer ce nom en cliquant simplement sur le nœud et en changeant ses propriétés dans la vue détaillée.

Maintenant sélectionnez votre séquence enregistrée et exécutez-la avec le bouton de rejeu  . Vous voyez maintenant votre séquence exacte d'événements souris et clavier rejouée dans la fenêtre du SUT.

Ce type de test peut être tout à fait spectaculaire et utilisé pour des démonstrations de votre application, mais il teste surtout l'implémentation sous-jacente Swing, et non pas la logique applicative. Dans les sections suivantes nous implémenterons des tests qui vérifient le contenu des champs et vérifient aussi les connexions causales dans le **FileChooserDemo**, mais tout d'abord nous allons structurer les séquences enregistrées dans une suite de test.

3.4 Construction d'une Suite de Test

La structure de base d'une suite de test est composée des nœuds suivants :

- Les nœuds test racine contiennent un nombre arbitraire de nœuds séquences et tests.
- **Procedures** contient des séquences réutilisables qui peuvent être organisées en fonctions modulaires.
- **Extras** est le bloc-notes pour les expérimentations.
- **Windows and components** contiennent tous les éléments importants de la suite de test : les éléments enregistrés du SUT tels que les fenêtres, les menus et les boutons. Chaque élément au sein de la section Windows and components contient les propriétés de l'élément, pour que *qftestJUI* puisse facilement trouver le composant lors du rejeu d'un test ou d'une séquence.

Il y a quatre nœuds différents utilisés pour organiser une séquence : **Test**, **Setup**, **Cleanup**, et finalement le nœud général **Sequence**. Les tests sont implémentés avec le nœud test, les séquences spéciales **Setup** et **Cleanup** contiennent les étapes importantes pour s'assurer que le SUT s'exécute dans un état bien défini. Une mise en application courante de la paire **Setup/Cleanup** est de démarrer et d'arrêter l'application SUT, ainsi le SUT est toujours dans un état connu quand un test s'exécute. La paire **Setup/Cleanup** peut toutefois être utilisée à n'importe quel niveau et pour n'importe quel but jugé utile, comme ouvrir et fermer un dialogue par exemple.

Commençons par ajouter une séquence **Setup** à notre nœud test vide de premier niveau avec le menu **Insert -> Sequence nodes -> Setup**. La seconde étape consiste à déplacer le nœud **Start Java SUT client** depuis le nœud Extras (voir section précédente) et à le coller dans la séquence **Setup**.

Afin de déplacer ce nœud depuis Setup, vous devez tout d'abord vous assurer que le nœud **Setup** est ouvert (regardez à nouveau l'emplacement du marqueur d'insertion). Le déplacement du nœud **Start Java SUT client** peut désormais être effectué avec le menu contextuel (clic souris droit) ou avec les touches clavier **Ctrl+X** (couper) et **Ctrl+V** (coller).

Une fois fait, ajoutez un nœud **Wait for client to connect** avec le menu **Insert -> Process nodes -> Wait for client**. Ce nœud fait patienter *qftestJUI* le temps que le client SUT démarre et s'assure de la connexion au SUT.

Nous avons besoin d'une chose supplémentaire pour que le nœud **Setup** soit complet. Après que le SUT se soit connecté à *qftestJUI*, cela peut prendre un peu de temps avant que la première fenêtre ne s'affiche à l'écran. Nous devons nous assurer que le test ne démarre pas avant, aussi nous devons attendre que cette première fenêtre apparaisse.

A cette fin, insérez un nœud **Wait for component** après le nœud **Wait for client** avec le menu **Insert -> Miscellaneous -> Wait for component**. Si le SUT s'exécute, l'attribut **Client** doit être rempli avec le nom du client SUT, **FileChooserDemo** dans notre cas. Pour paramétrer l'**ID du Composant**, cliquez sur le bouton  au-dessus du champ pour afficher une boîte de dialogue avec les nœuds **Windows and components** disponibles. Sélectionnez le nœud **Window** ayant pour étiquette **JFrame frameFileChooserDemo.**

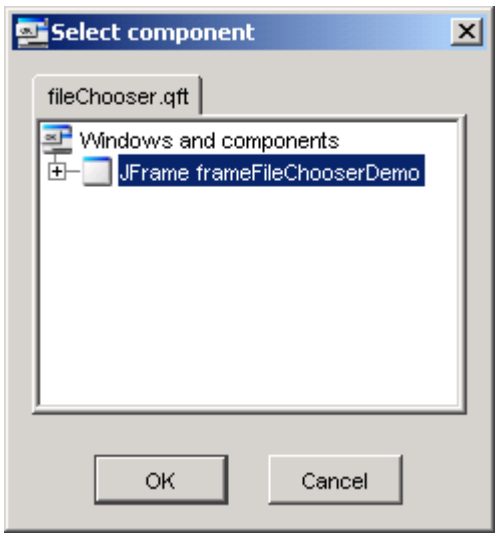

Figure 3.5 - Dialogue de Sélection de Composant

Maintenant vous pouvez arrêter l'application **FileChooserDemo** et la relancer en exécutant le nœud **Setup**. Cela devrait vous donner :

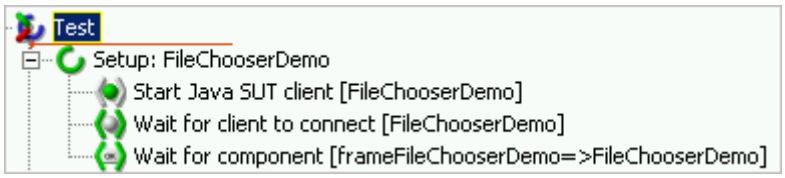

Figure 3.6 - Séquence Setup de FileChooserDemo

L'étape suivante consiste à créer le nœud **Sequence** pour notre flux de saisie (**clickstream**). Vous devez fermer les branches du nœud **Setup** pour que le marqueur d'insertion soit placé après **Setup**, et non dedans. Alors vous pouvez déplacer votre séquence enregistrée du nœud **Extras** vers le marqueur d'insertion pour que le nœud **Sequence** apparaisse directement après la séquence **Setup**.

En dernier lieu vous pouvez créer une séquence **Cleanup** pour arrêter l'application. Cette séquence contient deux nœuds : un pour arrêter le client, et un autre pour s'assurer qu'il est bien terminé. Sur la figure ci-dessous, vous pouvez voir quels nœuds sont nécessairement mis en exécution pour cet exercice. Votre suite de test doit ressembler à ce qui suit :

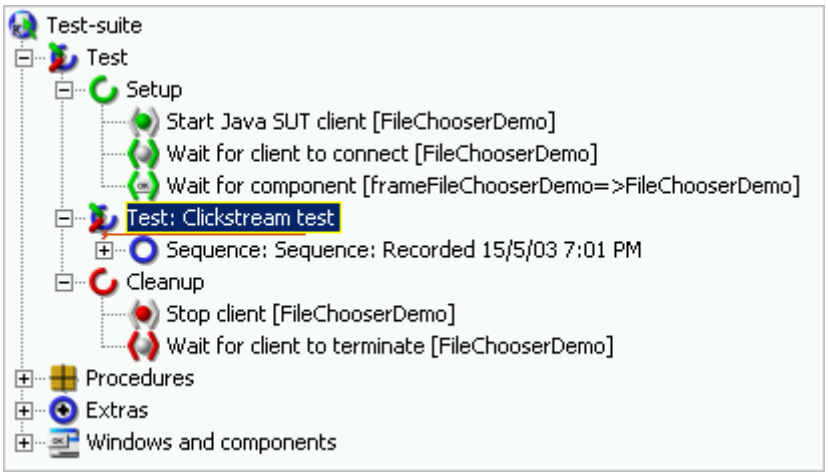

Figure 3.7 - Organisation de votre Suite de Test

Vous venez d'effectuer les étapes les plus importantes de structuration de votre suite de test. Dans la section suivante, vous développerez votre suite de test en introduisant un contrôle sur un champ texte spécifique.

3.5 Ajout d'un Contrôle de Texte

Pour surveiller le comportement du client, nous utilisons des nœuds de contrôle, qui interrogent certains états et propriétés d'éléments dans le SUT. Le premier contrôle que nous introduirons est le **Text-check** (*Contrôle de texte*), qui vérifie qu'un champ texte contient la chaîne de caractère attendue. Notre contrôle sera effectué sur un bouton dans la fenêtre **FileChooser**.

Si le SUT n'est pas en exécution, démarrez-le maintenant en utilisant notre séquence **Setup**. Au sein de la fenêtre **FileChooserDemo**, cliquez sur le bouton **Show FileChooser**.

Pour commencer l'enregistrement d'un contrôle, cliquez le bouton d'enregistrement de contrôle (*Record check*) et passez sur la fenêtre **FileChooser**. Maintenant lorsque vous déplacez votre souris sur les composants, leur couleur est intervertie. Pour enregistrer le contrôle, allez au bouton **Open**, et faîtes un clic droit. Un menu contextuel s'affiche alors et vous donne le choix entre trois contrôles différents : **Text**, **Enabled state** et **Geometry**, comme l'illustre la figure ci-dessous :

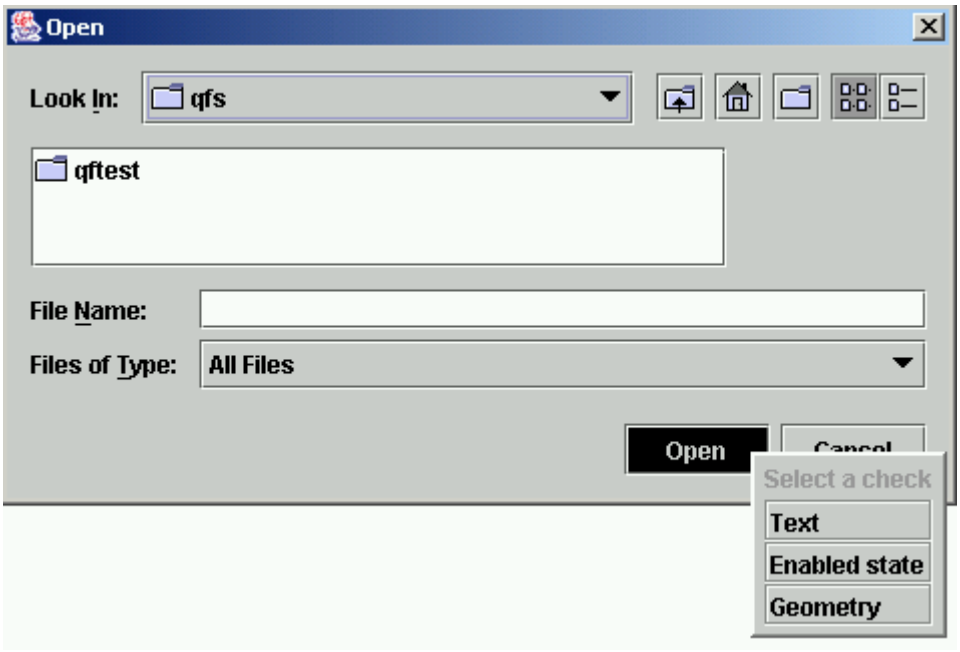

Figure 3.8 - Enregistrement d'un Contrôle dans la Fenêtre FileChooser

Choisissez **Text** dans le menu contextuel, retournez à la fenêtre de votre suite de test, et arrêtez l'enregistrement en cliquant sur le bouton d'arrêt .

Ensuite, enregistrez une autre séquence flux de saisie arbitraire en utilisant le bouton , puis refermez la fenêtre **FileChooser**.

Maintenant vous savez comment organiser les séquences enregistrées dans votre test. Vous pouvez comparer vos résultats avec la manière dont nous avons organisé le test :

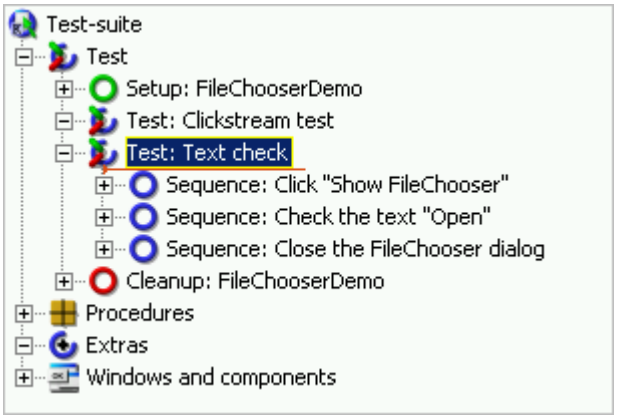

Figure 3.9 - Organisation du Test Text-Check dans l'Arbre

A ce stade, vous pouvez donner à votre test une exécution préliminaire. Arrêtez le client SUT s'il est exécution, cliquez sur le nœud test racine, et démarrez l'exécution en appuyant sur le bouton de rejeu  .

Le résultat de l'exécution du test est alors stocké dans le run-log. Pour accéder au run-log, sélectionnez le menu **Run -> Run log** ou utilisez le raccourci clavier **Ctrl+L**.

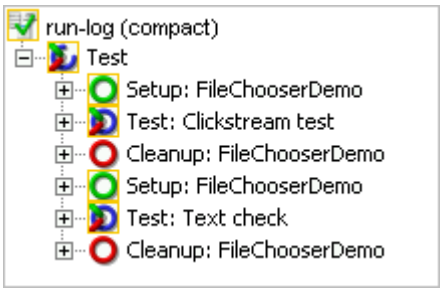

Figure 3.10 - Run-Log de votre Test

Dans le run-log vous voyez comment la paire **Setup/Cleanup** est exécutée avant et après chaque test.

Dans ce test il n'y avait aucune erreur ou exception à l'exécution. Mais vous pouvez voir des cadres jaunes autour de certains nœuds, qui indiquent des avertissements. Dans le cas du FileChooser, ils sont dus à des composants qui n'ont pas de noms attribués. Vous trouverez plus de détails sur l'importance des noms de composants dans le ***Manuel Utilisateur***.

A la fermeture de la fenêtre du run-log il vous est demandé si vous voulez sauvegarder le run-log, dans la mesure où il est uniquement disponible jusqu'à ce que *qftestJUI* soit fermé (afin d'économiser de la mémoire) ou lorsqu'un nombre maximum d'exécutions de test est atteint. Cette option est configurable via le menu **Edit -> Options.**

Maintenant modifions le test afin de créer une erreur dans le contrôle de texte. Cliquez sur le nœud **Check-text** afin de visualiser ses détails :

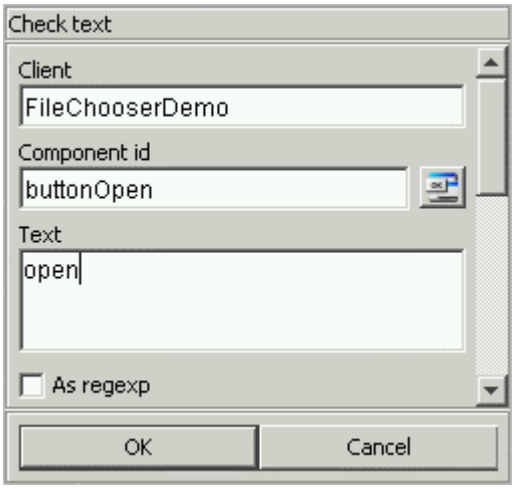

Figure 3.11 - Propriétés du Nœud Check Text

Vous voyez ici que dans le champ **Text** une valeur a été saisie (**"Open"**) qui représente la valeur à laquelle *qftestJUI* s'attend. Changez ce texte par toute autre valeur arbitraire et relancez le test.

Maintenant lorsque vous rejouez le test, une boîte de dialogue apparaît à la fin indiquant qu'il y a eu "**0 Exceptions, 1 Error, 0 Warnings.**"

Si vous ouvrez le run-log, vous voyez que des champs rouges entourent certains nœuds, indiquant que certaines erreurs se sont produites dans les nœuds fils. Vous pouvez utiliser le menu run-log **Edit -> Find next error** pour passer directement au nœud qui a causé l'erreur, ou utiliser le raccourci clavier **Ctrl+N**.

Maintenant nous allons développer notre suite de test pour y inclure un test qui vérifie la logique applicative du SUT.

3.6 Vérification de la Logique Applicative

Jusqu'ici nous nous sommes essentiellement concentrés sur l'implémentation Swing des fenêtres et des composants dans l'application SUT. Ce que nous n'avons pas encore abordé, c'est l'idée de tester la logique applicative réelle, ce qui sera donc l'objet de cette section.

Pour vérifier la logique applicative, il nous faut une relation causale entre certaines actions utilisateur qui génèrent une réaction de l'application. Dans le FileChooserDemo, nous avons mis une telle fonctionnalité qui illustre ce type de relation dans laquelle un champ texte change d'état - de désactivé à activé - sur le clic du bouton radio **Custom** (situé dans la partie gauche de la fenêtre).

Pour notre test, nous avons choisi d'enregistrer un flux de saisie qui active le champ texte, puis vérifie que le champ est réellement activé, et finalement réinitialise l'application par un clic sur le bouton **Open**. Cette séquence est un vrai test action-réaction de l'application puisque deux composants ne sont pas connectés dans Swing. Vérifiez que le contenu du champ texte n'est pas un test de la logique, mais de l'état initial de l'application complète.

Les étapes pour ce test sont très similaires à celles que vous avez suivies pour le test **check text,** avec la séquence flux de saisie et le contrôle enregistrés séparément. Notre solution suite de test est illustrée dans la figure ci-dessous :

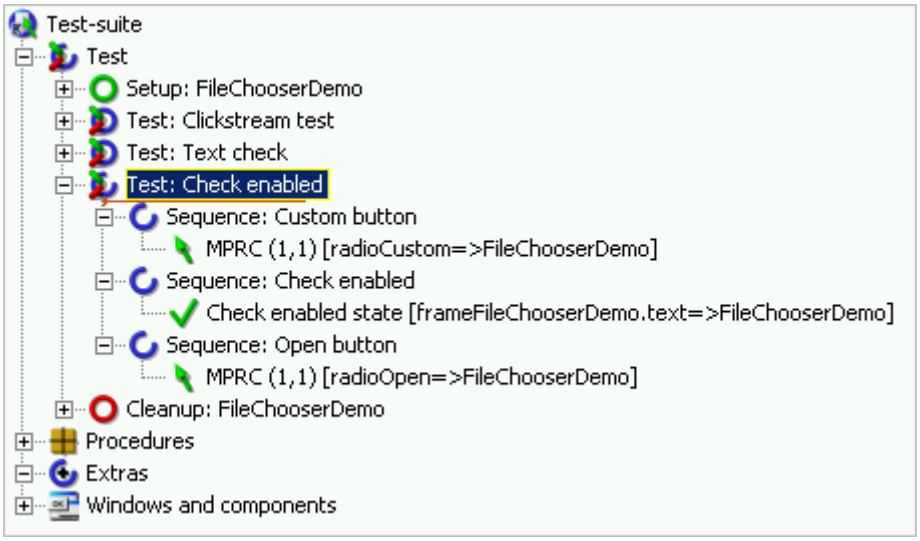

Figure 3.12 - Test Contrôle de l'Etat Activé

Voici un résumé de ce que nous avons fait :

- La première séquence clique le bouton **Custom**, activant ainsi le champ texte.
- La seconde séquence vérifie que le champ texte est réellement actif. Merci de noter également qu'ici il n'y a pas de connexion directe dans Swing entre le bouton et le champ texte, ainsi nous testons bien la logique applicative.
- La troisième séquence réinitialise l'application en appuyant sur le bouton **Open**.

Ce chapitre du didacticiel étant clos, nous allons maintenant nous concentrer sur le diagnostic d'erreurs en présentant plus en détail le débogueur de *qftestJUI*.

CHAPITRE 4 - UTILISATION DU DEBOGUEUR

4.1 Introduction

Dans ce chapitre, nous allons apprendre à exécuter une suite de test avec le débogueur intégré et intuitif de *qftestJUI*. Pour ceux qui sont familiers avec les programmes de débogage en Java ou en d'autres langages de programmation, vous trouverez ce débogueur similaire aussi bien en termes de fonctionnalités que d'utilité.

Pour le didacticiel sur le débogueur, nous utiliserons une suite de test avec laquelle vous vous êtes déjà familiarisé(e) : **Options.qft**. Si cette suite de test n'est pas déjà en exécution sur votre système, merci de bien vouloir vous référer au Chapitre 2.

4.2 Démarrage du Débogueur

Le débogueur de *qftestJUI* peut être démarré soit en sélectionnant un nœud (ou des nœuds) à exécuter et en appuyant sur les boutons **step-in**  ou **step-over**  , ou en utilisant les menus **Debugger -> Step In** ou **Debugger -> Step Over**. Les sections suivantes décrivent plus en détail les fonctions de ces opérations.

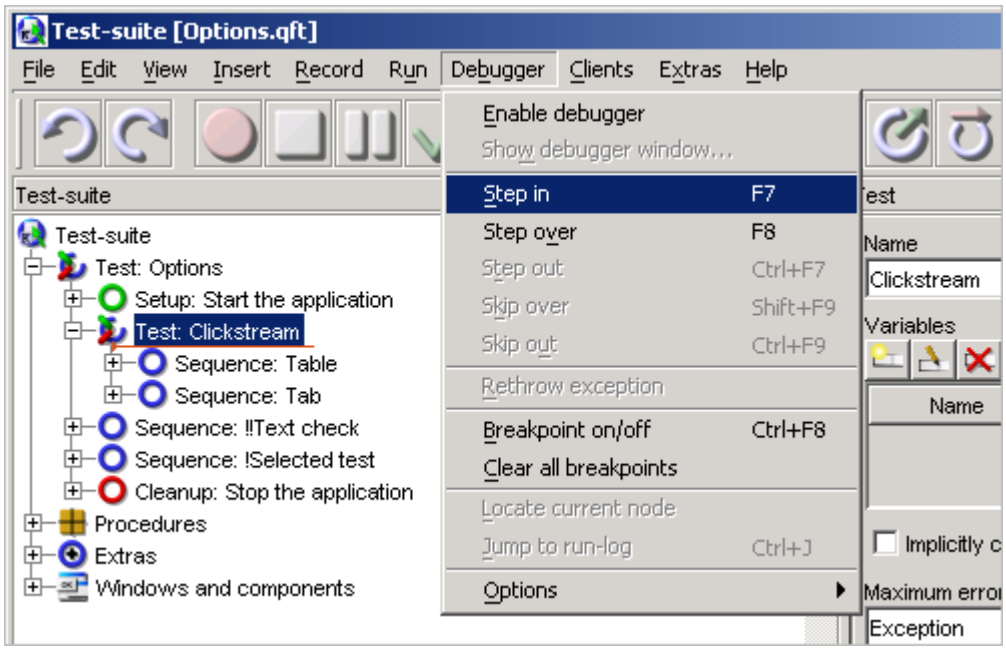

Figure 4.1 - Sélection d'un Nœud dans la Suite de Test Options.qft et Démarrage du Débogueur

Veuillez noter que si vous exécutez les tests sur votre suite de test et que vous utilisez le bouton de rejeu pour démarrer l'exécution, le débogueur ne sera normalement pas visible. Si vous activez le débogueur avec le menu **Debugger -> Enable debugger**, le débogueur sera activé automatiquement si une exception non capturée est générée. Si un point d'interruption défini par l'utilisateur est atteint, le débogueur est activé sans condition. Dans tous les cas, l'exécution de la suite de test sera arrêtée et le nœud qui a arrêté l'exécution apparaîtra avec une flèche.

4.3 Fenêtre Débogueur

Le débogueur est exécuté soit depuis la vue normale de la suite de test, soit en ouvrant la fenêtre D ébogueur dédiée via le menu **Debugger -> Show Debugger Window** (une fois que le débogueur a été démarré).

