Inleiding tot het schrijven van plug-ins voor RKWard

Thomas Friedrichsmeier
Meik Michalke
Vertaler/Nalezer: Freek de Kruijf
Vertaler/Nalezer: Jaap Woldringh

Inleiding tot het schrijven van plug-ins voor RKWard

Inhoudsopgave

1 Inleiding **9**

2 Vooraf: Wat zijn plugins in RKWard? Hoe werken zij? **10**

3 Menu-ingangen aanmaken **11**

3.1 Het regelen van de volgorde van de menu-ingangen 14

4 Het definiëren van de GUI (Grafische interface) **15**

4.1 Een dialoog definiëren 15

4.2 Het toevoegen van een interface voor de assistent (wizard) 18

4.3 Enige beschouwingen bij het ontwerpen van een GUI 19

4.3.1 <radio> vs. <checkbox> vs. <dropdown> 20

5 Genereren van R code volgens de instellingen in de GUI **21**

5.1 JavaScript gebruiken in plugins voor RKWard 21

5.1.1 preprocess() 21

5.1.2 calculate() 22

5.1.3 printout() 22

5.2 Conventies, policies, en achtergrond 23

5.2.1 De `local()` omgeving begrijpen 23

5.2.2 Programma-code formatteren 24

5.2.3 Omgaan met complexe opties 24

5.3 Tips en trucs 24

6 Een help-pagina schrijven: **26**

7 Logische interacties tussen GUI-elementen **29**

7.1 GUI-logica 29

7.2 GUI logica in scripts 31

8 Plugins inbedden in Plugins **32**
8.1 Voorbeelden van plugins in plugins . . . 32
8.2 Invoegen in een dialoog . . . 32
8.3 Het genereren van code tijdens het invoegen . . . 33
8.4 Inbedden in een assistent (wizard) . . . 33
8.5 Minder ingebedde inbedding: knop voor verdere opties . . . 33
8.6 Inbedden/definiëren van onvolledige plugins . . . 34

9 Werken met vele soortgelijke plugins **36**
9.1 Overzicht van de verschillende werkwijzen . . . 36
9.2 Met behulp van de JS include statement . . . 36
9.3 Includen van `.xml`-bestanden . . . 37
9.4 <snippets> gebruiken . . . 38
9.5 <include> en <snippets> vs. <embed> . . . 40

10 Te gebruiken concepten voor gespecialiseerde plugins **41**
10.1 Plugins die een plot maken . . . 41
10.1.1 Een plot tekenen in het uitvoervenster . . . 41
10.1.2 Toevoegen mogelijkheid voorbeeldweergave . . . 41
10.1.3 Generieke plotopties . . . 42
10.1.4