Vous pouvez également faire en sorte que la fenêtre débogueur s'ouvre automatiquement lorsque le débogueur démarre en changeant les propriétés globales par le menu **Debugger -> Options -> Always Show Debugger Window.**

La fenêtre débogueur ressemble à la vue normale de la suite de test avec laquelle vous êtes désormais familiarisé(e). Elle contient cependant seulement un sous-ensemble des opérations de *qftestJUI* nécessaires au débogage. La figure ci-dessous montre la fenêtre Débogueur après le démarrage du débogueur pour la suite de test **Options.qft.**

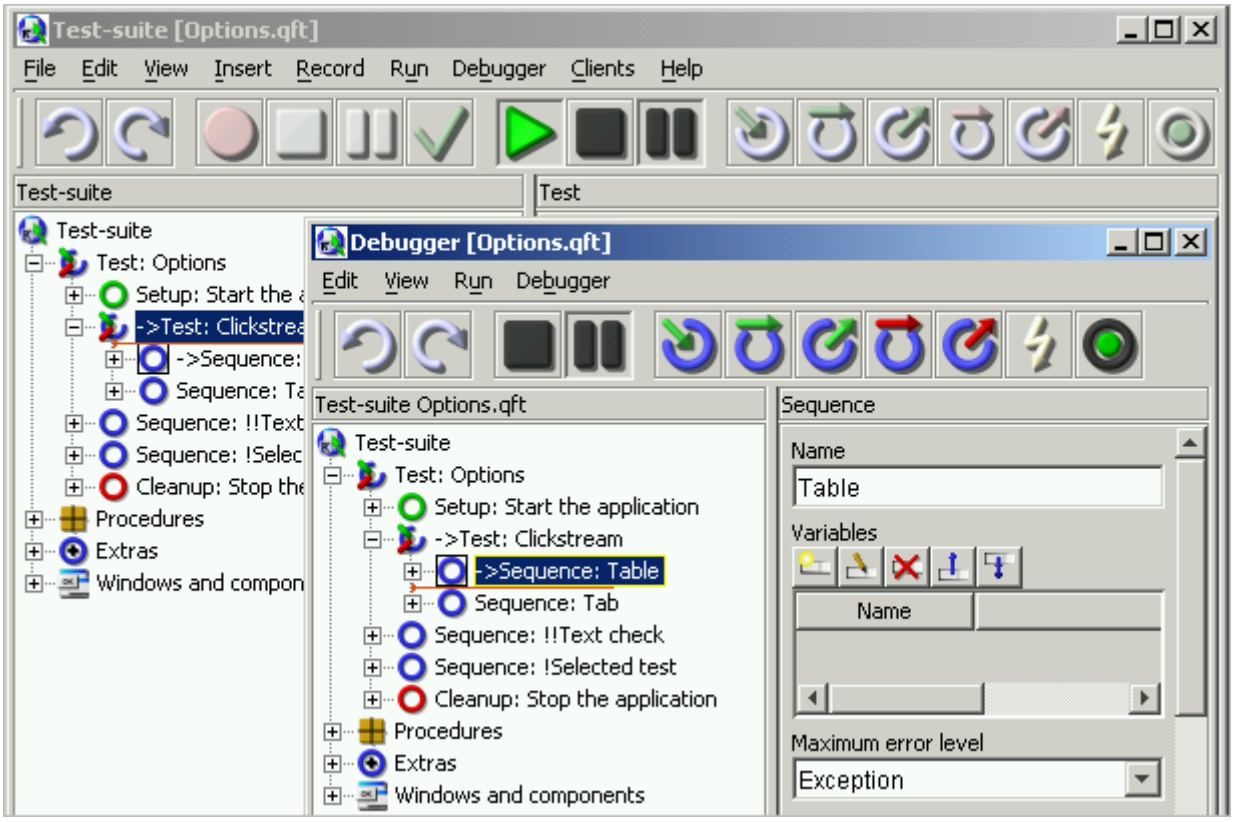

Figure 4.2 - Fenêtre Débogueur

Quand la fenêtre Débogueur est affichée, vous remarquez que les opérations de débogage dans la fenêtre principale suite de test sont alors désactivées, permettant ainsi le contrôle du débogueur uniquement à partir de la fenêtre Débogueur.

4.4 Nœud Actif versus Nœud Sélectionné

Quand le débogueur s'exécute, un indicateur est utilisé pour marquer le nœud courant, qui montre simplement où est le débogueur pendant qu'un test ou une séquence s'exécute. Veuillez noter que *qftestJUI* marque la hiérarchie de l'exécution, depuis le point d'entrée au nœud actif, avec un petit pointeur (Cf. figure 4.2 - Fenêtre Débogueur).

Si à n'importe quel moment pendant que le débogueur s'exécute, vous ne pouvez pas voir immédiatement le nœud courant, vous pouvez appuyer sur le bouton de localisation du nœud courant (**Locate Current Node**) ou sélectionner le menu **Debugger -> Locate Current Node** pour faire en sorte que le débogueur sélectionne le nœud courant.

Un nœud "sélectionné" est un nœud qui est mis en évidence quand vous cliquez dessus avec la souris. Cela peut parfois être source de confusion, parce que le nœud actif pendant le débogage est souvent également sélectionné. Il faut donc apprendre à les distinguer !

4.5 Exécution Pas à Pas d'un Test ou d'une Séquence

Vous êtes maintenant prêt(e) à commencer l'exécution pas à pas des nœuds de votre suite de test pour voir le comportement du débogueur. Avec le nœud actif positionné à Sequence: Table (Cf. figure 4.2), appuyez sur le bouton **step-in**  .

Vous verrez que le débogueur provoque l'ouverture du nœud courant pour afficher ses nœuds fils, et que c'est le premier nœud fils qui est désormais sélectionné, comme le montre la figure ci-dessous :

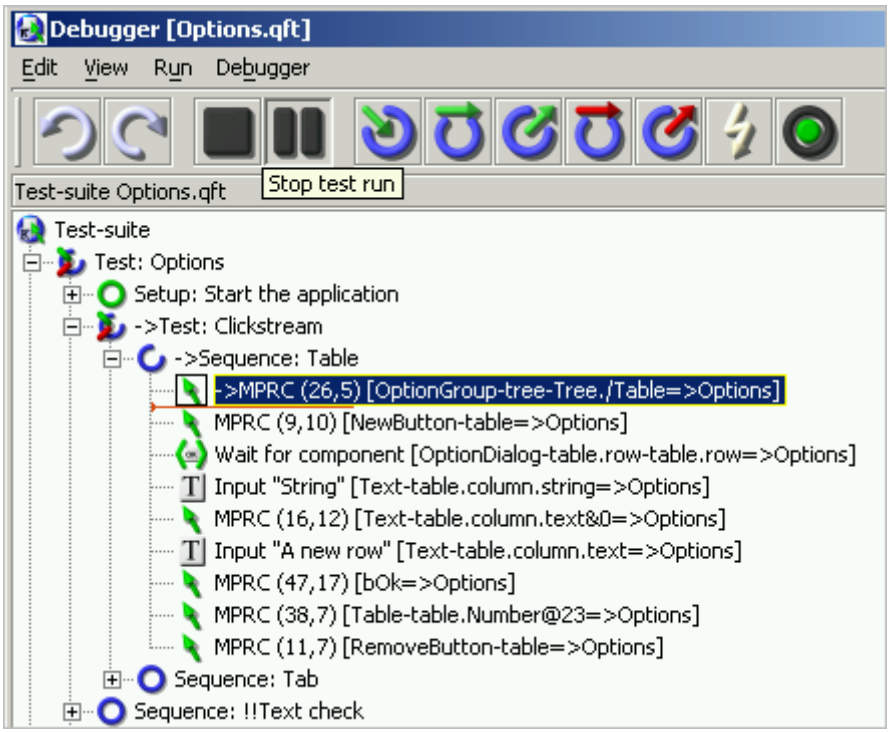

Figure 4.3 - Après une opération de step-in

Essayez d'appuyer sur le bouton **step-in**  plusieurs fois. Le débogueur exécutera chaque nœud, réinitialisera le nœud actif au nœud suivant disponible, et se mettra en pause jusqu'à la commande suivante.

Maintenant que vous avez vu la fonctionnalité de l'opération de step-in, appuyez sur le bouton **Continue test-run** pour permettre à cette séquence particulière de s'exécuter complètement.

Après l'exécution de la première séquence de la suite de test, la vue Débogueur montre quelque chose de similaire à la figure ci-dessous :

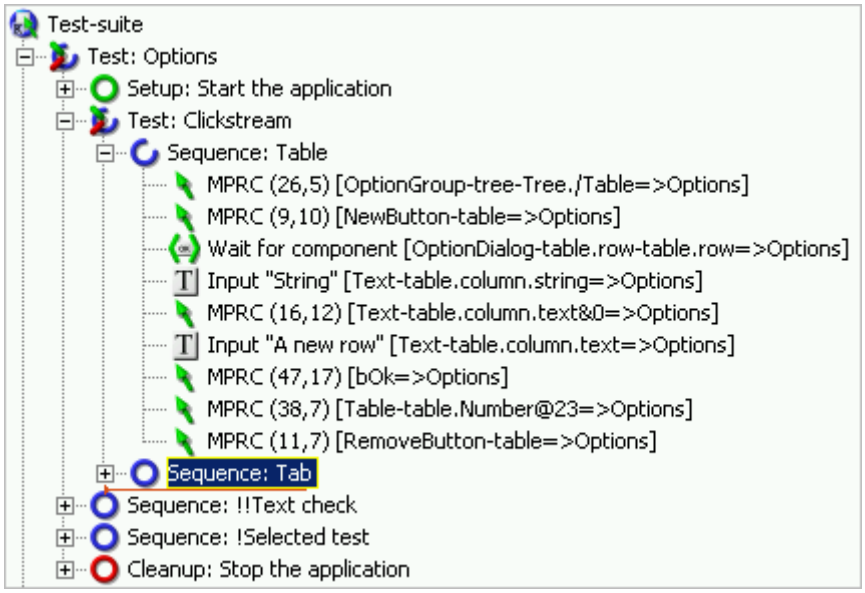

Figure 4.4 - Préparation de la commande step-over

Si ce nœud **Sequence: Tab** n'est pas sélectionné dans votre fenêtre comme montré dans la figure, alors sélectionnez-le. Une fois que vous êtes prêt(e), appuyez sur le bouton **step-over** .

Cette fois, le débogueur réagit en démarrant l'exécution de la séquence, mais à la différence de l'opération de step-in, vous ne voyez pas les étapes individuelles. Tandis que la séquence s'exécute, vous pouvez ouvrir le nœud **Sequence: Tab** pour voir qu'il y a plusieurs nœuds fils dessous, en effet il y a plusieurs autres séquences situées ici.

L'opération **step-over** est donc une façon pour vous d'exécuter un nœud sans avoir à vous préoccuper des détails des étapes individuelles de ce nœud.

Lorsque l'exécution du nœud **Sequence: Tab** est achevée, le nœud **Sequence: Text Check** est alors sélectionné. Appuyez sur le bouton **step-in** afin que la vue du débogueur vous montre ce qui suit :

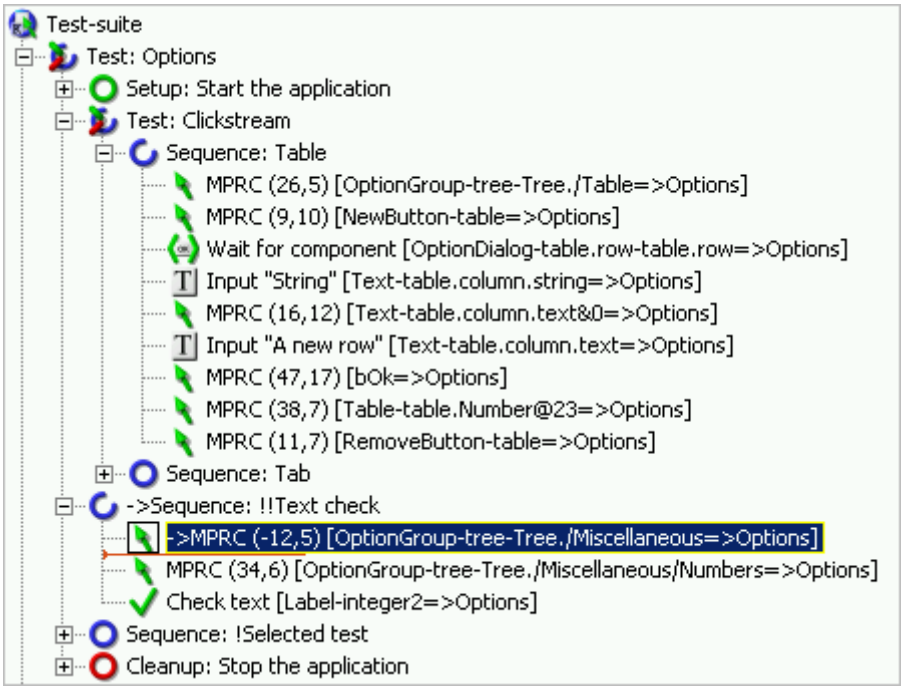

Figure 4.5 - Préparation de la commande step-out

Maintenant appuyez sur le bouton **step-out**  . Vous verrez que le débogueur provoque l'exécution du reste des nœuds dans la séquence et que le nœud **Sequence: Selected test** est alors sélectionné.

Que fait l'opération **step-out** ? Très simplement, elle exécute tous les nœuds trouvés au même niveau dans la hiérarchie que le nœud courant et elle s'arrête lorsqu'un nœud d'un niveau supérieur dans la structure hiérarchique est trouvé. Tout nœud fils trouvé durant l'exécution du step-out est bien évidemment exécuté comme prévu.

Dans cet exemple, la fonctionnalité de step-out peut sembler similaire à celle de l'opération **Continue test-run**. C'est en partie vrai, mais il y a une différence bien nette : l'opération **step-out** s'arrête au nœud suivant qui est plus élevé dans la hiérarchie, tandis que **Continue test-run** fonctionne jusqu'à ce qu'il trouve un nœud qui soit au même niveau que le nœud point d'entrée avec lequel vous avez commencé. Si cela vous semble compliqué, essayez simplement de jouer avec les deux opérations au sein de la suite de test.

4.6 Saut de l'Exécution de Nœuds

Les fonctions **skip** étendent les possibilités du débogueur de *qftestJUI* d'une manière puissante qui n'est typiquement pas possible pour un débogueur dans un environnement de programmation standard. En bref, elles vous permettent de sauter un ou plusieurs nœuds sans avoir à les exécuter du tout. Cela peut s'avérer utile pour différentes raisons, pour naviguer plus rapidement à une certaine position dans l'exécution du test ou pour sauter un nœud qui génère une erreur.

Installez la suite de test à nouveau comme vous l'avez fait au début de la section précédente, et appuyez sur le bouton **step-in** à nouveau jusqu'à ce que le nœud courant soit à l'intérieur de la séquence (Cf. figure 4.3). Maintenant appuyez sur le bouton **skip-out**  . Vous voyez immédiatement que ***qftestJUI*** a simplement sauté la séquence dans laquelle vous étiez sans exécuter les nœuds restants de cette séquence !

Votre suite de test devrait désormais ressembler à la figure 4.4. Laissez-la telle qu'elle est, et maintenant appuyez sur le bouton **skip-over**  . Ici vous voyez un comportement similaire à **skip-out**, *qftestJUI* a sauté le nœud sélectionné sans exécuter aucun des nœuds fils (si vous ouvrez le nœud que vous venez de sauter, vous verrez qu'il y a en effet d'autres séquences, aucune d'entre elles n'a été exécutées).

Remarque : Utilisez prudemment le **skip-over** et le **skip-out**. Sauter une séquence avant son achèvement peut faire que votre SUT soit laissé dans un état inconnu, si bien que d'autres séquences ou tests de votre suite de test ne puissent réagir avec.

4.7 Paramétrage des Points d'Interruption

Paramétrez un point d'interruption est une façon déterministe pour permettre à votre suite de test de s'exécuter jusqu'à ce qu'elle atteigne le nœud que vous avez spécifié comme point d'interruption, lorsque le débogueur sera actif et vous permettra de continuer comme vous le voulez.

Pour paramétrer un point d'interruption, cliquez simplement sur un nœud et sélectionnez le menu **Debugger -> Breakpoint on/off**, ou tapez le raccourci clavier **Ctrl+F8**. Le point d'interruption est indiqué avec un (**B**) devant le nœud du nœud, comme montré ci-dessous :

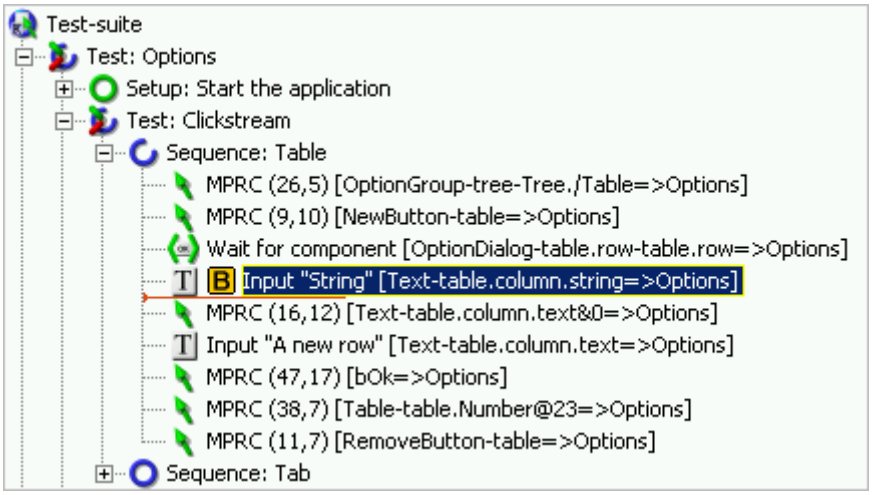

Figure 4.6 - Paramétrage d'un Point d'Interruption

Pour voir comment un point d'interruption fonctionne, sélectionnez le nœud le plus haut dans la hiérarchie de votre suite de test et appuyez sur le bouton de rejeu.

Pour désactiver un point d'interruption, cliquez simplement sur le nœud où le point d'interruption a été paramétré, et sélectionnez le menu **Debugger -> Breakpoint on/off** ou utilisez le raccourci clavier **Ctrl+F8**. Le menu **Debugger -> Clear all breakpoints** est également utile pour supprimer tous les points d'interruption paramétrés dans votre suite de test.

Il n'y a aucune limite quant au nombre de points d'interruption que vous pouvez paramétrer dans votre suite de test, mais veuillez noter que les points d'interruption ne sont pas sauvegardés avec votre session. Aussi lorsque vous fermez la suite de test et puis l'ouvrez à nouveau, les points d'interruption auront disparu.

4.8 Résolution de Problèmes d'Exécution

Lorsque vous commencez à créer et à exécuter vos propres suites de test, vous pouvez vous retrouver confronté(e) à des problèmes comme un ID de composant ou un nom de procédure inconnu. C'est un problème courant, souvent dû à un ID ou un nom incorrect ou peut-être à un changement dans le nom ou l'ID auquel vous vous référencez. Pendant l'exécution, ce problème se manifeste par l'arrêt de l'exécution de votre suite de test, avec un message approprié dans qftestJUI en rapport avec l'élément inconnu.

Cela est l'occasion de démontrer une des plus puissantes fonctionnalités du débogueur de *qftestJUI* : la possibilité de corriger de tels problèmes à la volée pendant l'exécution (runtime) de la suite de test.

En premier, faisons en sorte que votre suite de test génère une exception quand elle s'exécute avec un ID de composant inconnu. Prenez n'importe quel nœud au sein de la suite de test et éditez le champ ID de composant qui est évidemment mauvais. Pour cet exemple, nous avons pris le premier nœud sous la séquence **Table** utilisé en figure 4.6 :

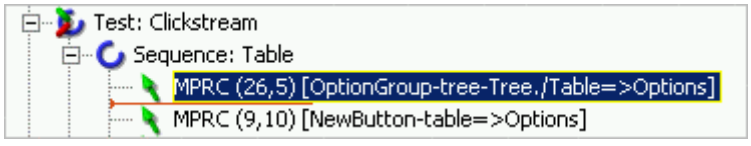

Figure 4.7 - Sélection d'un Noeud à Modifier

et dont l'ID de composant a été modifié pour ressembler à :

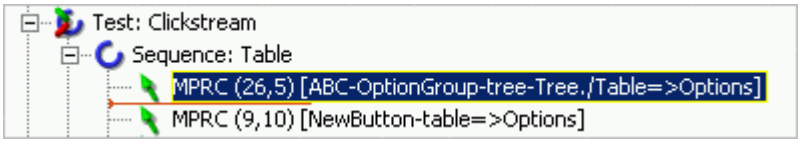

Figure 4.8 - Nœud Modifié

Pour éditer le champ ID composant, cliquez sur le nœud et alors sélectionnez le menu **Edit -> Edit**, ou alors éditez le champ dans la fenêtre Détails (**View -> Show Details** si la fenêtre Détails n'est pas visible à votre écran). Cliquez sur **OK** dans le dialogue d'édition ou la fenêtre Détails après avoir effectué vos changements. Vous verrez alors un avertissement indiquant que l'ID du composant n'est pas valide. *qftestJUI* essaie de trouver les problèmes avant qu'ils ne se produisent à l'exécution. Toutefois, pour cet exemple, nous voulons forcer une référence incorrecte, aussi laissez l'entrée modifiée telle qu'elle est.

Nous pouvons maintenant exécuter la suite de test modifiée. Sélectionnez le nœud **Test: Clickstream** et appuyez sur le bouton de rejeu pour commencer l'exécution. Le test en cours d'exécution va rapidement s'arrêter avec l'apparition d'un message d'erreur :

Figure 4.9 - ID de Composant Inconnu

Cliquez sur **OK** pour fermer la boîte de dialogue. Vous voyez que l'exécution de la suite de test est en pause avec le nœud modifié sélectionné comme nœud courant. Le débogueur attend vos directives. Voulez-vous sauter le nœud incriminé ? Arrêter complètement l'exécution de votre suite de test pour examiner le problème ? Modifier à la volée le nœud ? ou Continuer l'exécution ? Dans ce cas, nous optons pour le dernier choix.

Le débogueur étant en pause, vous pouvez éditer à nouveau le champ **ID** du composant pour le remettre à son état original. Si vous avez oublié quel était son état original, sélectionnez le menu **Edit -> Undo** pour le restaurer (en vous assurant que vous n'avez fait aucun autre changement dans votre suite de test).

Une fois l'ID correct restauré, appuyez sur le bouton **Continue test-run**  pour permettre à l'exécution de reprendre là où elle s'était arrêtée. Vous verrez que la suite de test fonctionne comme avant !

4.9 Saut au Run Log

Parfois retrouver des erreurs dans votre suite de test n'est pas aussi simple que dans l'exemple précédent. La cause de certains problèmes peut résider dans une séquence d'événements qui a amené l'exception ou l'erreur finale que vous avez devant vous.

Dans de telles circonstances, le run-log peut être d'une grande utilité à des fins de débogage, dans la mesure où il montre en détail les différentes étapes de *qftestJUI* pour exécuter chaque nœud et la manière dont il résout les ID de composant, les noms de procédure, les variables ou d'autres éléments.

Comme le run-log peut être fastidieux pour rechercher, plus particulièrement si votre suite de test devient assez complexe, le débogueur vous fournit un mécanisme simple pour passer immédiatement à la région d'intérêt dans le run-log quand une erreur se produit.

Nous allons démontrer cela en utilisant le même exemple simple que celui utilisé en section 4.8, qui a généré une exception en raison d'un ID de composant inconnu. Répétez les mêmes étapes décrites dans cette section et exécutez la suite de test comme précédemment, sauf que maintenant au lieu de corriger l'ID de composant comme nous l'avons fait lorsque l'erreur s'est produite, nous allons en premier inspecter le run-log. Pour passer à l'endroit approprié dans le run-log, sélectionnez le menu **Debugger -> Jump to Run-Log**, ou utilisez le raccourci clavier **Ctrl+J**. Le run-log s'ouvre et le nœud incriminé est automatiquement sélectionné comme montré ci-dessous :

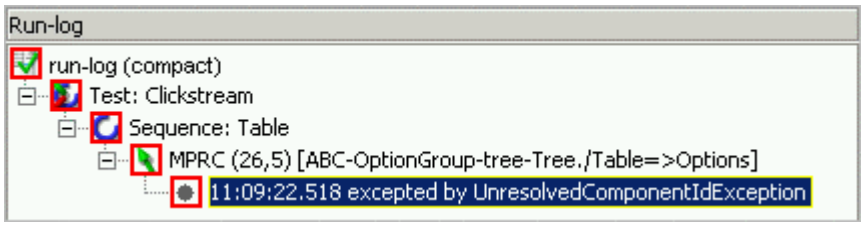

Figure 4.10 - Saut au run-log

Prenez quelques minutes maintenant pour regarder les différents éléments qui vous sont présentés dans le run-log. Bien que l'exemple que nous utilisons est facilement compréhensible, vous familiariser avec ce puissant outil vous sera d'une grande aide dès lors que vos suites de test deviendront compliquées.

Quand vous avez fini, vous pouvez continuer avec l'exemple comme précédemment ou arrêter simplement l'exécution du test. Cependant, rappelez-vous de ne pas sauvegarder les changements dans la suite de test si vous n'avez pas corrigé l'erreur.

CHAPITRE 5 - ECRITURE D'UNE PROCEDURE

Dans les chapitres 5 et 6, nous commençons à explorer des concepts plus avancés de *qftestJUI*, et cela en utilisant une approche pratique afin de vous donner une vue d'ensemble. Nous introduirons, entre autres, le concept important et utile des procédures au sein de *qftestJUI*. D'autres concepts, qui s'avèreront utiles pour développer vos suites de test, seront également présentés.

Pour commencer, nous allons créer une procédure simple qui sélectionnera une **case à cocher** (*checkbox*) au sein du client SUT.

5.1 Démarrage avec une Nouvelle Suite

A ce stade du didacticiel, vous êtes prêt(e) à créer votre propre suite de test en étapes successives. Depuis le menu de *qftestJUI*, sélectionnez le menu **File -> New Suite**.

Conseil : Comme vous l'avez remarqué dans ce didacticiel, des raccourcis clavier sont disponibles pour la plupart des options du menu. Les raccourcis clavier sont des façons rapides de réaliser des actions utiles avec un simple tapotement du clavier. L'option **File -> New Suite**, par exemple, peut également être accessible par le raccourci clavier **Ctrl+N**. Les raccourcis clavier sont indiqués à côté des options des menus dans *qftestJUI*, mais beaucoup d'autres raccourcis sont disponibles, même s'ils ne sont pas immédiatement évidents dans l'IHM. La liste complète est accessible dans la section Raccourcis Clavier du Manuel Utilisateur. Vous pouvez également trouver un petit assistant à attacher à votre clavier qui montre le rôle des touches clavier de *qftestJUI*.

Une suite de test a naturellement besoin d'un client SUT pour être testée avec. L'application **Options Demo**, avec laquelle vous vous êtes familiarisé(e) dans les chapitres précédents de ce didacticiel, est une application très simple qui est idéalement adaptée à nos besoins comme au SUT. La prochaine étape consiste à établir cette application en tant que client SUT pour notre nouvelle suite de test.

Pour cela, vous devez créer un nœud **Setup** qui lancera l'application et établira le lien entre l'application et *qftestJUI*. Pour cette étape, vous pouvez simplement copier la logique depuis **Options.qft** et l'utiliser dans notre nouvelle suite de test. Pour cela, ouvrez **Options.qft** (si vous n'êtes pas familiarisé(e) avec cette suite de test alors référez-vous au chapitre 2) et ouvrez le nœud **Test: Options**. Le premier nœud visible sera le nœud  **Setup: Start the application**. Cliquez sur ce nœud et copiez-le avec **Edit -> Copy**.

Retournez à votre nouvelle suite de test. Il vous faut ouvrir le nœud Test pour qu'une insertion soit possible, et alors vous pouvez coller le nœud copié avec **Edit -> Paste**.

Un nœud Cleanup est habituellement associé à un nœud **Setup** et il est utilisé pour rétablir les étapes réalisées avec le **Setup**. Dans ce cas, nous voulons que le nœud **Cleanup** arrête l'application SUT. A partir de la suite **Options.qft**, copiez le nœud  **Cleanup: Stop the application** et collez-le dans notre nouvelle suite de test juste après le nœud **Setup**.

Vous devez suivre une dernière étape pour que cette section soit compète : copiez les composants et les fenêtres (*Windows and components*) depuis **Options.qft** dans votre nouvelle suite. La raison de cette étape peut ne pas vous sembler évidente à ce niveau, mais nous y reviendrons plus en détail ultérieurement. Les descriptions de composants stockées sous la section **Windows and components** d'une suite de test permettent à *qftestJUI* de reconnaître des éléments, tels que des menus et des boutons dans le SUT. Ouvrez le nœud **Windows and components** de **Options.qft**, sélectionnez les nœuds que vous voyez, et

ensuite utilisez les fonctions **Copier** et **Coller** pour les copier dans le même nœud de votre nouvelle suite de test. Si vous vous y perdez, vous pouvez vous référer à la figure :

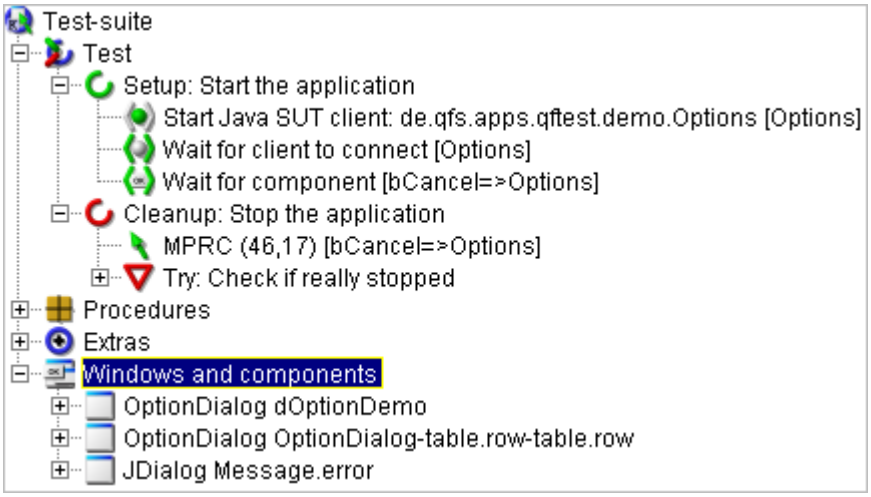

Figure 5.1 - Squelette pour la Nouvelle Suite de Test

A ce stade, vous avez créé un squelette fonctionnel pour votre nouvelle suite de test que nous remplirons ultérieurement avec différentes procédures et tests.

Dès maintenant vous pouvez exécuter le SUT en sélectionnant le nœud **Setup** et en cliquant sur le bouton de rejeu  . Une fois le SUT en exécution, ramenez cette fenêtre et sélectionnez le nœud **Miscellaneous** dans l'arbre **Preferences**. Cela affichera plusieurs options diverses pour Demo, parmi lesquelles la case à cocher **A BooleanOption** :

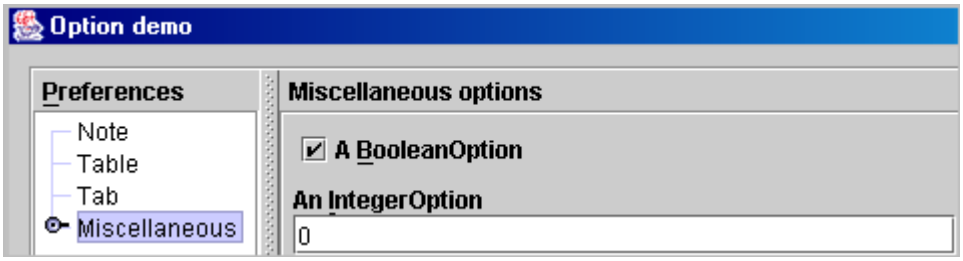

Figure 5.2 - Case à cocher avec les Options Miscellaneous du SUT

Il s'agit de la case à cocher que vous allez manipuler avec votre nouvelle procédure dans les sections qui suivent.

5.2 Création de la Procédure

Une procédure est une séquence d'instructions constituant un sous-programme. Une procédure est typiquement définie comme une méthode utilisée pour accomplir quelque chose, comme par exemple ouvrir une fenêtre. Dans cette partie du didacticiel, vous allez donc créer une procédure qui sélectionne une case à cocher dans le SUT.

En premier, ouvrez le nœud  **Procedures** dans votre nouvelle suite de test.

Conseil : *qftestJUI* vous donne de bons repères visuels comme un nœud qui est ouvert ou fermé. Par exemple, l'icône **Procedures** change d'aspect de  (*fermé*) à  (*ouvert*).

Maintenant cliquez sur le nœud **Procedures** qui est sélectionné, et sélectionnez le menu **Insert -> Procedure nodes -> Procedure**.

Une nouvelle fenêtre de dialogue apparaît alors, vous permettant d'entrer la description de la nouvelle procédure. Nommez-la **selectCheckbox** et appuyez sur le bouton **OK**. Voici à quoi la suite de test doit ressembler après l'ajout de votre première procédure :

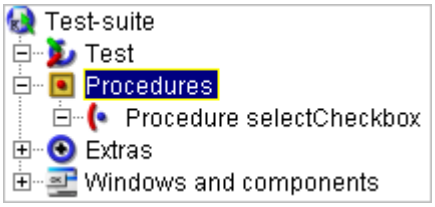

Figure 5.3 - Ajout d'une Procédure

Conseil : Les propriétés d'un nœud dans *qftestJUI* (comme le nom de la procédure) peuvent être changées tout simplement en cliquant sur le nœud et en éditant ses propriétés dans la fenêtre Details, ou par un clic droit sur le nœud et en sélectionnant **Edit**. Dans tous les cas, il vous faudra appuyer sur le bouton **OK** pour confirmer les changements effectués.

5.3 Ajout d'un Nœud de Contrôle

Nous sommes désormais prêts à ajouter de la substance réelle à la procédure. Mais en premier lieu pensez à la manière dont vous allez paramétrer la case à cocher. Une des premières étapes consiste à vérifier l'état de la case à cocher avant de procéder à son paramétrage. Si elle est déjà cochée, alors il n'y a rien à faire si c'est de ne la laisser telle qu'elle. Autrement, si elle n'est pas cochée, nous devons la paramétrer.

De tels contrôles sont faits dans *qftestJUI* avec les **nœuds de contrôle** (*check-nodes*), qui vous permettent d'effectuer différents types de requêtes concernant l'état d'éléments dans le SUT. Celui avec lequel nous commencerons s'appelle le nœud de contrôle **Check selected state**. Comme son nom l'indique, il interroge l'état d'un composant pour voir s'il est sélectionné ou non.

Le chapitre précédent du didacticiel nous a permis d'acquérir une première expérience des nœuds de contrôle. Si cela n'est pas le cas, merci de vous référer à la section 3.5 du chapitre 3 dès maintenant. Pour ajouter le nœud **Check selected state**, vous devez maintenant réaliser des étapes similaires.

Ramenez la fenêtre du SUT et sélectionnez le nœud **Miscellaneous** dans l'arbre **Preferences** comme montré en figure 5.2. Cela vous permettra d'obtenir facilement le composant correct une fois que vous aurez démarré l'enregistrement. Maintenant retournez dans votre suite de test et cliquez sur le bouton enregistrement d'un contrôle  , et ensuite retournez à la fenêtre du SUT. Comme précédemment, lorsque vous déplacez la souris sur les différents composants, cela intervertit leur couleur, qui indique la sélection. Sélectionnez la case à cocher **BooleanOption** (que nous avons identifié précédemment) et faîtes un clic droit. En plus des différents types de contrôles que nous avons vu dans les exemples précédents, le menu contextuel qui s'affiche contient une option pour **selected state**, comme montré dans la figure ci-dessous :

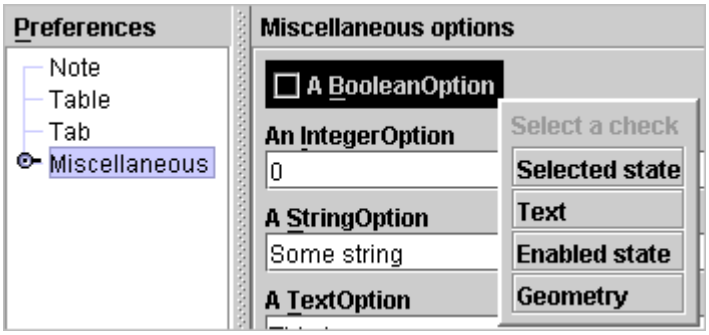

Figure 5.4 - Enregistrement du Nœud Check selected state

Choisissez l'option **selected state**, retournez dans votre suite de test et arrêtez l'enregistrement en appuyant sur le bouton d'arrêt  .

qftestJUI placera la séquence enregistrée dans le nœud **Extras** de votre suite de test, qui sera immédiatement visible dès l'arrêt de l'enregistrement. Si vous ouvrez la séquence enregistrée, vous voyez que le nœud **Check selected state** est bien présent. Ce nœud doit maintenant être transféré de son emplacement actuel vers votre procédure. Vous pouvez transférer soit le nœud, soit la totalité de la séquence enregistrée. Dans la mesure où il n'y a qu'un seul nœud fils dans la séquence, la différence est purement esthétique.

Utilisez la fonction **Copier** ou **Couper** puis Coller pour la placer dans votre procédure. Veillez à ouvrir le nœud de la procédure **selectCheckbox** avant de coller le nœud copié, autrement il ne sera pas placé au bon emplacement dans la procédure :

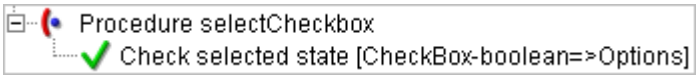

Figure 5.5 - Nœud Check Selected State dans la Procédure

Conseil : *qftestJUI* fournit un dispositif vous permettant d'insérer directement une séquence enregistrée à un emplacement donné, évitant ainsi d'avoir recours à la fonction **Copier-Coller**. Sélectionnez le menu **Edit -> Options**. Dans la fenêtre d'options, sélectionnez alors l'onglet **Record** pour afficher les options d'enregistrement. Dans cette fenêtre, sélectionnez la case à cocher **Insert recording at current selection**. Dès lors que cette option est sélectionnée, *qftestJUI* insère n'importe quelle séquence enregistrée après le nœud que vous avez sélectionné, et non pas dans le nœud **Extras.**

Faîtes un essai ! Sélectionnez votre procédure et cliquez sur le bouton de rejeu  . En fait pas grand chose ne se produit, si ce n'est l'affichage d'un message dans la barre inférieure d'état de *qftestJUI* vous indiquant que la procédure est achevée. Cette première étape est donc terminée.

5.4 A Propos des Composants

Avant de continuer, voici quelques informations supplémentaires et utiles à connaître sur les contrôles et les composants.

Un composant est un élément du SUT, comme un bouton, un menu ou une case à cocher, dans notre cas. Chaque composant doit avoir un ID unique, qui est enregistré dans la suite de test. Aussi qftestJUI peut alors trouver le composant à l'exécution de la suite de test.

Si vous cliquez sur le nœud **Check selected state** que vous avez enregistré dans la section précédente, vous verrez ses propriétés dans la fenêtre **Details** sur la droite. Parmi ses propriétés il y a un champ appelé **Component ID**, qui doit être

renseigné avec l'identité de la case à cocher identifiée en figure 5.2. Vous verrez que ce composant a pour ID de composant **Checkbox-boolean**.

Cet ID de composant que vous voyez est seulement un nom, un moyen pratique pour vous et pour *qftestJUI* de référencer le même composant. Ces ID sont enregistrés dans la section **Windows and components** de votre suite de test, dès lors que vous enregistrez quelque chose dans le SUT. Si vous ouvrez tous les nœuds dans cette section, comme montré ci-dessous, vous trouverez finalement le composant que nous avons enregistré pour **Check selectedstate** :

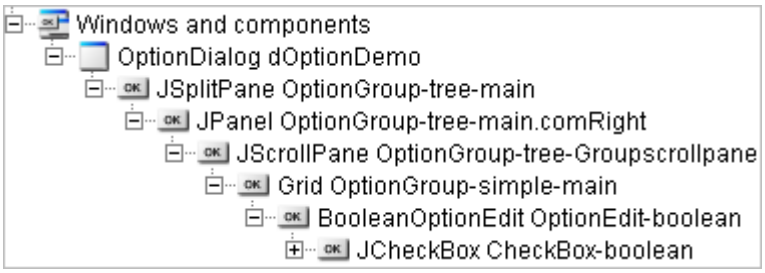

Figure 5.6 - Le Composant Case à cocher dans le Composant Arbre

Conseil : Un moyen facile de localiser un composant est d'utiliser la fonctionnalité **locate-component** de *qftestJUI*. Affichez le menu contextuel par un clic droit sur le nœud **Check selected state** et sélectionnez **Locate component**, ou alors sélectionnez le nœud et appuyez sur les touches de votre clavier **Ctrl+W**. Cela vous amènera directement au nœud Component pour le contrôle. Appuyer sur **Ctrl+barre d'espacement** vous ramènera au nœud contrôle.

Allez-y et cliquez sur le nœud pour l'ID de composant **CheckBox-boolean**. Vous voyez alors comment *qftestJUI* identifie de façon interne les composants.

La reconnaissance de composants au sein de *qftestJUI* est un sujet complexe, et nous n'approfondirons pas davantage ce sujet dans le présent document. Pour plus de détails sur les composants, merci de bien vouloir vous référer au ***Manuel Utilisateur***.

5.5 Mécanisme Try/Catch

Mais que se passe-t-il si votre case à cocher n'est pas vérifiée ? Décochez votre case maintenant, et rejouez la procédure. Vous devez voir un message comme celui-ci quand la procédure s'achève :

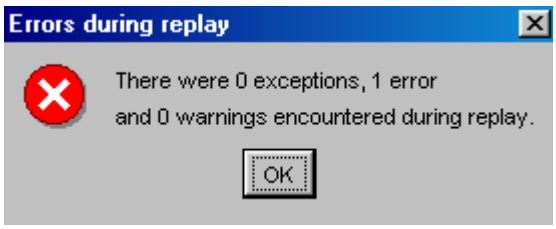

Figure 5.7 - Erreur Check Selected State

L'erreur s'est produite parce que votre nœud **Check selected state** s'attendait à ce que la case soit cochée.

Cliquez à nouveau sur le nœud **Check selected state** pour visualiser ses propriétés dans la fenêtre de détails. Parmi les détails, vous verrez des champs marqués **error level of message** et une case à cocher **Throw exception on failure**. En utilisant ces propriétés, vous pouvez modifier la mesure que *qftestJUI* prendra quand un contrôle échoue. Essayez de changer ces paramètres et observez les résultats après le rejeu de la procédure.

Finalement, nous voulons que ce contrôle génère une exception quand il échoue. C'est l'occasion d'introduire le mécanisme du try/catch. Paramétrez la case à cocher **Throw exception on failure** dans les propriétés comme ci-dessous :

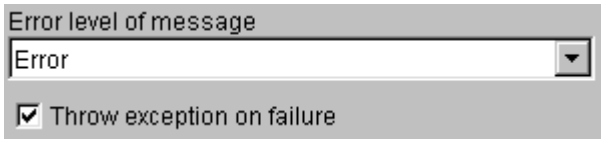

Figure 5.8 - Génération d'une Exception sur un Echec

Le mécanisme du **Try/Catch** est une technique courante utilisée par la plupart des langages de programmation pour gérer les exceptions (c'est-à-dire des conditions inattendues) quand elles se produisent. Le concept est simple : '**try**' pour effectuer une certaine action ; si une exception se produit pendant l'action, alors '**catch**' la capture et effectue d'autres séquences d'opérations. Si ce mécanisme ne vous est pas familier, ne vous inquiétez pas. Les choses deviendront plus claires une fois mises en œuvre.

Insérez un nœud **Try** directement au début de votre procédure. Pour se faire, cliquez sur le nœud de la procédure (le point d'insertion est toujours après le nœud qui est sélectionné), puis sélectionnez le menu **Insert -> Control structures -> Try**. Un dialogue vous permettant d'entrer les propriétés du nœud s'affiche alors. Pour ce nœud, vous pouvez ne renseigner aucun des champs, cependant pour des questions de lisibilité, il est parfois utile de saisir un nom. Nous écrirons **checkbox selected**.

Ouvrez le nœud **Try**, ainsi vous pouvez y insérer des nœuds. Toutes les actions prises dans le bloc **Try** peuvent potentiellement être capturées par un nœud **Catch** si une exception surgit (qui sera expliquée peu après). Ainsi, nous voulons maintenant insérer l'action que nous voulons **Try** - qui est notre nœud contrôle **Check selected state**. Ce nœud doit être dans notre procédure, mais probablement pas dans le bon emplacement. Utilisez les fonctions **Couper** et **Coller** pour insérer ce nœud immédiatement après votre nœud **Try**.

Un nœud **Try** est inutile sans **Catch**, c'est donc l'étape suivante. Insérez un nœud **Catch** après votre nœud de contrôle (mais encore dans le bloc **Try**) avec le menu **Insert -> Control structures -> Catch**. Encore une fois, un dialogue apparaît pour vous permettre de renseigner les propriétés du nœud. Pour le nœud **Catch**, vous devez définir le type d'exception que vous voulez capturer. Le premier champ est **exception class**, qui contient une liste de toutes les exceptions possibles. Sélectionnez **CheckFailedException**.

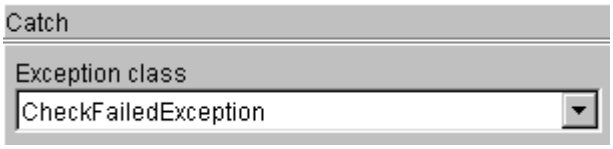

Figure 5.9 - Sélection d'une Exception à Capturer

Le mécanisme **Try/Catch** est destiné à être un bloc d'instructions déterministe. L'exception (ou les exceptions) devant être capturée(s) sont en rapport direct avec les actions entreprises dans le bloc Try. Dans notre cas, le bloc **Try** contient un nœud de contrôle à **Check selected state**, aussi l'exception à capturer est **CheckFailedException**. Bien sûr, cela n'est pas immédiatement évident pour vous lors de l'implémentation de ce mécanisme. Si vous n'êtes pas sûr(e) du type d'exception qui sera lancé et que vous avez besoin de capturer, le run-log peut s'avérer utile. Essayez de rejouer votre procédure. Ne vous inquiétez pas si les nœuds **try/catch** ne sont pas complets. Ce qui est important c'est que le nœud **Check selected state** soit encore présent et ait pour propriété **Throw exception on failure**. Après, ouvrez le menu **Run** et sélectionnez le run-log le plus récent (le plus récent est toujours en position "**1**"). Parcourez le run-log et ouvrez les nœuds. Vous devez alors voir l'élément contenant l'erreur générée, comme ci-dessous :

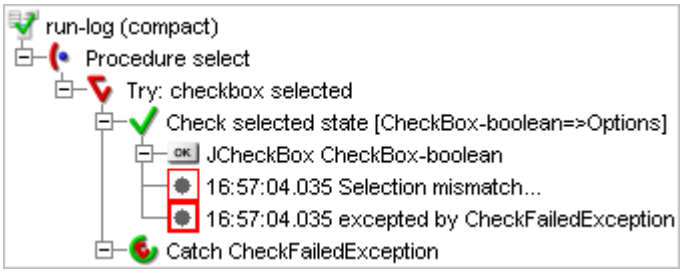

Figure 5.10 - CheckFailedException dans le Run-Log

Conseil : Pour plus d'informations sur les exceptions et une liste complète des exceptions possibles, merci de vous référer au ***Manuel Utilisateur.***

A ce stade, votre procédure doit ressembler à cela :

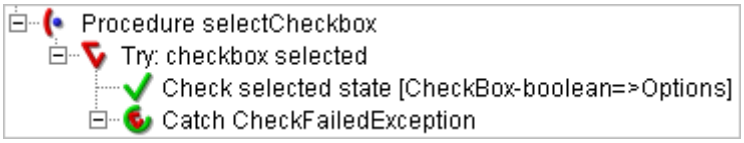

Figure 5.11 - Procédure avec Try/Catch

Conseil : Laissez *qftestJUI* vous aider si vous avez des questions sur un nœud donné : faîtes un clic droit sur le nœud pour lequel vous avez besoin d'aide et sélectionnez l'option "**What's this?**" du menu contextuel - *qftestJUI* vous amène alors directement à la bonne rubrique du Manuel Utilisateur.

5.6 Réalisation de la Logique de Sélection

Vos nœuds **try/catch** sont en place, la dernière étape consiste à effectuer les actions qui seront mises en place si **CheckFailedException** est générée et capturée. Rappelez-vous que l'exception sera générée si le contrôle échoue, c'est-à-dire si la case à cocher n'est pas paramétrée. L'action à prendre est de paramétrer la case à cocher si l'exception est capturée.

Ici nous enregistrons une autre séquence du SUT. Cliquez le bouton  et affichez la fenêtre SUT. La seule chose que nous voulons enregistrer est un clic sur la case à cocher **BooleanOption**. Donc cliquez sur la case à cocher et changez l'état de "**paramétré à non-paramétré**", ou vice-versa (cela n'a pas d'importance car nous voulons enregistrer le changement d'état).

Maintenant retournez à votre suite de test et cliquez sur le bouton  . Après l'arrêt de l'enregistrement, *qftestJUI* place la séquence enregistrée dans la section **Extras** de votre suite de test. En jetant un coup d'œil dans la séquence enregistrée, vous verrez un nœud **MPRC**, qui représente pour un événement souris combiné "**Déplacé, Pressé, Relâché, Cliqué**". En regardant les propriétés du nœud, vous verrez des éléments familiers comme l'ID de composant de la case à cocher.

Conseil : Vous n'êtes pas obligé(e) de faire des changements dans les propriétés de ce nœud, mais y ajouter un commentaire peut quelquefois s'avérer utile. Les commentaires n'ont aucun effet sur le test, ils permettent seulement une meilleure lisibilité de la suite de test.

Comme nous l'avons fait précédemment lors de l'utilisation d'un nœud enregistré, vous devez déplacer ce nœud **MPRC** depuis Extras dans votre procédure en utilisant les fonctions **Copier/Couper** et **Coller**. Ouvrez le nœud **Catch** dans votre procédure et insérez le nouveau nœud après. Votre procédure doit ressembler à ceci :

Figure 5.12 - Procédure avec un Mécanisme Try/Catch Complet

Votre procédure est désormais une unité fonctionnelle complète. Cliquez sur le nœud **procédure** et essayez de l'exécuter plusieurs fois en utilisant le bouton de rejeu  , et en changeant l'état de la case à cocher dans le SUT, ainsi vous pouvez voir les deux chemins logiques en exécution. Utiliser la fonction **Step-in** du Débogueur pour exécuter pas à pas votre procédure est un excellent moyen pour voir exactement comment le mécanisme try/catch fonctionne.

Juste une petite remarque. Une bibliothèque qui contient un ensemble de procédures utiles comme **selectCheckbox** fait partie de la distribution de *qftestJUI* (Cf. Chapitre 8). Dans un premier temps, nous allons consacrer un peu de temps à l'amélioration de notre procédure.

5.7 Amélioration de la Procédure

Nous allons effectuer une dernière amélioration à notre procédure avant de clore ce chapitre du didacticiel. Le problème avec cette procédure est qu'elle fonctionne uniquement pour une case à cocher spécifique. Et si le SUT avait plus d'une case à cocher à paramétrer ? Devons-nous écrire des procédures séparées pour chaque case à cocher ?

La solution consiste à utiliser des variables. Une variable peut être utilisée pour substituer l'ID du composant de la case à cocher, ainsi la procédure peut être utilisée pour n'importe quelle case à cocher du SUT.

La syntaxe des variables dans *qftestJUI*, comme avec tout programme, est spécifique. Une variable est définie simplement avec un nom. Toutefois, lorsqu'on se réfère à une variable la syntaxe requise est : "**$(name)**" - ce sujet sera développé dans les sections suivantes.

Remplaçons les références dans la procédure à la case à cocher spécifique par une variable maintenant. Appelons la variable "**$(id)**" par exemple. Remarquez qu'il y a deux références au composant case à cocher, un dans le nœud **Check selected state**, et l'autre dans le nœud **MPRC**. Pour effectuer ce remplacement, affichez les propriétés du nœud et localiser le champ ID du composant. Supprimez le texte **CheckBox-boolean** et saisissez **$(id)** - la procédure doit désormais ressembler à :

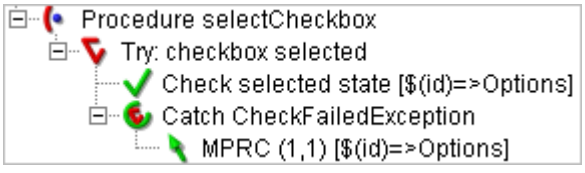

Figure 5.13 - Variable Remplaçant le Composant Fixe

5.8 Appel de la Procédure

Si vous essayez de rejouer la procédure maintenant, vous obtiendrez une erreur indiquant que **The variable 'id' was not bound**. C'est une manière pour *qftestJUI* de vous faire savoir qu'aucune valeur n'a été positionnée pour la variable.

Alors comment positionne-t-on une valeur pour la variable ? Pour notre exemple, la façon la plus courante et la plus utile est d'"appeler" la procédure avec une valeur spécifique pour la variable, c'est-à-dire en "passant un argument (ou paramètre)" à la procédure.

Maintenant insérons un appel à la procédure. Nous retournons au nœud **Test** principal de la suite de test. Fermez le nœud **Setup** pour insérer un nœud après, et maintenant sélectionnez le menu **Insert -> Procedure nodes -> Procedure cal**l. Une fenêtre de dialogue s'affiche alors pour vous permettre de renseigner les propriétés du nœud. Entrez le nom de la procédure dans le premier champ qui est la procédure **selectCheckbox**.

Ensuite nous voulons ajouter la variable qui sera passée comme argument à la procédure. C'est accompli avec la table **Variables** qui apparaît après le nom de la procédure dans la fenêtre de propriétés. Cliquez sur la première icône  pour cette table pour ajouter une nouvelle ligne. Un nouveau dialogue **Edit table row** apparaît alors et vous permet de faire une nouvelle entrée. Dans cette fenêtre, entrez le nom (**id**) puis la valeur (qui est le composant que nous voulons modifier **CheckBox-boolean**). L'appel à la procédure soit alors ressembler à :

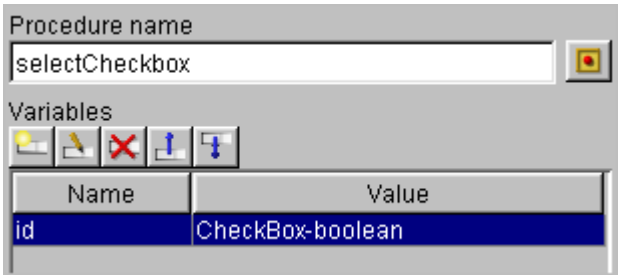

Figure 5.14 - Propriétés d'un Appel de Procédure

Conseil : En dehors des icônes qui servent de fonctions pour ajouter, éditer et supprimer des éléments dans la liste de variables pour la procédure, vous remarquerez une autre icône dans l'image ci-dessus : le bouton jaune et rouge  situé après le champ **Procedure name**. Cette icône est utilisée pour afficher une liste de procédures utiles situées dans votre suite de test, au cas où vous ne vous souviendriez plus du nom de la procédure. Localisez la procédure **selectCheckbox** et appuyez sur **OK**. *qftestJUI* remplira automatiquement le nom dans les propriétés.

Une fois les propriétés pour l'appel à la procédure renseignées, cliquez sur **OK**. Vous pouvez alors sélectionner le nouveau nœud et appuyer sur le bouton de rejeu  . Vous voyez maintenant que l'appel passe l'ID de notre case à cocher à la procédure pour que **selectCheckbox** fonctionne correctement. Si le SUT a plusieurs cases à cocher, vous pouvez maintenant ajouter plusieurs appels à **selectCheckbox**, chacun avec un ID de composant différent à la procédure.

CHAPITRE 6 - CREATION D'UNE PROCEDURE GENERALE

Dans ce chapitre, nous vous montrerons de quelle manière vous pouvez rendre la procédure que vous avez écrit dans le chapitre précédent encore plus utile. Avec l'introduction des **packages**, vous verrez comment construire votre propre bibliothèque d'utilitaires pour des fonctions d'ordre général.

6.1 Création d'un Package

Alors que vous commencez à écrire plus de procédures, vous trouverez rapidement que les choses commencent à être fouillies et désordonnées. Comme un dossier de fichiers qui vous aide à organiser vos documents en différentes catégories, un package dans *qftestJUI* vous aide à grouper ensemble vos procédures. La définition d'un package dans *qftestJUI* est donc tout simplement un groupe de procédures et/ou d'autres packages, mais rarement un mélange des deux.

Dans le dernier chapitre, vous avez écrit une procédure utilitaire case à cocher appelée **selectCheckbox**, qui vous a permis de vérifier une case à cocher dans le SUT. Nous pourrions cependant imaginer d'autres utilitaires pour une case à cocher. Et si vous voulez désélectionner une case à cocher ou questionner si une case à cocher est grisée/désactivée ? Il y a toute sorte de possibilités, et toutes tombent dans une catégorie générale, c'est-à-dire utilitaires de case à cocher. Aussi créons un package pour ces procédures.

Cliquez sur le nœud **Procedures** de votre suite de test et sélectionnez le menu **Insert -> Procedure nodes -> Package**. Cela affichera la fenêtre de dialogue de propriétés dans laquelle vous devez seulement renseigner un nom pour le package. Appelez-le **checkbox**.

6.2 Déplacement de la Procédure

Vous avez déjà une procédure qui traite d'une case à cocher et qui peut maintenant être déplacée dans votre nouveau package. Utilisez les fonctions **Couper** et **Coller** pour déplacer la procédure **selectCheckbox** dans le package **checkbox**.

Vous remarquerez que le nom de votre procédure est quelque peu superflu maintenant qu'il est dans le package **checkbox**. Changez le nom de la procédure **selectCheckbox** par **select**. C'est un petit avantage de l'organisation du package : tout ce qui est à l'intérieur du package **checkbox** est supposé traiter d'une case à cocher.

Voici à quoi votre suite de test doit maintenant ressembler :

Figure 6.1 - Package Checkbox

6.3 Création de la Procédure Deselect

L'étape suivante consiste à développer le package un petit peu en ajoutant une autre procédure utilitaire. Un choix sensé serait, bien sûr, une procédure **deselect** qui décoche une case à cocher.

Votre nouvelle procédure **deselect** fonctionnera essentiellement de la même façon que select avec seulement quelques changements mineurs dans la logique. Commençons par faire une copie de select et la coller dans le package **checkbox**. Ne vous inquiétez pas si vous avez deux procédures avec le même nom dans le package, ce point sera corrigé ultérieurement.

Vous avez maintenant deux procédures identiques dans le package **checkbox**. Cliquez sur l'une d'entre elles pour afficher ses propriétés et en changer le nom par **deselect**.

La dernière étape pour cette section est de changer la logique de manière à ce que **deselect** recherche désormais l'état désélectionné de la case à cocher avant d'agir. Pour cela, vous avez seulement besoin de faire un changement. Ouvrez les branches de la procédure **deselect** jusqu'à ce que vous localisiez le nœud **Check selected state**. Rappelez-vous que c'est le nœud de contrôle qui interroge l'état de la case à cocher. La procédure **select** est implémentée de sorte qu'une exception soit générée si la case à cocher n'est pas sélectionnée. Nous pouvons inverser cette logique maintenant pour **deselect** en localisant la case à cocher **Selected** dans les propriétés du nœud et en la décochant, comme suit :

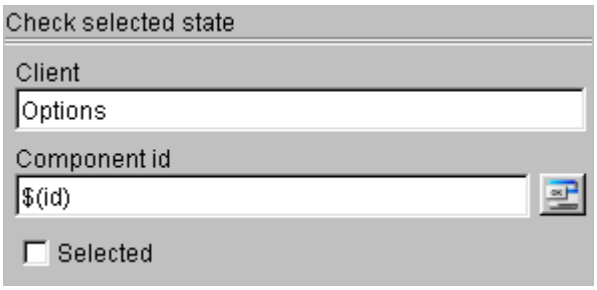

Figure 6.2 - Vérification pour un Etat Non-sélectionné

Maintenant le nœud contrôle attend un état non sélectionné du composant et génèrera une exception si le composant est sélectionné, ce qui est exactement ce à quoi nous voulons arriver. Vous pouvez vouloir changer certaines des étiquettes et/ou des commentaires dans la procédure, par exemple le nom du nœud **Try** pourrait être changé en **checkbox not selected**. Voici comment votre nouveau package doit être formé :

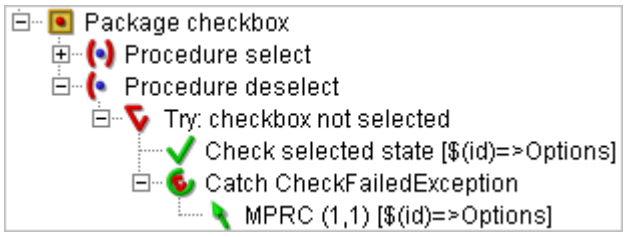

Figure 6.3 - Package Checkbox avec deux Procédures

6.4 Appel des Procédures

Dans le chapitre précédent (Cf. section 5.8), vous avez créé un appel à la procédure **selectCheckbox**. Maintenant, il vous faut changer cet appel pour qu'il ait un nom correct.

Les appels aux procédures dans des packages suivent une règle de syntaxe qui référence la hiérarchie entière de la procédure d'une manière élégante : **package.procedure** - ainsi, votre appel de procédure sera maintenant à la procédure nommée **checkbox.select**.

Conseil : Les packages peuvent, selon notre définition, contenir non seulement des procédures, mais également d'autres packages. Si une procédure est située dans une telle structure de package emboîtée, alors la même règle pour appeler la

procédure s'applique comme ci-dessus, avec autant d'états package. devant le nom de la procédure que nécessaire. Par exemple, votre appel de procédure pourrait être quelque chose comme **menu.file.open**, qui implique deux packages (**menu** et **file**) avec la procédure nommée **open**.

Essayez de créer un autre appel de procédure maintenant à la procédure **checkbox.deselect**. Vous devez désormais savoir comment procéder en utilisant les techniques décrites dans ce didacticiel.

6.5 Création d'une Procédure Polyvalente

Les deux procédures que vous avez maintenant dans le package **checkbox** vous permettront de sélectionner ou de ne pas sélectionner séparément une case à cocher. Comment serait-elle si nous avions créé une procédure simple qui combine les deux fonctions ? Dans cette section et les suivantes, vous allez créer une procédure appelée **setState** qui positionnera l'état d'une case à cocher à "sélectionné" ou "non sélectionné". Cette nouvelle procédure inclura une variable que nous appellerons **select**, qui si elle est positionnée à **true** fera que la procédure sélectionnera la case à cocher, autrement (**select** est **false**), la case à cocher ne sera pas sélectionnée.

Commençons par créer la nouvelle procédure **setState** dans le package **checkbox**. Si vous avez besoin d'aide pour créer une nouvelle procédure, merci de bien vouloir vous référer à la section 5.2 du chapitre 5.

6.6 Paramétrage par Défaut

Comme nous venons de l'indiquer, votre nouvelle procédure contiendra la variable **select** qui sera utilisée pour décider quoi faire avec la case à cocher. Comme avec l'ID du composant que vous avez passé à la procédure dans votre appel de procédure, une valeur **select** (soit **true**, soit **false**) doit également être passée comme argument à **setState**.

Il est cependant habituel pour les procédures qui se servent d'une telle variable de définir un paramétrage par défaut pour la variable. Cela signifie que passer une valeur **select** à la procédure comme argument est facultatif. Quand aucune valeur n'est passée, la variable prend la valeur par défaut.

Cliquez sur le nœud procédure **setState** pour visualiser ses propriétés dans la fenêtre de détails. Vous remarquerez qu'une zone pour définir des variables est incluse dans les propriétés. En fait, des variables pour une procédure sont ajoutées, éditées et supprimées de la même manière que ce que vous avez vu en section 5.8 du chapitre 5). De la même manière, ajoutez maintenant la variable **select** et donnez-lui pour valeur **true**. Voici à quoi les propriétés de votre procédure doivent ressembler :

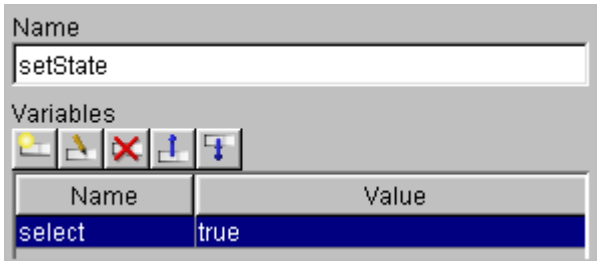

Figure 6.4 - Valeur par défaut pour la Variable Procédure

Ce que vous venez de faire est de dire à *qftestJUI* que s'il ne voit pas une valeur pour **select** lorsque la procédure est appelée (c'est-à-dire, elle n'a pas été passée à la procédure par l'appel de procédure), alors la valeur par défaut est utilisée. Si une valeur est fournie comme un argument passé à la procédure, alors *qftestJUI* ignorera la valeur par défaut pour utiliser celle fournie.

Conseil : Pour ceux qui se sont familiarisés avec les variables, vous comprendrez la portée d'une variable. Une variable pourrait être, par exemple, définie dans une tâche et alors utilisée de manière erronée dans une autre si vous oubliez qu'elle est déjà définie. A ce stade du développement d'une suite de test, vous ne devriez pas rencontrer ce type de problème, toutefois vous pourriez vouloir en savoir plus sur les variables (merci de vous référer alors au ***Manuel Utilisateur***).

6.7 Construction If/Else

La variable **select** nous aide à décider si une case à cocher est sélectionnée ou pas. Nous en sommes maintenant au point où nous pouvons utiliser cette variable dans la construction logique **if/else**.

Utiliser le mécanisme **if/else** est facile. Pour commencer, une condition est donnée. Si cette condition évalue une déclaration à **true**, alors les nœuds sous le bloc **if** sont exécutés. Autrement (la déclaration est **false**), les nœuds sous le bloc **else** sont exécutés.

Notre condition est, bien sûr, la valeur de la variable select. Si elle est paramétrée à **true**, alors nous sélectionnons la case à cocher, etc.

Ouvrez et cliquez sur le nœud de procédure **setState** de sorte que vous puissiez insérer un nouveau nœud comme premier élément de la procédure. Maintenant sélectionnez le menu **Insert -> Control structures -> If**. Une fenêtre de dialogue à remplir avec les propriétés du nœud s'affiche. Pour ce nœud, vous devez remplir la condition pour évaluer la variable **select** avec précision comme montré ci-dessous :

Figure 6.5 - Déclaration de Condition pour le nœud If

Rappelez-vous que se référer à une variable implique l'utilisation de la syntaxe "**$(variable)**". Les guillemets simples et doubles dans la condition sont la manière de *qftestJUI* d'évaluer les chaînes. Une syntaxe différente est requise pour différents types de conditions comme des expressions mathématiques. Référez-vous aux ***sections Variables et If control node dans le Manuel Utilisateur***.

Si la condition évalue à **true**, alors les nœuds sous le nœud **If** sont exécutés. Vous pouvez maintenant insérer la logique qui fera que la case à cocher sera sélectionnée. Mais comment faire? Il y a plusieurs manières : vous pouvez, par exemple, copier les nœuds depuis le procédure **select** à cette partie de **setState**. Toutefois, nous opterons pour la solution la plus facile en faisant un appel à la procédure **checkbox.select**, ici : (Cf. section 5.8 du chapitre 5).

Maintenant nous en venons à la partie **Else** de la construction. Après le dernier nœud dans notre bloc **if** (vous en avez seulement un, mais vous pourriez en avoir plus), vous pouvez insérer le nœud **Else** en utilisant le menu **Insert -> Control structures -> Else**. Vous n'êtes pas obligé(e) de remplir les informations dans la fenêtre de dialogue de propriétés qui s'affiche pour le nœud **Else**. C'est automatiquement supposé être l'antithèse logique du nœud **If** correspondant.

Enfin, nous finalisons la construction par l'ajout du/des nœud(s) devant être exécuté(s) quand la condition du nœud If n'est pas vraie. Donc ici nous insérons un appel à la procédure **checkbox.deselect**.

La procédure **setState** est maintenant finalisée ! Voici à quoi elle doit ressembler :

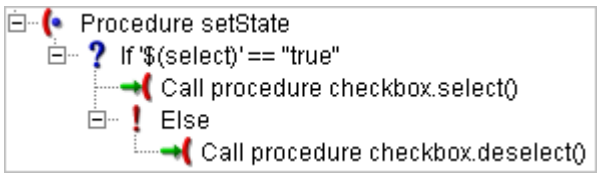

Figure 6.6 - La Construction If/Else

Conseil : Pour notre exemple, la nature booléenne de la variable select se prête bien à une construction **if/else** simple. Cependant, pour d'autres exemples, tels qu'une condition multi-états comme les couleurs d'un feu tricolore, vous pouvez implémenter la construction **if/elseif/else** plus avancée. Le concept est le même, vous êtes seulement autorisé(e) maintenant à évaluer des conditions multiples. Par exemple, votre logique pourrait fonctionner comme ceci :

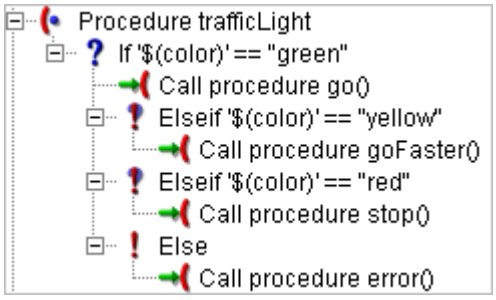

Figure 6.7 - Construction If/Elseif/Else

Veuillez noter que la partie **Else** de la construction if/else ou if/elseif/else est facultative !

6.8 Appel de la Procédure Polyvalente

Maintenant vous pouvez essayer d'appeler votre nouvelle procédure **setState**. Créez un nœud d'appel de procédure, comme vous l'avez fait en section 5.8 du chapitre 5). Vous aurez besoin de la valeur **id** comme argument pour l'appel de procédure, mais maintenant vous pouvez aussi (facultatif) ajouter la seconde variable **select**, à l'appel, comme suit :

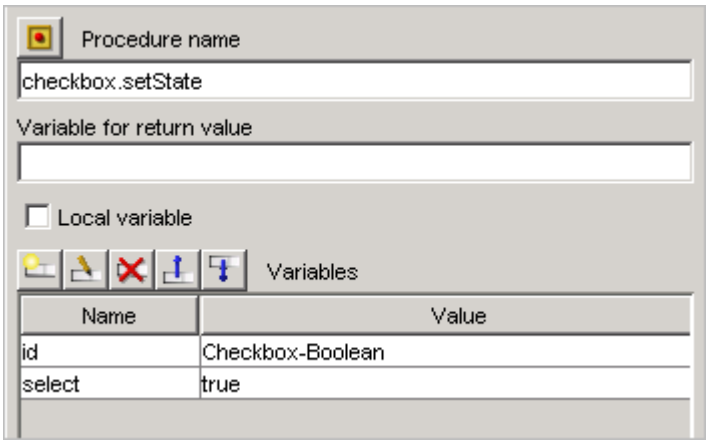

Figure 6.8 - Appel de Procédure à setState

Remarque : Parfois vous pouvez vouloir créer une procédure qui retourne une valeur, c'est-à-dire une procédure **getState**. A cette fin un nœud de retour peut être utilisé dans la procédure. La valeur retournée peut alors être attribuée à une variable globale ou locale en dehors de la procédure, utilisant l'attribut **Variable for return value** du nœud **Procedure** call visible dans la figure ci-dessous. Pour plus d'informations, merci de vous référer au ***Manuel de Référence***.

6.9 Documentation de la Procédure

Maintenant que vous avez finalisé l'implémentation de votre procédure avec succès, nous allons vous montrer une manière commode pour créer la documentation. Pour le langage de programmation Java, il existe une manière standard pour documenter le code source - cela s'appelle ***javadoc***. Cet utilitaire sait analyser votre code source Java et en extraire des informations pour générer une documentation, assez riche et conviviale, au format HTML. Outre les informations purement liées au code Java, javadoc sait récupérer des commentaires que vous vous devez d'adjoindre à votre code.

qftestJUI offre un mécanisme similaire pour documenter vos packages et procédures. Dans la suite de ce document, nous vous montrerons comment cela est fait. Si vous voulez jeter un premier coup d'œil sur le résultat produit, veuillez regarder le fichier **qfs.html**.

Maintenant, nous voulons commencer à préparer notre procédure pour la documentation. Merci d'ouvrir le nœud procédure **setState**, s'il n'est pas déjà ouvert. Comme vous pouvez le voir dans la figure ci-dessous, la procédure a deux paramètres. Pour les documenter, nous ajoutons deux lignes dans le champ **Comment** du nœud, commençant par **@param** suivi du nom du paramètre et une description appropriée. **@param** est un tag spécial pour indiquer la relation de la description aux paramètres de procédure. Une description générale de la procédure peut être ajoutée au début du champ **Comment**. Remarquez que c'est possible d'utiliser des balises HTML pour formater le texte :

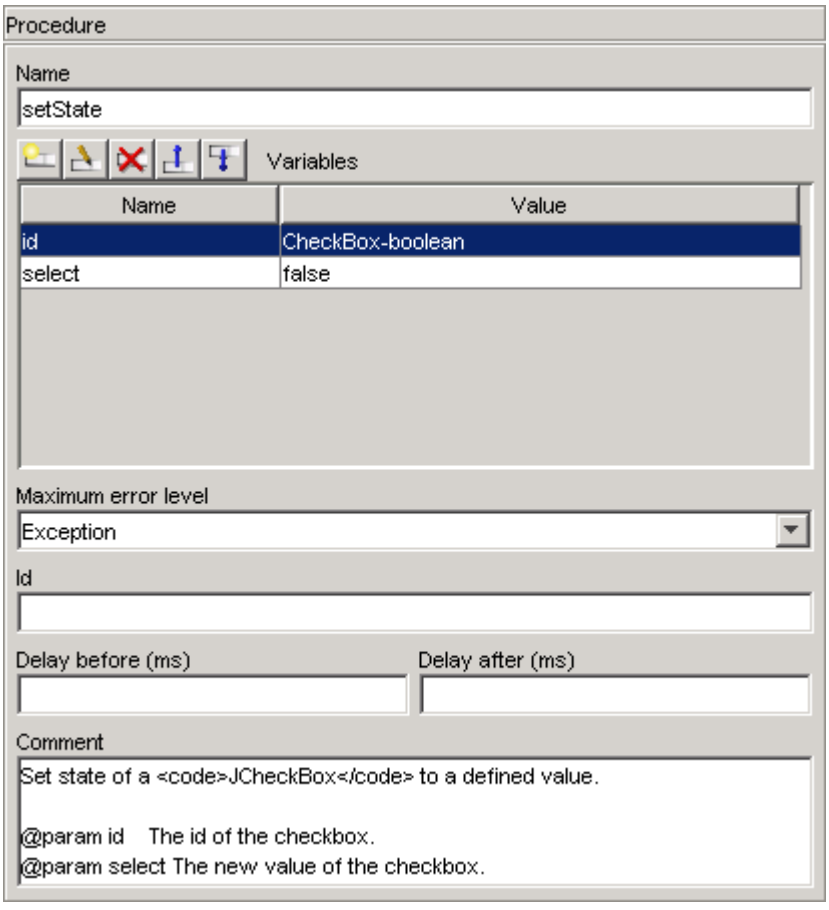

Figure 6.9 - Commentaires pour la Documentation Package

Merci d'ajouter des commentaires similaires pour votre procédure **setState** pour que nous pussions commencer à générer la documentation.

Le processus de génération est très facile. Sélectionnez le menu **File -> Create HTML/XML pkgdoc...**

Vous devez confirmer le dialogue s'affichant qui contient quelques options sur la génération de document sans changement en appuyant sur le bouton **OK**. Alors votre navigateur par défaut devrait apparaître automatiquement (si le navigateur est déjà ouvert, alors cliquez sur la fenêtre) et afficher le contenu suivant :

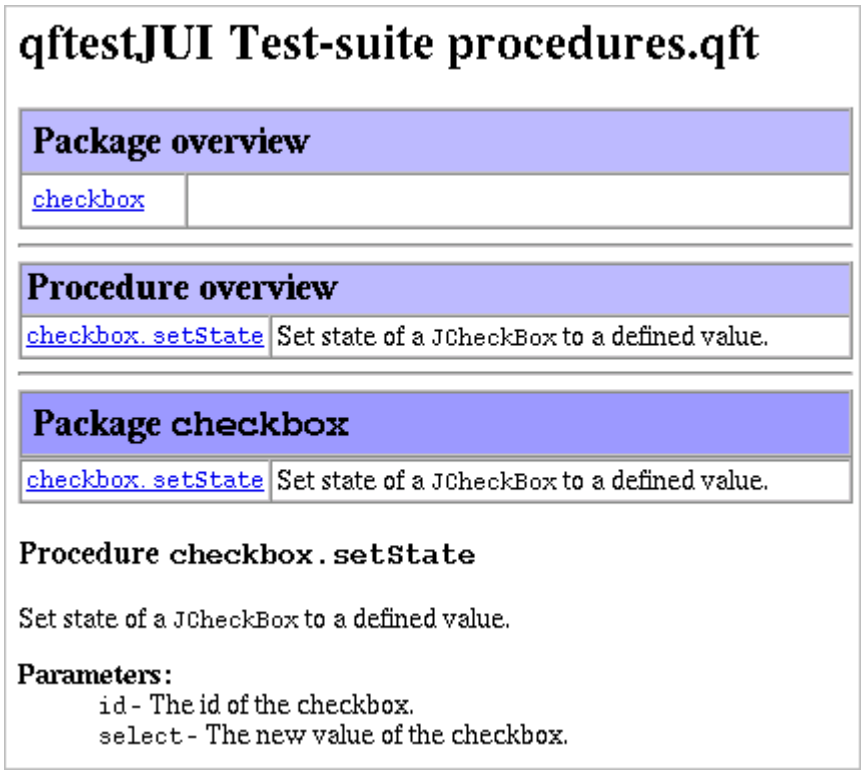

Figure 6.10 - Documentation Package

Ici nous voyons trois parties dans le document. La présentation générale du package (**Package overview**), la présentation générale de la procédure (**Procedure overview**) et la description détaillée du package case à cocher (**Package Checkbox**). Dans le package checkbox, il y a la procédure setState avec son texte de description générale et ses paramètres.

Avec le dispositif de documentation de package, vous pouvez facilement générer une description utile des packages et des procédures, plus particulièrement si vous voulez fournir une fonctionnalité dans une sorte de bibliothèque. Une telle bibliothèque incluant de la documentation est également fournie avec *qftestJUI*. C'est expliqué de façon détaillée dans le Chapitre 8.

Maintenant vous pouvez jouer avec la fonctionnalité de génération de documentation. Vous pouvez par exemple ajouter une description générale au package **checkbox** et essayez des balises supplémentaires, comme **@version**, **@author**, **@result**, **@throws**. Merci de vous référer au Manuel Utilisateur pour la liste complète des balises.

Commentaire : Vous pouvez également utiliser *qftestJUI* pour créer automatiquement la documentation de package en mode batch. Merci de vous référer au ***Manuel Utilisateur*** pour plus de détails sur les arguments de lignes de commande applicables.

6.10 Sauvegarde de la Suite de Test

A ce point, vous êtes en bonne voie pour développer des suites de test évoluées. Vous pouvez sauvegarder le travail effectué jusque là. La sauvegarde d'une suite de test peut être effectuée simplement à partir du menu **File -> Save**. Vous

pouvez nommer la suite de test comme vous voulez, mais nous vous suggérons d'utiliser le nom de **utils.qft**, dans la mesure où nous allons nous y référer ultérieurement et modifier cette suite de test dans le prochain chapitre.

Remarque : Si vous évaluez *qftestJUI* en mode démo, alors vous ne pouvez pas sauvegarder la suite de test. Merci de nous contacter pour obtenir une version d'évaluation pleinement fonctionnelle (sales@kapitec.com).

Dans le chapitre suivant, nous allons commencer à explorer un nombre plus important des possibilités de *qftestJUI* en créant une bibliothèque modularisée de votre suite de test **utils.qft**. C'est également le bon moment pour arrêter le didacticiel au cas où vous voudriez explorer des fonctionnalités de *qftestJUI* par vous-même (à l'aide du Manuel utilisateur), et revenir plus tard au didacticiel.

CHAPITRE 7 - MODULARISATION

Dans ce chapitre, nous allons approfondir les concepts étudiés dans les précédents chapitres. Notre objectif est de vous montrer comment vous pouvez modulariser les suites de test. Dans la mesure où les caractéristiques que nous allons présenter ici sont plus avancées que celles étudiées précédemment, il est important que vous ayez bien compris les concepts étudiés précédemment.

Alors que vous commencez à développer des suites de test spécifiques à vos besoins, vous vous trouverez souvent dans des situations dans lesquelles vous effectuez les mêmes étapes à plusieurs reprises, ou de façon redondante. La suite **A**, par exemple, peut souvent nécessiter l'utilisation d'opération de menu dans votre SUT dont la suite **B** a également besoin. Au lieu d'avoir la même procédure dans chacune des suites, il est beaucoup plus utile et fiable, d'avoir un source commun pour la procédure.

Cela nous amène à introduire le concept de la suite d'utilitaire (ou de bibliothèque) qui contient des procédures communes pour l'utilisation par n'importe quel nombre d'autres suites de test. Votre **utils.qft** que vous avez sauvegardé dans la section précédente (Cf. section 6.10 du chapitre 6), va donc servir comme une sorte de bibliothèque à laquelle les autres suites de test peuvent accéder pour les procédures d'utilitaires communes. L'avantage d'une telle structure est que vous avez une bibliothèque de routines stables qui peut être utilisée pour effectuer leur fonction précisément par d'autres suites de test. Une telle modularisation de suites est facile à implémenter dans *qftestJUI*.

7.1 Création de la Suite Pilote

Ouvrez maintenant une nouvelle suite avec le menu **File -> New Suite**. Avec cette suite de test, vous allez créer un test réel pour le SUT.

Vous devez également ramener votre suite de test **utils.qft**, qui contient le package **checkbox** que vous avez écrit dans le chapitre précédent.

Sauvegardez la nouvelle suite pilote dans le même répertoire que **utils.qft**. C'est nécessaire, ainsi vous pouvez utiliser des noms de chemins d'accès relatifs quand une suite de test se réfère à une autre. Le nom du fichier n'a pas d'incidence.

7.2 Appartenance de la Suite d'Utilitaires

Si l'application SUT n'est pas déjà en exécution, démarrez-la. Vous remarquerez que les nœuds de démarrage pour le SUT sont encore situés dans **utils.qft**. Une vraie suite d'utilitaires ne devrait avoir aucune connexion directe à un SUT, tout simplement parce que la suite d'utilitaires peut être utilisée par d'autres suites de test qui testent différents SUT.

Par exemple, disons que vous avez deux SUT complètement différents, **A** et B, que vous devez tester. Aussi vous voudriez créer deux suites de test pour chaque SUT. Mais, dans chacune des applications SUT, des cases à cocher sont présentes, qui peuvent être sélectionnées ou pas en utilisant votre suite d'utilitaires **utils.qft**. **utils.qft** doit donc être suffisamment polyvalente pour gérer ce genre de situation.

utils.qft contient des éléments qui les lient directement au SUT Options Demo, que nous avons créé en section 5.1, c'est-à-dire les nœuds **Setup/Cleanup** tout comme la section **Windows and components**. Cette information spécifique au SUT est maintenant sans rapport avec la suite d'utilitaires et doit être déplacée dans la suite de test qui testera directement le SUT.

Aussi, de la même manière qu'en section 5.1 du chapitre 5, vous pouvez maintenant déplacer les nœuds **Setup/Cleanup** et le contenu du nœud **Windows and components** dans votre nouvelle suite. Utilisez bien le menu **Couper**, et non pas **Copier**, pour que les nœuds soient bien enlevés de **utils.qft**. Une fois effectuée, votre nouvelle suite de test doit ressembler au squelette que nous avons vu sur la figure 5.1. Vous pouvez aussi nettoyer les nœuds de l'appel de procédure que vous avez créés dans **utils.qft**.

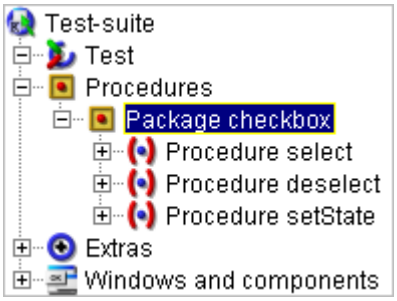

Figure 7.1 - Suite Utilitaire utils.qft

7.3 Création d'une Séquence de Test

Dans cette section, vous utiliserez votre nouvelle suite de test pour créer une séquence de test pour le SUT. La séquence de test sera simple : sélectionnez la fenêtre **Miscellaneous** du SUT et décochez la case **Boolean option**.

Après le nœud Setup dans votre suite de test, insérez un nœud séquence avec le menu **Insert -> Sequence nodes -> Sequence**. Donnez-lui un nom tel que **Deselect checkbox**. Ouvrez le nœud pour pouvoir y insérer de nouveaux nœuds.

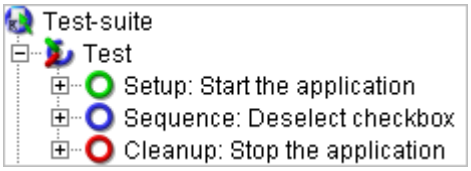

Figure 7.2 - Désélection du Nœud Séquence Checkbox

Maintenant cliquez sur le bouton d'enregistrement  de votre suite de test afin d'enregistrer une séquence. Ramenez la fenêtre du SUT et cliquez sur l'élément **Miscellaneous** sous la liste **Preferences**. Vous voyez s'afficher la sous-fenêtre **Miscellaneous Options** dans le SUT. Retournez à votre suite de test et cliquez sur le bouton  pour arrêter l'enregistrement.

Dans la séquence enregistrée qui apparaît dans le nœud **Extras**, vous voyez seulement un événement enregistré, à savoir le clic de l'élément **Miscellaneous**. Déplacez ce nœud dans votre nouveau nœud séquence, ce qui devrait vous donner ceci :

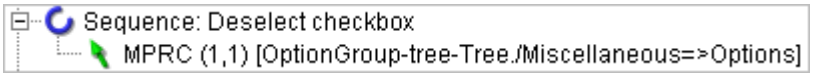

Figure 7.3 - Nœud Séquence avec le Clic Enregistré

7.4 Appel d'une Procédure dans la Suite Utilitaire

L'étape suivante dans la séquence du test consiste à faire un appel à la suite utilitaire pour désélectionner la case à cocher. Comme vous l'avez fait en section 6.8 du chapitre 6, insérez un nœud appel de procédure à la procédure **checkbox.setState**. Cette fois, toutefois, nous avons besoin de procéder à une légère modification, dans la mesure où la procédure est dans une autre suite de test.

Les appels de procédure dans *qftestJUI* peuvent être précédés du nom de la suite (**suite#procedure**) pour indiquer que la procédure cible est située dans la suite de test fournie. Pour notre exemple, l'appel de procédure est correctement déclaré **utils.qft#setState**. N'oubliez pas de renseigner les variables comme cela a été fait en section 6.8 du chapitre 6, avec **select** positionné à **false**.

Figure 7.4 - Appel de Procédure à Utils.qft

7.5 Ajout d'un Include

Au lieu de faire des appels de procédure qui incluent une référence explicite à la suite de test comme avec **utils.qft#setState**, *qftestJUI* fournit un mécanisme simple qui vous permet d'***inclure*** certaines suites de test comme partie de votre structure globale de suite de test.

Un **include** est une référence implicite à une suite de test. Quand *qftestJUI* voit un nœud appel de procédure sans une référence explicite à une suite de test (c'est-à-dire qu'il n'y a pas de nom de suite de test précédant le symbole # devant le nom de la procédure), alors il essaie de chercher dans la suite active la procédure appelée. Si la procédure n'est pas trouvée dans la suite, alors la recherche se fait dans la liste d'**include** de la suite de test.

Cliquez sur le nœud racine **Test-suite** de votre suite de test. Dans les propriétés de ce nœud, vous verrez une section pour **include files**. Dans cette liste, ajoutez maintenant le nom de votre suite utilitaire : **utils.qft**. Les entrées sont ajoutées et modifiées dans cette liste comme vous l'avez fait avec les variables en utilisant les petits boutons ajouter, éditer et supprimer. Vous devez obtenir ce résultat :

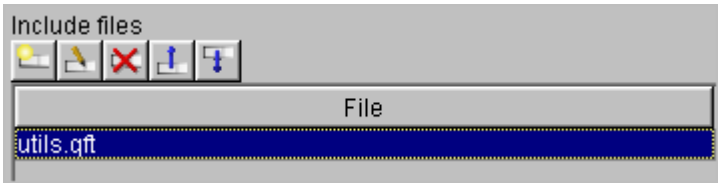

Figure 7.5 - Liste Include

Conseil : Pour des informations détaillées sur la manière dont *qftestJUI* résout des problèmes concernant les appels de procédure, merci de vous référer à la ***section Include File Resolution du Manuel Utilisateur***.

7.6 Modularisation pour Plusieurs SUT

Si vous vous rappelez la discussion en section 7.2, une suite utilitaire telle que **utils.qft** ne doit pas contenir de références directes au client SUT. Si vous regardez bien **utils.qft**, vous verrez qu'il y a en effet encore de telles références à **Options Demo**. Jetez un coup d'œil à l'un de vos nœuds **Check selected state**, par exemple, et vous verrez la référence au client :

Figure 7.6 - Référence Constante au Client SUT

Cette référence directe, bien sûr, vous empêche d'utiliser **utils.qft** avec d'autres suites de test, mais la solution est simple. Au lieu de référencer directement le SUT, vous remplacerez la référence avec une variable qui peut (et doit) être positionnée avec la suite de test qui utilise **utils.qft**.

Partout où vous voyez une référence au client **Options**, remplacez-la avec une référence variable, appelons-la **$(client)** comme montré ci-dessous :

Figure 7.7 - Référence Variable au Client SUT

Vous devez changer toutes les références au client dans **utils.qft**, mais *qftestJUI* vous facilite cette tâche avec la fonctionnalité recherche/remplacement. Cliquez sur le nœud **Test** de haut niveau de la suite de test et sélectionnez le menu **Edit -> Replace**.

L'étape suivante consiste donc à modifier le pilote de la suite de test pour qu'une valeur de la variable client soit disponible quand des procédures à l'intérieur de **utils.qft** sont utilisées. Une façon de procéder est de passer la variable client comme un argument pour chaque appel de procédure dans **utils.qft**. Ce type de solution est parfaitement viable.

Une autre solution, plus élégante, consiste à positionner la variable une fois dans votre suite de test, aussi vous n'avez plus à vous en préoccuper. Ouvrez le nœud de haut niveau **Test-suite**. Dans la fenêtre de propriétés de la suite de test, vous verrez une section (au-dessous des includes et dépendances) dans laquelle vous pouvez définir les variables pour la suite de test. Dans cette zone, ajoutez la variable **client** et donnez-lui pour valeur **Options** :

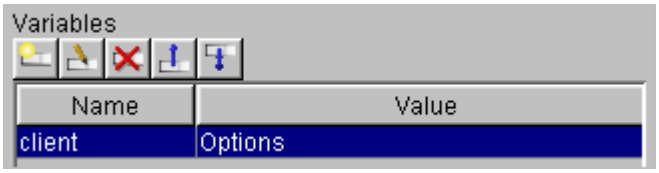

Figure 7.8 - Paramétrage d'une Variable de Suite de Test par Défaut

Une fois que vous avez effectué cette étape, vous n'aurez plus à vous préoccuper de cette variable jusqu'à la création d'une nouvelle suite de test qui utilise **utils.qft**. Votre suite utilitaire est désormais complètement modulaire !

CHAPITRE 8 - BIBLIOTHEQUE STANDARD

Nous finissons sur le thème de la modularisation dans ce nouveau chapitre en présentant une bibliothèque standard de fonctions utilitaires. Vous vous êtes déjà familiarisé(e) avec certains de ces utilitaires, comme les opérations pour traiter les cases à cocher. D'autres utilitaires restent à découvrir. Les sections qui suivent traitent en détail de chacune de ces opérations.

Il y a plusieurs choses à apprendre de ce chapitre. Mais surtout, vous verrez une bibliothèque d'utilitaires exploitables. En fait, la bibliothèque a évolué à partir d'une bibliothèque très similaire utilisée pour les tests de régression de qftestJUI chez Quality First Software GmbH. En étudiant les opérations de cette bibliothèque, vous aurez des idées sur la manière d'améliorer la modularisation de vos propres suites de test et donc de simplifier le processus global de test.

Les procédures dans cette bibliothèque doivent fonctionner avec n'importe quelle application Java/Swing standard ; elles ont été conçues indépendamment de n'importe quel SUT spécifique. Aussi elles doivent s'adapter aux besoins de votre SUT. La bibliothèque est contenue dans le fichier **qfs.qft** et est incluse dans la distribution de *qftestJUI*.

En complément de la description fournie dans ce didacticiel, vous pouvez trouver la documentation HTML de la bibliothèque standard au format javadoc. Le fichier **qfs.html** se trouve dans le répertoire **qftestJUI-1.08.4/include.**

8.1 SUT pour Test

Comme nous l'avons déjà mentionné, la bibliothèque standard que nous introduisons ici n'est pas spécifique au SUT. Cependant, pour des raisons de démonstrations dans ce didacticiel, nous utiliserons une application spécifiquement conçue pour être utilisée avec la bibliothèque standard.

Affichez la suite de test **StdLibDemo.qft**, se trouvant dans le répertoire **qftestJUI-1.08.4/doc/tutorial** du répertoire d'installation *qftestJUI*. Vous devez voir :

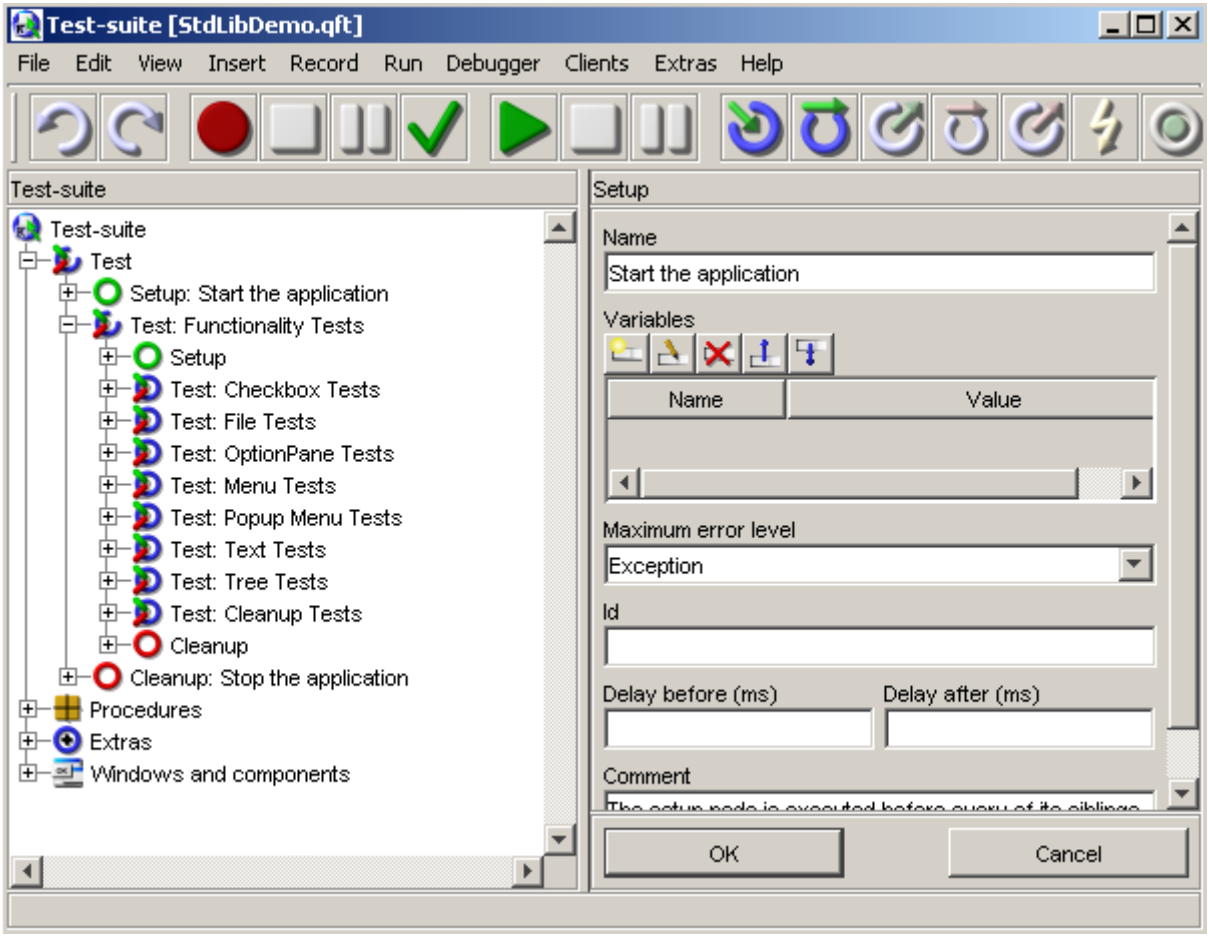

Figure 8.1 - Suite de Test StdLibDemo.qft

Démarrez l'application SUT en exécutant le nœud **Setup**. Après quelques instants, la fenêtre du SUT doit apparaître :

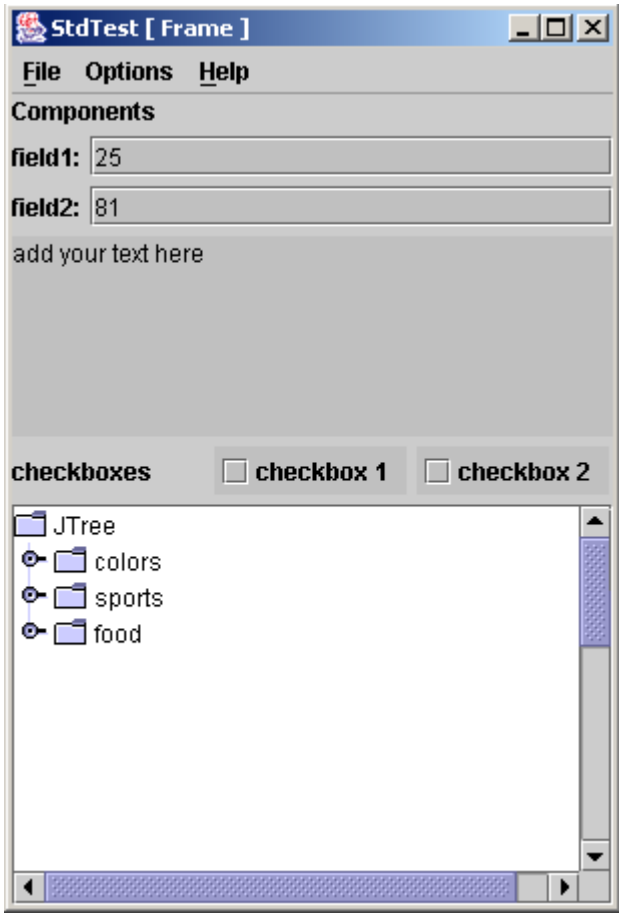

Figure 8.2 - SUT pour le Test de la Bibliothèque Standard

8.2 Bibliothèque Standard

Chargez le fichier de la suite de test **qfs.qft**, qui se trouve dans le répertoire **qftestJUI-1.08.4/include** d'installation de *qftestJUI*.

Ici nous avons un aperçu des packages disponibles :

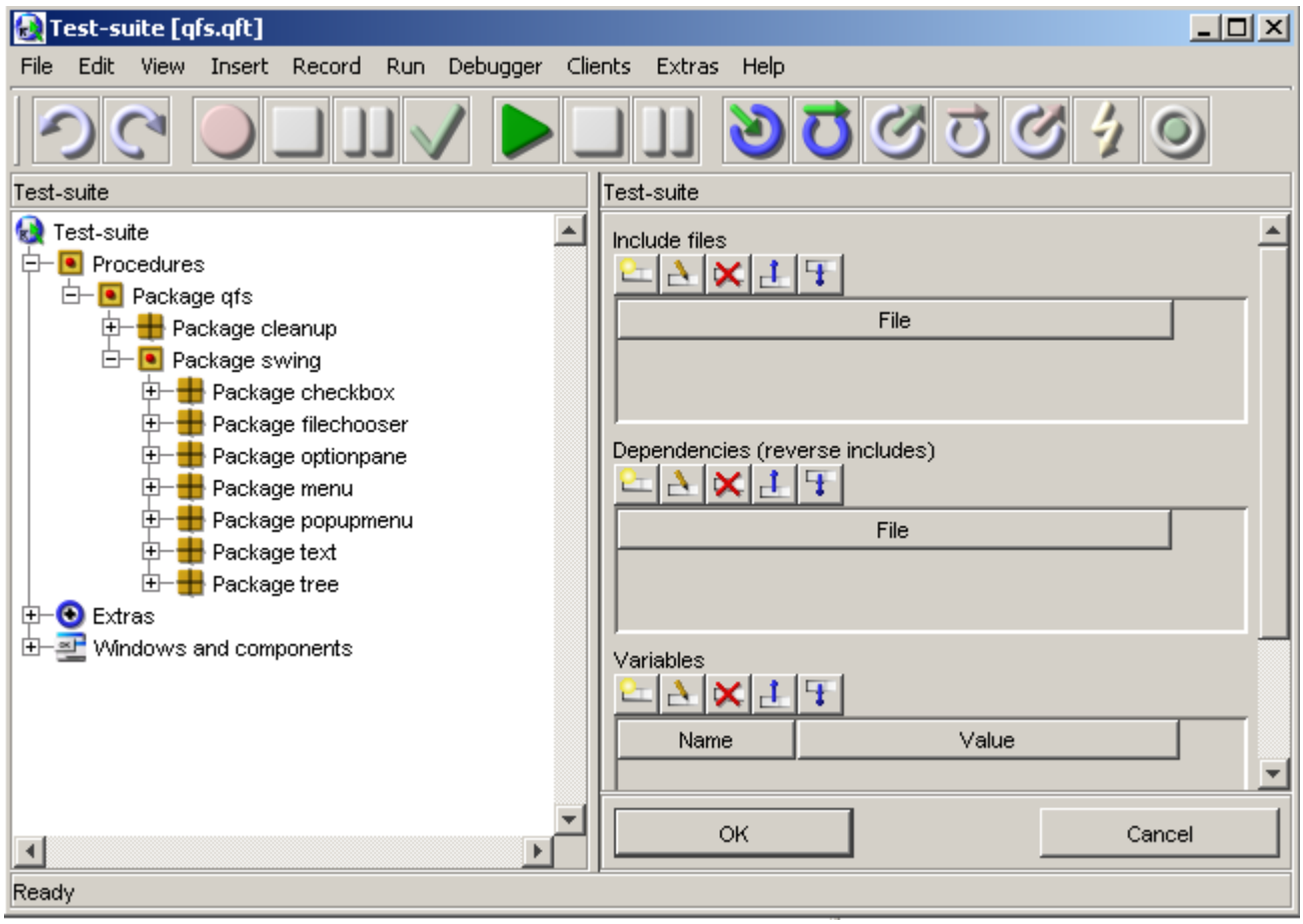

Figure 8.3 - Bibliothèque Standard

La suite contient principalement des procédures pour traiter des composants Java/Swing. Nous y avons également inclus le package **cleanup**, qui est utile pour gérer les effets d'exceptions inattendues se produisant dans une suite de test. Dans un premier temps, nous allons traiter du package **swing**, et ensuite revenir à **cleanup**.

Dans toutes les procédures de cette bibliothèque, vous remarquerez que la variable **$(client)** est référencée. C'est un mécanisme standard que vous verrez souvent répété dans beaucoup de suites de test pour créer l'indépendance depuis un SUT spécifique. Ici, la bibliothèque suppose que la suite de test utilise la bibliothèque qui positionnera une valeur pour **$(client)** avant d'utiliser une procédure. Par exemple, ici nous voyons les propriétés du nœud racine **Test-suite** de la suite **StdLibDemo.qft**, dans lequel nous positionnons la valeur **$(client)** comme une variable de suite. Ainsi, tout nœud exécuté pour cette suite de test (tels que les appels de procédure à **qfs.qft**) auront accès à la variable.

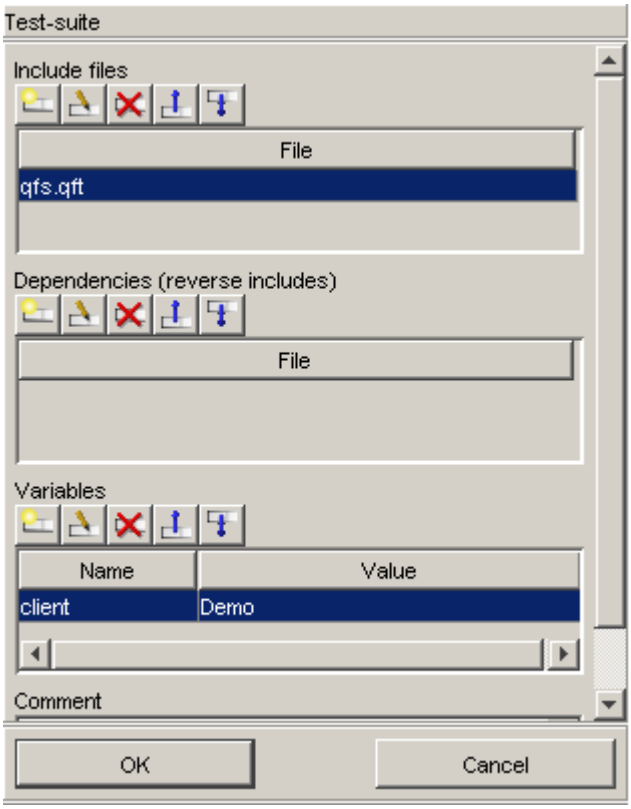

Figure 8.4 - Paramétrage de la Variable $(client)

Vous pourriez aussi positionner la valeur **$(client)** par appel à une procédure dans **qfs.qft**. Un tel schéma peut être utile lorsque vous avez plus d'un SUT. Veuillez remarquer que **qfs.qft** ne peut pas avoir de valeur par défaut positionnée pour la variable **$(client)** car cela le lierait à un SUT spécifique. Il vous appartient de positionner une valeur à la variable **$(client)** pour la bibliothèque standard à utiliser !

Vous avez certainement remarqué sur la figure ci-dessus que nous avons inclus **qfs.qft** dans la suite de test de démonstration sans aucune référence directe au répertoire dans lequel le fichier se trouve. La raison est que le répertoire **/include** de la distribution *qftestJUI* a été ajouté à *qftestJUI* comme un chemin de bibliothèque implicite pour toutes vos suites de test. Cela signifie concrètement que tout fichier suite de test situé dans ce répertoire peut être inclus dans votre propre suite de test.

Pour d'autres chemins de bibliothèque, regardez les options globales de *qftestJUI* via le menu : **Edit -> Options -> General -> Library.**

8.3 Package Utilitaire Case à cocher

Nous commençons par regarder le package **qfs.swing.checkbox**, qui est un bon point de départ car les procédures ici devraient vous être familières si vous avez développé la suite de bibliothèque **utils.qft** décrite au Chapitre 7.

Les procédures disponibles dans ce package sont :

- **select :** Sélectionne (coche) une case à cocher. Si la case à cocher est déjà sélectionnée, alors aucune action ne sera prise.
- **deselect :** Dessélectionne (décoche) une case à cocher. Si la case à cocher est déjà dessélectionnée, alors aucune action ne sera prise.
- **set :** Paramètre une case à cocher dans un état donné (vrai ou faux).

Pour chacune de ces procédures, vous passez l'ID du composant case à cocher comme un argument de variable. La bibliothèque prend en charge la vérification de l'état de la case à cocher, à savoir si elle a été ou non correctement paramétrée tel q'attendu. Pour voir l'usage de ces procédures, regardez dans le nœud **Checkbox Tests** de **StdLibDemo.qft**.

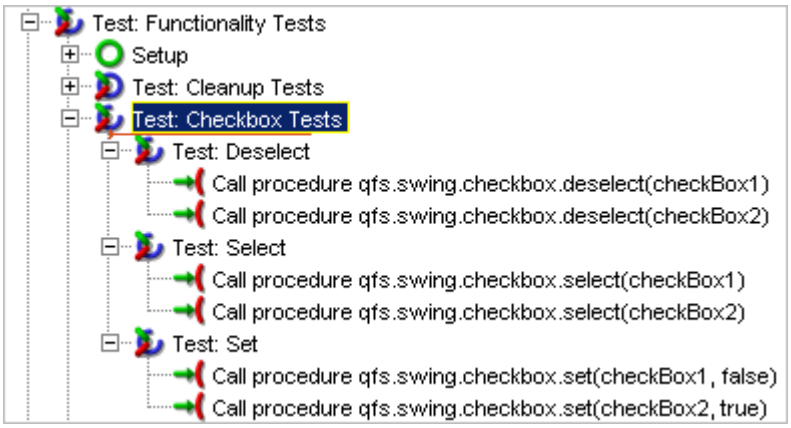

Figure 8.5 - Exemple d'Utilisation du Package qfs.swing.checkbox

L'utilisation des autres procédures dans ce package suit le modèle général vu ici.

8.4 Package Utilitaire Menu

Le package **qfs.swing.menu** vous permet de sélectionner facilement des items et des items de cases à cocher depuis les menus ou les sous-menus. Les procédures sont :

- **selectItem :** Sélectionne un item depuis un menu.
- **selectSubItem :** Sélectionne un item depuis un sous-menu.
- **setCheckItem :** Paramètre l'état d'un item de menu de case à cocher (vrai ou faux).
- **setSubCheckItem :** Paramètre l'état d'un item de case à cocher depuis un sous-menu (vrai ou faux).

Pour chacune de ces procédures, vous devez passer l'ID composant des menus comme l'item et/ou le sous-item pour sélectionner ou cocher : l'utilisation varie légèrement en fonction de la nature de la procédure. Regardez l'exemple d'utilisation du nœud **Menu Tests**. Par exemple, un appel à **setSubCheckItem** ressemble à :

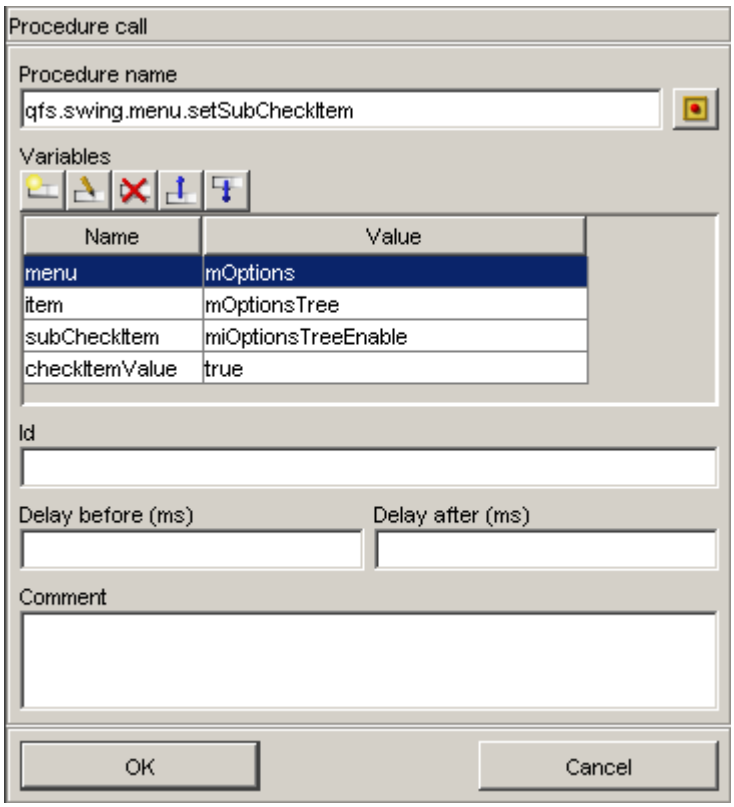

Figure 8.6 - Exemple d'Appel à qfs.swing.menu.setSubCheckItem

Ici nous voyons les arguments de variable requis pour implémenter l'appel à la procédure. Ces arguments sont organisés ainsi :

menu -> item -> subCheckItem (checkItemValue)

qui représente l'implémentation suivante dans le SUT :

Options -> Tree -> Enable (true)

Clearer peut être le processus réel comme vu dans le SUT :

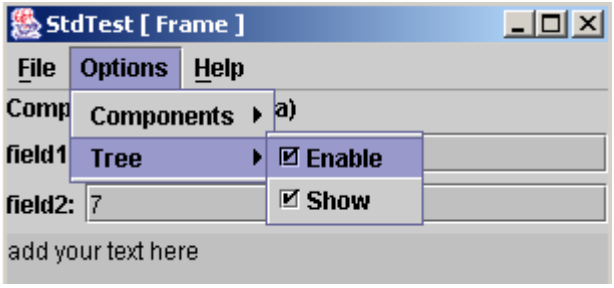

Figure 8.7 - Sélection d'un Item de Sous-Menu Check dans le SUT

8.5 Package Utilitaire Menu Contextuel

Les menus contextuels sont similaires à des menus normaux car ils contiennent des items, des sous-menus et des menus cases à cocher. La principale différence est le moyen par lequel le menu contextuel est invoqué, c'est-à-dire par un clic droit sur un composant qui a été configuré pour afficher un menu contextuel.

Le package **qfs.swing.popupmenu** contient les mêmes procédures que **qfs.swing.menu** ; seule l'implémentation diffère. Pour chaque procédure, votre suite de test doit en premier prendre soin d'ouvrir le menu contextuel avant d'appeler une procédure depuis le package **popupmenu**, car un menu contextuel est un composant spécifique. Tenez compte de l'exemple suivant comme montré dans le SUT :

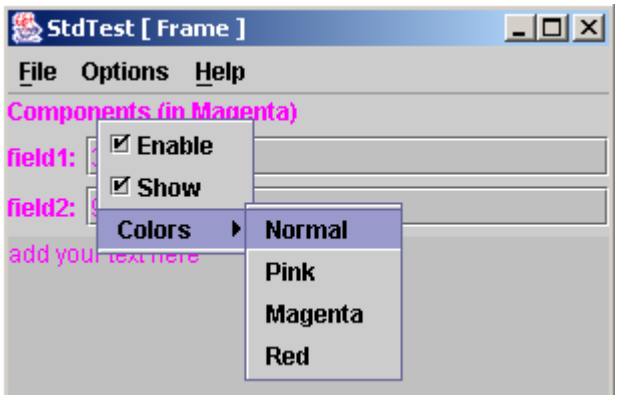

Figure 8.8 - Sélection d'un Item de Sous-Menu de Case à cocher dans un Menu Contextuel

Nous avons implémenté cet exemple au travers du test suivant dans **StdLibDemo.qft**. Le premier nœud affiche le menu contextuel, et alors l'appel à la procédure est fait, suivi par un contrôle des résultats.

Figure 8.9 - Exemple d'Appel à qfs.swing.popupmenu.setSubCheckItem

8.6 Package Utilitaire Tableau

Le package **qfs.swing.table** fournit des procédures utilitaires pour les tableaux. Actuellement il ne contient qu'une seule procédure pour vous aider à redimensionner la largeur d'une colonne d'un tableau. Les valeurs relatives et absolues de largeur sont possibles.

- **ResigeColumn :** Redimensionne la largeur d'une colonne d'un tableau. Le truc dans l'implémentation est le nœud **Fetch Geometry** qui récupère la largeur de la colonne active. L'action de redimensionnement est effectuée en glissant la souris vers la gauche ou la droite.

8.7 Package Utilitaire Texte

Le package **qfs.swing.text** fournit des procédures utilitaires pour les text fields et les text areas, un besoin que vous avez certainement déjà eu lors du développement de suites de test.

- **clearField :** Efface un text field. Cette procédure ne peut pas être utilisée avec un text area.
- **clearArea :** Efface un text area. Cette procédure ne peut pas être utilisée avec un text field.

Remarque : Ces procédures sont maintenant presque superflues, parce que les nœuds **Text input** peuvent effacer le composant cible si nécessaire. Cependant, les procédures peuvent encore servir.

Pour chacune de ces procédures, passez simplement l'ID du composant text field/area. Ici nous voyons un test dans **StdLibDemo.qft** qui se sert du composant **text area** dans le SUT. Dans ce test, premièrement nous effaçons le **text area** et saisissons deux lignes de texte. Alors le **text area** est vérifié pour confirmer qu'il contient le texte attendu. Ensuite, nous effaçons à nouveau le **text area** et vérifions qu'il est vide.

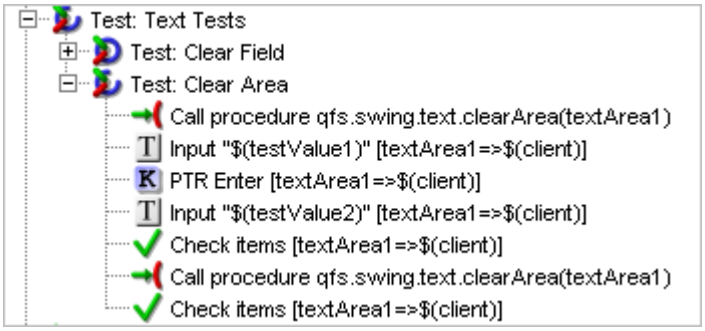

Figure 8.10 - Exemple d'Utilisation de qfs.swing.text.clearArea

8.8 Package Utilitaire Arbre

Nous avons fourni quelques procédures simples d'accès pour manipuler des arbres dans le package **qfs.swing.tree**. Cela inclut :

- **collapse :** Ferme un nœud d'un arbre.
- **expand :** Ouvre un nœud d'un arbre.

Pour n'importe laquelle de ces procédures, vous passez simplement l'ID du composant du nœud que vous voulez manipuler. Par exemple, ici nous voyons un appel à la procédure pour ouvrir un nœud arbre :

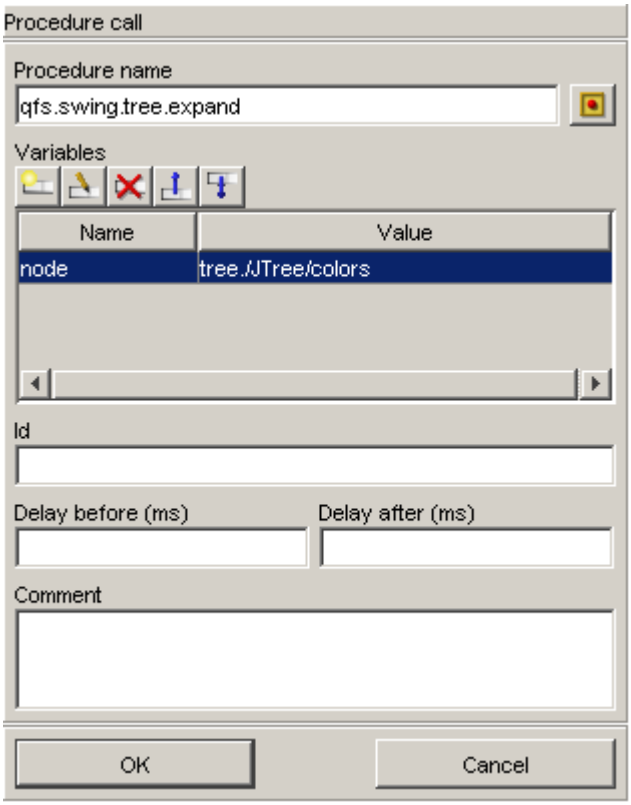

Figure 8.11 - Exemple d'Utilisation de qfs.swing.tree.expand

Remarquez que dans cet exemple, nous utilisons un composant d'un nœud arbre que nous avons enregistré plus tôt. Si vous regardez dans le nœud **Windows and components** de la suite de test **StdLibDemo.qft**, vous verrez les composants **Item tree** individuels que nous avons enregistrés :

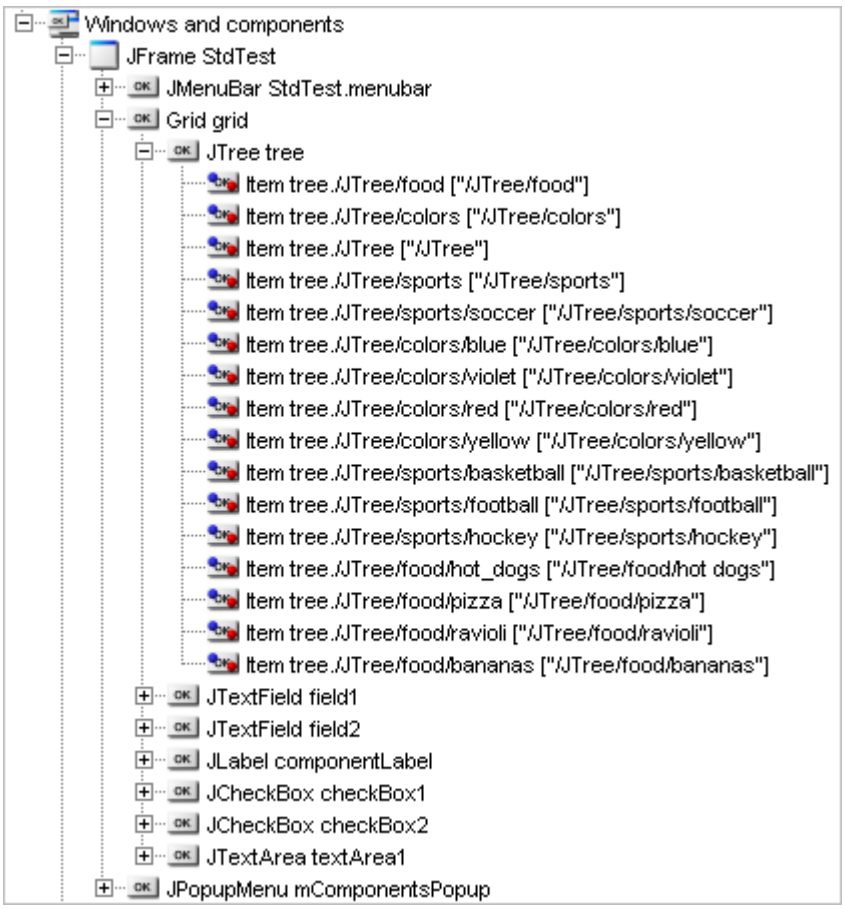

Figure 8.12 - Items Composant d'Arbre

8.9 Package Utilitaire Sélecteur de Fichiers

Le package **qfs.swing.filechooser** présente une légère variation sur les procédures utilitaires que nous avons vu dans cette bibliothèque jusqu'à maintenant. Ce package est destiné non seulement à aider à manipuler tout dialogue de sélecteur de fichiers (le composant **Swing JFileChooser**), mais également à enregistrer et identifier des composants de cette fenêtre de dialogue. Nous listerons d'abord les procédures et ensuite nous entrerons dans le détail :

- **selectFile :** Sélectionne un fichier dans une fenêtre de dialogue sélecteur de fichier.
- **enableNameResolver :** Installe le solveur de noms du **JFileChooser**.
- **disableNameResolver** : Désinstalle le solveur de noms du **JFileChooser**.

La première procédure (**selectFile**) est simple et directe. Vous fournissez simplement le nom du fichier qui sera entré dans **text field file name** du dialogue. Votre suite de test doit en premier gérer l'ouverture du dialogue. Par exemple :

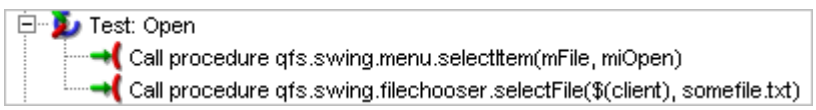

Figure 8.13 - Exemple d'Utilisation de qfs.swing.filechooser.selectFile

Toutefois, utiliser **selectFile** ne fonctionnera pas jusqu'à ce que votre suite de test autorise le solveur de noms ! Nous y venons.

La procédure **enableNameResolver** installe et permet une extension à *qftestJUI* appelé le **JFileChooserResolver**. Ce solveur de noms vous aidera énormément à faire face aux imperfections du dialogue Java **JFileChooser**. C'est un **NameResolver** standard comme l'explique le chapitre intitulé ***Name Resolver Hooks*** dans le ***Manuel de Référence Technique***.

Ce qui est intéressant avec **JFileChooser** c'est que les composants dans les dialogues standards (et en effet la fenêtre de dialogue elle-même) n'ont pas été nommés avec l'opération Java **setName**. Cela peut porter à confusion lors du développement de suites de test, comme les mêmes composants apparaissent dans différentes fenêtres de dialogue. Par exemple, le text field **File Name** sera visible à la fois dans le dialogue de sélecteur de fichier **Save** et **Open**.

Pour *qftestJUI* ce n'est pas un problème. *qftestJUI* a de puissants algorithmes pour enregistrer et identifier des composants, et il ne devrait donc pas confondre un composant sans nom et non unique qui apparaît dans plusieurs fenêtres. La confusion sera la plupart du temps en terme de gestion de composants pour le développeur de suites de test. Regardez la liste de composants enregistrés pour les deux dialogues de sélecteur de fichiers **Open** et **Save** pour l'application **Demo** :

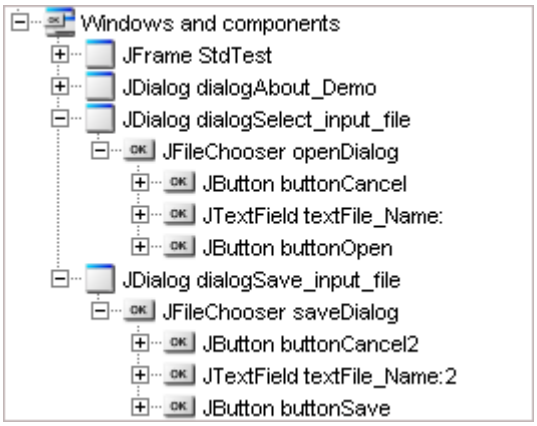

Figure 8.14 - Composants et Dialogues JFileChooser Enregistrés

Comme vous pouvez le voir, *qftestJUI* a correctement identifié les composants et leur a donné à chacun un nom unique, tel que **textFile_Name:2** - qui représente le second du même composant de ce type qui a été vu. Imaginez maintenant que vous avez beaucoup d'autres types de dialogue **JFileChooser**, et chacun avec sa propre liste de composants devant être maintenus séparément. Ecrire des suites de test sous ce type d'environnement peut rapidement devenir lourd.

Saisissez le **name resolver**. C'est une fonctionnalité qui permet à *qftestJUI* d'attribuer des noms à des composants sans nom d'un dialogue **JFileChooser** comme s'ils avaient été nommés dans le code source Java original en utilisant **setName**. Le résultat est que le **JFileChooser** et ses composants peuvent être traités de façon générique. Vous aurez juste besoin d'une liste de composants pour n'importe quel nombre de dialogue **JFileChooser** devant être manipulés dans votre suite de test. Voici un exemple d'utilisation :

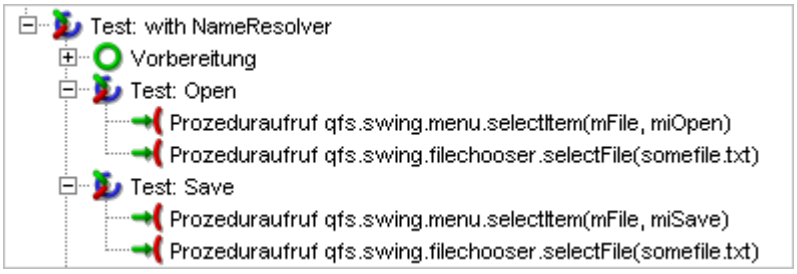

Figure 8.15 - Utilisation du Solveur de Noms

Nous voyons dans l'implémentation de **SelectFile** que les composants génériques sont référencés :

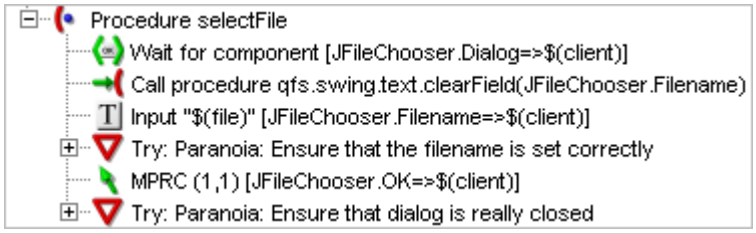

Figure 8.16 - Implémentation du selectFile avec les Composants Génériques

En effet pour votre propre suite de test, vous ne devriez pas avoir à maintenir des composants pour les dialogues **JFileChooser**, si vous utilisez le solveur de noms. Les composants génériques sont stockés dans **qfs.qft** :

Figure 8.17 - Composants FileChooser Génériques dans qfs.qft

Veuillez noter que dans notre suite de test de démonstration **StdLibDemo.Qft**, le solveur de noms est permis et neutralisé pendant l'installation et le nettoyage. C'est abusé et seulement fait pour les besoins de la démonstration. Vous devriez seulement avoir besoin d'activer le solveur de noms une fois dans votre suite de test et de le laisser tel quel. Il n'est pas utile de neutraliser le solveur de noms après qu'il ait été autorisé.

8.10 Package Utilitaire OptionPane

Le package **qfs.swing.optionpane** contient des procédures utilitaires solveur de noms similaires à celles décrites dans le dernier chapitre. **JOptionPane** est le composant Swing utilisé pour créer ces petites boîtes de messages informant l'utilisateur d'erreurs de programme, avertissement ou demande de confirmation.

Le besoin de solveur de noms pour ce composant résulte du fait que ni le dialogue, ni ses sous-composants (boutons de confirmation et étiquettes de texte de message), ne sont nommés. Aussi *qftestJUI* doit les identifier sur la base de leurs titres ou de leurs étiquettes de texte, ce qui résulte en la création de divers nœuds composant pour chaque instance de **JOptionPane**. Ceci peut générer un grand nombre de ces éléments et n'est donc pas souhaitable pour des raisons de clarté et de maintenabilité.

Maintenant regardons les procédures d'interface. Elles sont très simples :

- **enableNameResolver :** Installe le solveur de noms JOptionPane.
- **disableNameResolver :** Désinstalle le solveur de noms JoptionPane.

La procédure **enableNameResolver** installe et permet une extension à *qftestJUI* appelé **JOptionPaneResolver**. Ce solveur de noms facilite la manipulation de dialogues **JOptionPane**. C'est un **NameResolver** standard, comme expliqué dans le ***Manuel de Référence Technique***.

Il est en général suffisant d'installer le **NameResolver** une fois au début de l'exécution du test, par exemple directement après le nœud **Wait for client to connect**. Généralement, la désinstallation n'est pas nécessaire.

Comme mentionné ci-dessus, des composants génériques pré-définis sont fournis dans **qfs.qft**, ce qui vous évite alors de maintenir les vôtres. La figure suivante les montre :

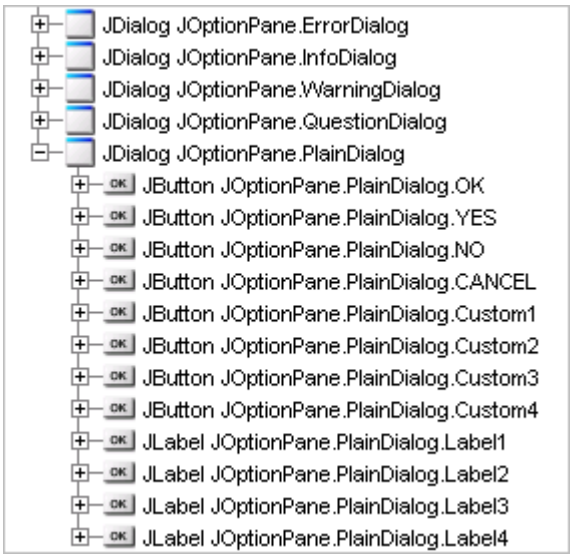

Figure 8.18 - Composants OptionPane Génériques dans qfs.qft

La figure vous montre que 5 types de dialogue **OptionPane** sont supportés :

- Erreur
- Information
- Avertissement
- Question
- Texte simple (*Plain*)

A titre d'exemple le nœud composant **plain dialog** a été ouvert pour vous montrer son contenu. Il contient des nœuds pré-définis pour les différents boutons possibles (boutons standards et personnalisés) et étiquettes, contenant généralement le texte du message

Dans notre suite de test de démonstration **StdLibDemo.qft** vous pouvez trouver un petit test pour ce solveur de noms dans le nœud test **OptionPane** qui teste les deux dialogues **About** disponibles dans le menu **Help** de la Demo. Vous voyez un test avec le solveur de noms et un autre sans. Regardez les composants sous **Windows and components** utilisés pour ce test. Sans le solveur de noms installé, alors deux composants légèrement différents doivent être maintenus. En utilisant le solveur de noms seul un des composants prédéfinis de **qfs.qft** est requis.

Remarque : L'implémentation du solveur de noms décrit dans ce chapitre doit pouvoir mapper la plupart des occurrences des composants **JOptionPane** aux composants prédéfinis dans **qfs.qft.** Cependant, il peut y avoir des instances de **JOptionPane** avec des composants personnalisés à l'intérieur, ayant pour résultat l'enregistrement séparé de représentations de composants.

8.11 Package Cleanup

Pour finir, nous donnons le traitement du package **qfs.cleanup**, qui est utile pour le nettoyage générique de l'environnement SUT après qu'une exception inattendue se soit produite. Imaginons, par exemple, qu'une exception soit générée lors d'une tentative de manipulation d'un menu dans le SUT. L'exception provoquera la redirection d'un chemin d'exécution dans votre suite de test vers un gestionnaire d'exception, ou la gestion d'une exception "implicite". Cela signifie que le flux normal d'exécution qui aurait dû fermer correctement le menu ouvert est maintenant interrompu. Sans action appropriée, ce menu pourrait rester ouvert et ainsi bloquer d'autres événements dirigés vers le SUT.

Voici une liste des procédures disponibles dans ce package :

- **closeAllModalDialogs :** S'assure que les fenêtres de dialogue modales du SUT sont fermées.
- **closeAllMenus :** Ferme tous les menus du SUT inconditionnellement.
- **implicitExceptionHandler :** Utilise comme un gestionnaire générique pour traiter implicitement des exceptions capturées.

Les premières procédures doivent être relativement claires aussi bien en terme de significations que d'objectifs. La procédure **implicitExceptionHandler** en fait inclut l'utilisation des autres procédures dans ce package, ce qui en fait un bon gestionnaire par défaut pour votre suite de test si des exceptions inattendues surgissent.

Le concept de la gestion implicite d'exception est un concept important, et nous allons y consacrer le reste de cette section. Si vous regardez dans les propriétés de n'importe quel nœud **Test**, vous verrez une case à cocher intitulée **implicitly catch exceptions**.

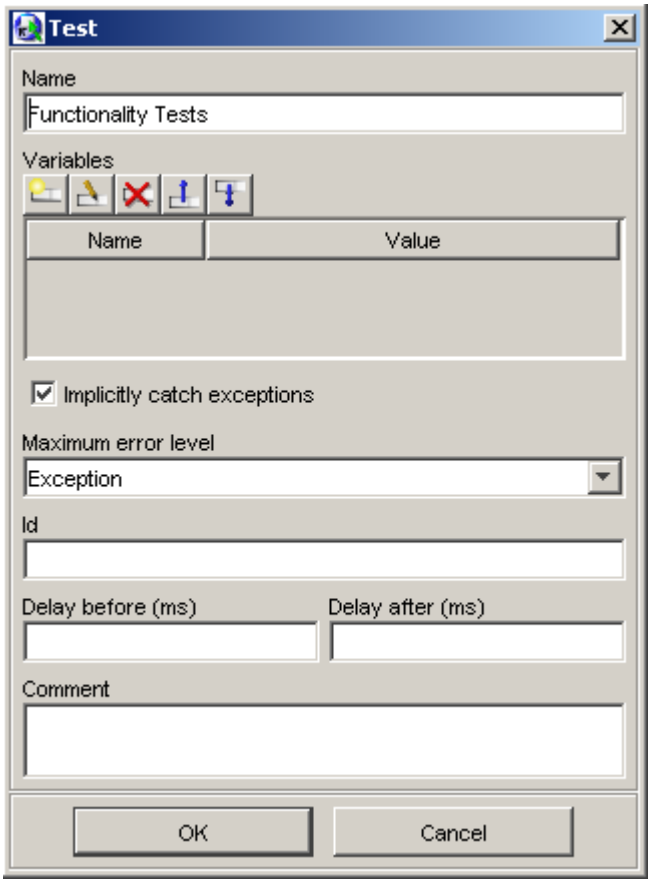

Figure 8.19 - Test Configuré pour Capturer Implicitement des Exceptions

Avec cette fonctionnalité activée dans un test, une exception qui se produit et qui n'est pas capturée explicitement dans un nœud **Catch** sera à la place redirigée vers le nœud **Cleanup** du nœud **Test.** Sans une capture implicite, une exception non capturée arrêtera l'exécution de la suite de test.

Les règles pour la gestion d'exceptions implicites sont simples : si une exception est générée et n'est pas capturée, alors *qftestJUI* cherche le nœud test parent suivant qui a été configuré pour capturer implicitement les exceptions. Il est donc possible de configurer une hiérarchie de gestion de capture implicite.

Remarque : Si un nœud **Test** gère implicitement une exception, son état final sera paramétré à **Exception** pour le refléter. L'erreur sera dûment reportée pour être sûr que l'exception ne passe pas inaperçue.

Dans **StdLibDemo.qft** nous voyons un exemple d'un test qui est configuré pour capturer implicitement des exceptions et dans lequel plusieurs exceptions inattendues se produisent. Le test est sans sens, mais il est utile dans ce cas car il illustre parfaitement cette fonctionnalité. Ici, nous produisons les erreurs en laissant ouverts les dialogues modales et les menus, puis en essayant de cliquer sur la fenêtre principale du SUT, générant ainsi une exception car étant bloquée par le dialogue modal ou le menu.

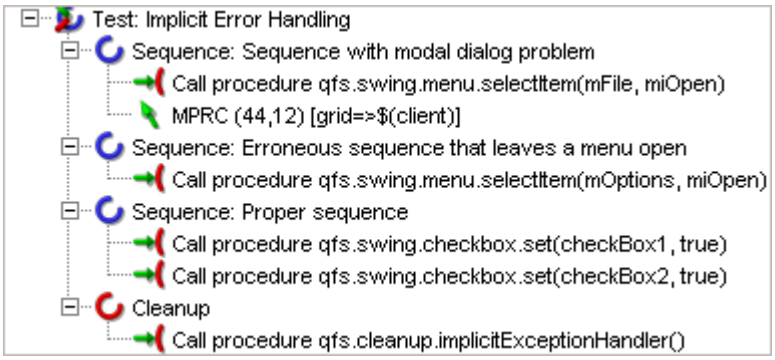

Figure 8.20 - Exemples d'Erreur Implicite

Exécutez le test. Avant de commencer, désactivez temporairement l'option **Edit -> Options -> Debugger -> Break on uncaught exception**. Autrement le débogueur s'ouvrira indépendamment de l'état **Implicitly catch exceptions**. Ce qui est intéressant à voir c'est le comportement, quand le drapeau **Implicitly catch exceptions** n'est pas coché, ou en neutralisant simplement l'appel à **implicitExceptionHandler**. Comme c'est souvent le cas, la vraie compréhension provient d'une première prise en main associée à des informations techniques (comme c'est le cas de ce document), c'est pourquoi nous vous encourageons à essayer ce test et les autres tests dans la suite de test pour visualiser le comportement par vous-même.

CHAPITRE 9 - GESTION DE COMPOSANTS D'IHM COMPLEXES

Jusqu'à maintenant, nous avons traité de composants d'IHM Java dits éléments d'IHM "simples", tels que des boutons radio ou text fields. Nous allons maintenant nous intéresser à des composants plus complexes, qui eux-mêmes contiennent des sous-éléments. Les exemples classiques pour de tels composants sont les arbres et les tables.

Prenons les nœuds d'un arbre. Un nœud n'est pas un composant d'IHM au sens Java, mais une représentation graphique de données. Cette distinction technique n'est pas d'une grande aide pour le test, pour lequel le comportement d'un tel nœud arbre est aussi important que le comportement d'un élément d'IHM plus simple.

Pour gérer ces cas, qftestJUI dispose d'un élément spécial appelé **item** qui représente des sous-éléments complexes. Nous allons donc traiter de la manipulation des items et la syntaxe utilisée par *qftestJUI* pour s'y référer. Une solide compréhension de ces concepts sera nécessaire car nous allons aborder des tests avec des scripts automatisés et alimentés par des données.

9.1 SUT Pour Test

Dans cette section, nous utiliserons une application simple comme SUT qui contient plusieurs éléments complexes. Nous l'avons appelé **ItemsDemo** et elle se trouve dans la distribution de la version courante de *qftestJUI*. Nous laisserons la tâche de création d'une suite de test pour encapsuler le SUT, mais voici un nœud **Start Java SUT client** qui exécutera l'application :

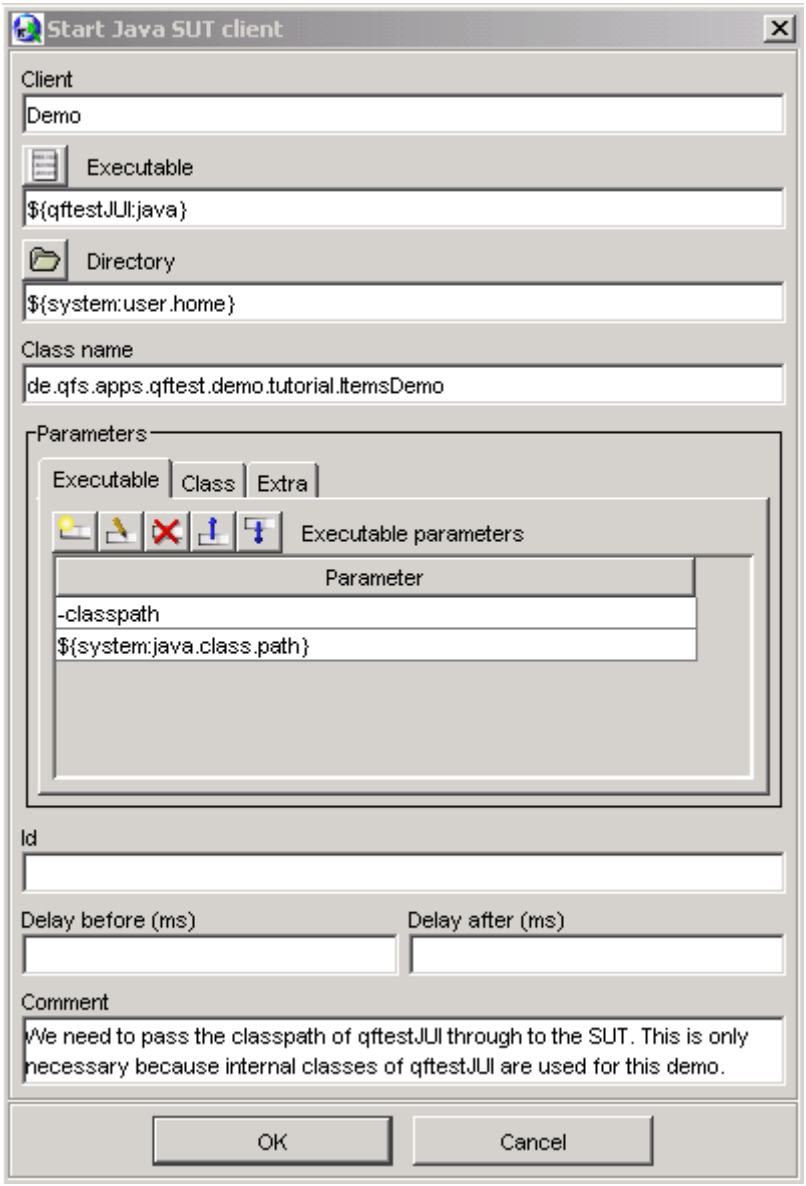

Figure 9.1 - Nœud Start Java SUT client Pour ItemsDemo

Créez une nouvelle suite et ajoutez ce nœud comme partie du nœud normal **Startup**. Le reste de l'infrastructure de la suite de test doit désormais être simple à mettre en place par vous. Vous devez donc voir :

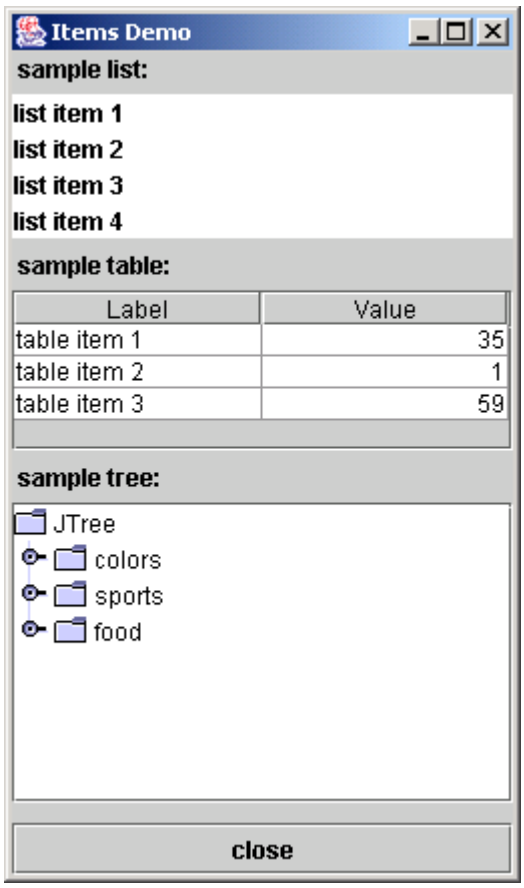

Figure 9.2 - Fenêtre ItemsDemo

9.2 Composants Complexes Unidimensionnels

Nous commencerons par l'analyse du plus simple des éléments complexes, qui est une liste unidimensionnelle, telle que représentée par la classe Java **JList**. Dans la fenêtre du SUT, vous verrez une **JList** comme premier composant, ayant pour étiquette **Sample list**.

Commencez à enregistrer et cliquez sur les éléments de la liste. A l'arrêt de l'enregistrement, vous verrez alors une séquence de ce type :

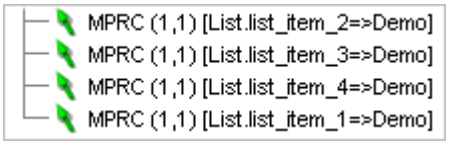

Figure 9.3 - Opérations Souris sur une JList

Cliquez sur un des nœuds **MPRC** enregistrés et regardez le composant qui est référencé.

Figure 9.4 - Référence à un Item JList

La référence est tout ce qu'il y a de plus simple, avec un type de structure parent-fils. Dans ce cas **List** est le nom du composant (la JList), et **list_item_2** est le nom de l'item individuel (enfant) de la liste. Regardons comment ce composant est réellement stockée par *qftestJUI*.

Avec le même nœud encore sélectionné, exécutez **locate component** soit par un clic-droit et en cliquant sur l'option **locate component**, soit en utilisant le raccourci clavier **Ctrl+W**. Vous devez alors voir une liste ouverte des éléments enregistrés du SUT dans la section **Windows and Components** de la suite de test, avec le nœud pour l'item **list** sélectionné :

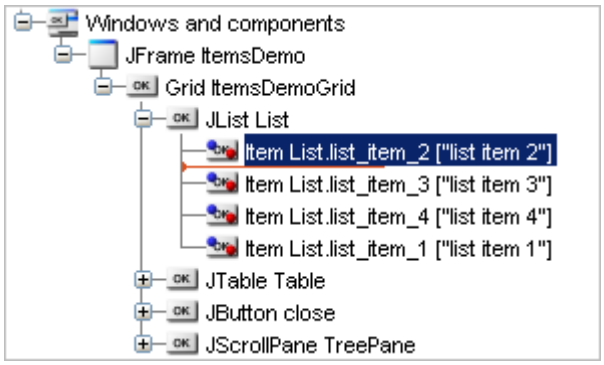

Figure 9.5 - Nœud Item JList

Ce que nous voyons alors est un nœud appelé **Item** qui est spécifiquement utilisé pour décrire les éléments de structures complexes, comme notre liste. Un nœud **Item** consiste en deux composants principaux : le parent à qui il appartient et l'index (ou indices) de(s) item(s) dans le parent. Regardez les propriétés pour ce nœud **item** :

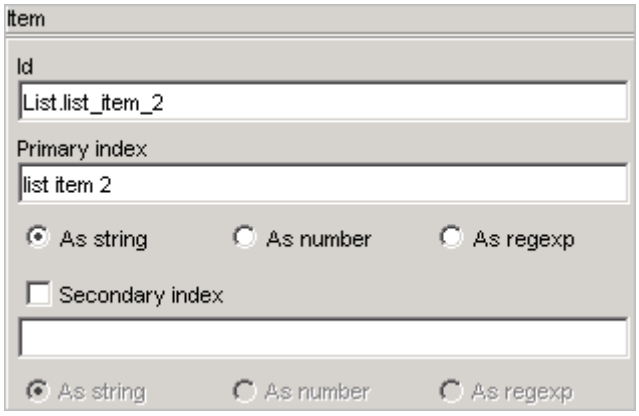

Figure 9.6 - Propriétés de l'Item JList

Ici nous voyons l'ID du composant (contenant la référence au composant parent) et un index primaire. La liste étant une structure unidimensionnelle, nous avons seulement besoin d'un index. Lorsque nous étudierons les tableaux dans la section suivante, l'index secondaire entrera alors en jeu.

Le premier index est décrit ici comme une chaîne de caractères, qui est le texte de l'item dans la liste du SUT. Ce type de mise en œuvre est adapté à la plupart des applications, mais bien sûr peut avoir de fâcheuses conséquences si jamais le texte des items dans la liste change. Alors pour éviter ce type de problème, *qftestJUI* vous offre deux autres possibilités : la première est de référencer l'item en tant qu'index numérique (commençant par l'index 0) de la liste. Veuillez changer l'index list item 2 par l'index numérique équivalent, c'est-à-dire 1. Alors changez l'interprétation de l'index primaire en cliquant sur le bouton radio **As number** :

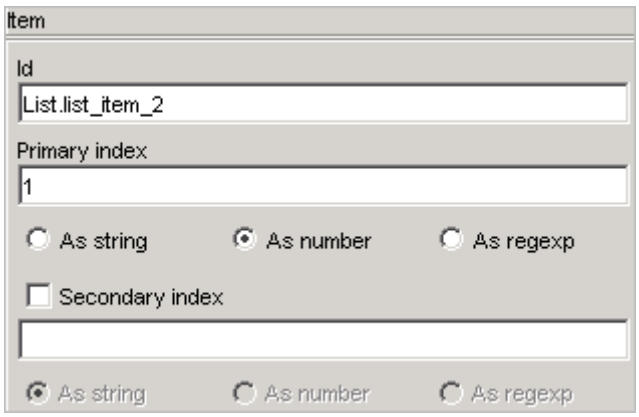

Figure 9.7 - Item JList avec un Index Numérique

Une autre possibilité pour indexer des items est offerte avec l'index d'expression régulière sophistiqué. Ici, *qftestJUI* effectue une recherche textuelle des items dans la liste jusqu'à ce qu'un pattern correspondant à l'expression régulière que vous avez fournie soit trouvé. Par exemple :

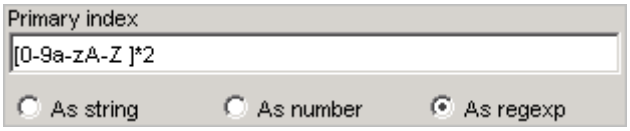

Figure 9.8 - Item JList avec un Index Expression Régulière

L'expression régulière donnée ici indique un nombre arbitraire de caractères alphanumériques ou avec des espaces blancs et avec le chiffre **2** pour terminer. Pour plus de détails sur l'écriture d'expressions régulières, merci de consulter le ***Manuel de Référence Technique***.

Essayez à nouveau d'exécuter votre séquence enregistrée pour la liste, mais maintenant avec des nœuds Item modifiés. Si vous avez correctement modifié les références d'index, alors le comportement de la séquence doit être exactement le même.

9.3 Composants Complexes Bidimensionnels

Maintenant nous allons passer à une structure un peu plus complexe : un composant bidimensionnel le mieux représenté par une classe Java est **JTable**. Un tableau, bien sûr, consiste en des lignes et des colonnes. Donc un item dans un tableau requiert deux indices afin d'être correctement référencé.

sample table:

Label	Value
table item 1	35
table item 2	1
table item 3	59

Figure 9.9 - Sample Table dans le SUT

Comme vous l'avez fait avec la liste, essayez d'enregistrer quelques clics sur les items dans la **sample table** du SUT. Veillez à enregistrer des clics sur les éléments des deux colonnes. Vous devez alors avoir :

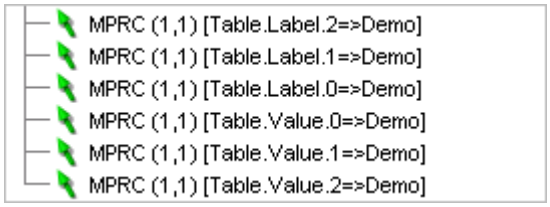

Figure 9.10 - Clics Souris Enregistrés dans la Jtable

Les nœuds doivent être similaires aux nœuds enregistrés pour la structure **JList**. Ici, nous avons simplement un nœud **MPRC** qui référence un composant dans la forme **parent.primaryIndex.secondaryIndex**. **JTable** est le parent de **Table**, **Label** ou **Value** est l'index primaire (colonne) et un nombre représente l'index secondaire (ligne). Utilisez **locate component** et regardez les nœuds Item qui ont été enregistrés :

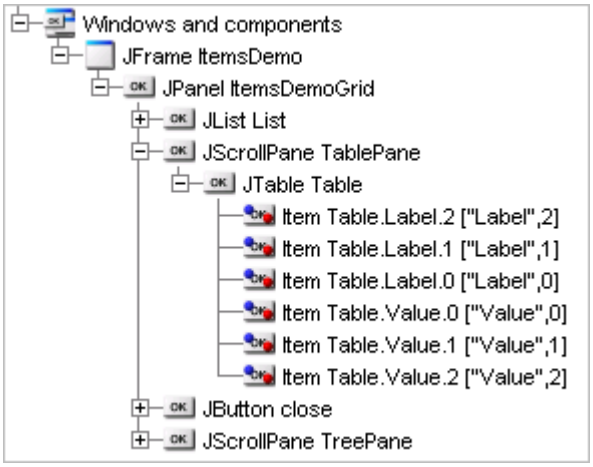

Figure 9.11 - Nœuds Items Enregistrés pour la JTable

En regardant un nœud individuel, vous voyez :

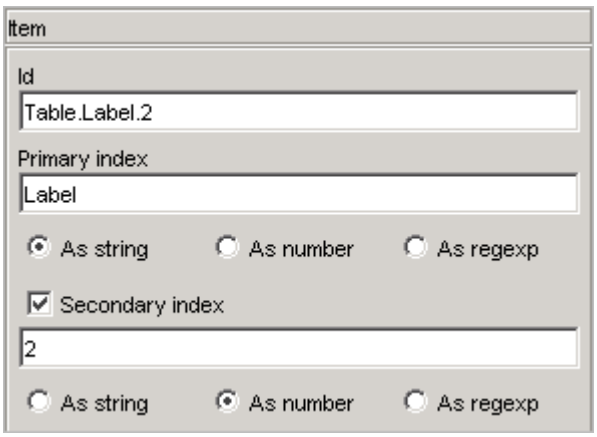

Figure 9.12 - Propriétés d'un Nœud Item JTable

Comme nous nous y attendions, il y a maintenant deux indices utilisés pour référencer l'item dans le tableau. La chaîne de caractères colonne du tableau **Label** est utilisée comme représentation de l'index primaire ou colonne. L'index secondaire est une valeur numérique indiquant la ligne du tableau. Comme les boutons radio le laissent entendre, les deux valeurs peuvent être changées dans d'autres formats, mais gardez à l'esprit que *qftestJUI* utilise par défaut un mode intelligent pour trouver la meilleure représentation pour assurer une identification appropriée de composants.

9.4 Nœuds Item versus Syntaxe

Le chapitre précédent a expliqué la représentation des items dans *qftestJUI*. Mais il existe un autre moyen d'accéder aux éléments de composants complexes. Cela est faisable en référençant l'élément cible par son composant parent étendu avec une certaine syntaxe spécifiant le sous-composant. Ce type de syntaxe est ce que *qftestJUI* appelle un "accès direct" à l'item. Trois types de syntaxe sont possible :

- **@** - textuel
- **&** - numérique
- **%** - expression régulière

La séquence suivante de clics souris utilise cet "accès direct" et fonctionne de la même manière que celle illustrée en figure 9.10. Essayez de comprendre les différentes représentations. Quand la syntaxe est utilisée, les mœuds item ne sont ni requis, ni accessibles.

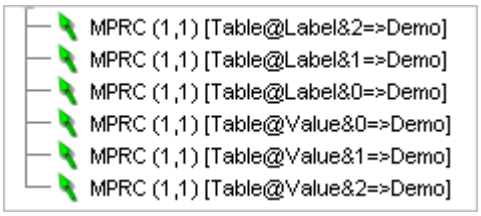

Figure 9.13 - Clic Souris Utilisant la Syntaxe pour l'Accès Direct

Que les sous-items soient représentés comme des nœuds items ou comme syntaxe ne fait aucune différence pour *qftestJUI*. Les deux méthodes ont leurs avantages. Les nœuds **item** sont particulièrement bien adaptés aux éléments statiques alors que la syntaxe est presque toujours préférable, s'il y a un certain degré de variabilité. *qftestJUI* essaie d'enregistrer les sous-items aussi intelligemment que possible, mais dans certains cas il peut être préférable de changer le paramétrage par défaut. En utilisant les options pour l'enregistrement de sous-item (se référer au ***Manuel Utilisateur***), vous pouvez précisément définir comment *qftestJUI* doit enregistrer les sous-items.

9.5 Importance du Caractère Unique

Nous insistons à nouveau sur l'importance du référencement unique d'items dans un composant complexe. Comme la nature du composant augmente en complexité, la complexité de référencement des items du composant augmente. Pour les deux exemples précédents (listes et tableaux), la structure est relativement simple. Toutefois, même avec ces composants, des entrées en doublon peuvent apparaître. Regardez la figure suivante :

expenses and income per year

Year	Expenses	Income
2001	$5061	$8500
2002	$4985	$9400
2003	$6200	$9400

Figure 9.14 - Tableau avec des Items en Doublon

Dans la mesure où des items en duplicata peuvent apparaître, vous devez considérer comment votre suite de test devrait gérer un tel tableau. A savoir, qu'est-ce qui pourra changer et qu'est-ce qui restera statique ? Les réponses déterminent le type de référencement que vous devez mettre en œuvre pour ces items dans le tableau. C'est pourquoi *qftestJUI* utilise le nombre ligne comme index secondaire lors de l'enregistrement de composants en mode intelligent.

Dans la prochaine section, nous nous attaquerons aux composants les plus complexes, dans lesquels la gestion d'items potentiels en doublon est d'une importance critique.

9.6 Arbres

Les arbres sont un cas spécial de composants complexes car leur framework hiérarchique ne se mappe pas à une structure linéaire. Le caractère unique d'un nœud doit également être pris en considération.

Toutefois, étant donné l'ensemble d'outils et de concepts étudiés dans les sections précédentes, *qftestJUI* rend la manipulation des arbres très viable. D'une importance spéciale est l'introduction de l'opérateur /, qui fonctionne comme le séparateur de chemin d'une hiérarchie de fichiers.

Essayons. Enregistrez donc quelques clics sur le **sample tree** du SUT. Vous devriez obtenir quelque chose de similaire à :

Figure 9.15 - Clics Souris Enregistrés avec le JTree

Tout d'abord, essayez d'ajuster et de jouer votre séquence enregistrée pour comprendre comment les arbres sont gérés. Par exemple, en retirant les clics intermédiaires qui ont ouvert le nœud **JTree**, alors les nœuds colors ne sont pas nécessaires si vous voulez simplement cliquer sur le nœud **yellow**. Dans ce cas, *qftestJUI* ouvrira automatiquement l'arbre.

Maintenant regardons la syntaxe pour le clic sur le nœud **/JTree/colors/yellow**. L'opérateur de chemin (' **/**') est mis en utilisation en décrivant le chemin pris par l'arbre pour trouver le bon nœud. Ce type de schéma présente de grands avantages car le problème de caractère unique d'un autre nœud avec le même nœud situé dans une branche différente de l'arbre s'en trouve éliminé. Le seul problème de caractère unique qui reste est qu'il y a des nœuds arbre du même nom situés dans la même branche (c'est-à-dire, il y a deux nœuds **yellow** situés dans **/JTree/colors**). Dans ce cas, nous pouvons revenir à notre mécanisme de référencement numérique d'un nœud.

La modification de l'indexation d'un nœud JTree est la mieux réalisée en utilisant la méthode d'accès direct introduite dans la section précédente. Ici, nous utilisons un des opérateurs **@**, **&**, ou **%** pour permettre respectivement des références directes textuelles, numériques ou expression régulière. Voici à quoi ressemble une référence numérique au même nœud en utilisant l'accès direct :

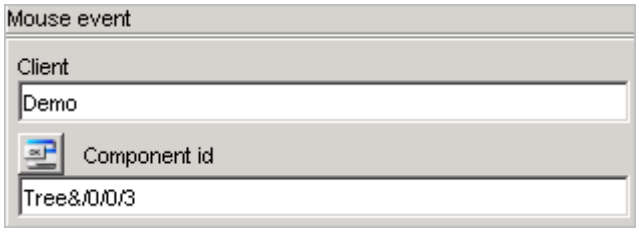

Figure 9.16 - Référence Numérique Direct à un Nœud JTree

CHAPITRE 10 - AVANT DE DEMARER VOTRE PROPRE APPLICATION

Depuis la version 1.07.0 de *qftestJUI* un nouveau mécanisme pour se connecter au SUT est fourni. Ce mécanisme utilise J**DK accessibility**, une interface Java officielle pour les outils d'accessibilité et de test. En plus d'un moyen facile pour configurer le démarrage du SUT, ce nouveau mécanisme permet également à *qftestJUI* de supporter le test de n'importe quelle application Java/Swing (sans aucune modification de son script de démarrage) et le test d'applets dans un navigateur. Dans ce didacticiel nous avons seulement décrit l'"ancien" mécanisme de démarrage du SUT. Ce chapitre vous éclaire sur la façon d'utiliser le nouveau mécanisme de connexion, qui est préférable.

Comme pré-condition de ce nouveau mécanisme de connexion vous devez préparer votre JDK, pour que *qftestJUI* utilise cette interface d'accessibilité. Nous l'appelons Instrumentation of the JDK. *qftestJUI* offre l'assistance nécessaire avec le menu **Extras -> Manage JDK instrumentation...** Il ouvre un dialogue qui vous permet de gérer l'instrumentation et la désinstrumentation du JDK installé sur votre ordinateur. Vous pouvez demander à *qftestJUI* de chercher tous les JDK installés démarrant de façon récursive depuis un emplacement donné. La (des)instrumentation du JDK fournie par la recherche peut être faite par un simple clic bouton, à condition que vous ayez les droits sur les fichiers système pour le faire. Finalement la liste des JDK résolus peut être sauvegardée pour de futurs changements de l'instrumentation. C'est tout pour la première étape et maintenant nous pouvons regarder comment démarrer le SUT.

Démarrez le SUT est très simple maintenant. La façon la plus simple est d'utiliser un nœud **Execute shell command**. Mais pour des raisons de cohérence et de compatibilité antérieure **Start SUT client** est le nœud préférable, ou **Start Java SUT client** si vous avez besoin d'une classe Java ou d'une archive Jar pour le lancer. Une description détaillée des différentes possibilités pour démarrer votre SUT est disponible dans le Manuel Utilisateur. C'est tout ce que vous avez besoin de savoir, donc il est maintenant tant que vous essayez par vous-même.

Si vous êtes intéressé(e) par les détails techniques de l'instrumentation du JDK et par ce qui se passe derrière, merci de bien vouloir vous référer à la ***Référence Technique du Manuel***.

LECTURE AVANCEE

Pour les utilisateurs avancés nous avons préparé une suite de test de démonstration dans **qftestJUI-1.08.4/doc/tutorial/demo_script.qft**. Elle contient un nombre d'exemples sur l'utilisation du script Jython et elle est utilisable à souhait. Des détails sur l'utilisation du langage de script Jython, incluant un didacticiel, peuvent être trouvés sur www.jython.org.

Pour finir nous aimerions insister sur l'importance et la valeur du Manuel Utilisateur de *qftestJUI*, qui contient des explications détaillées sur ce sujet avancé et également des exemples utiles.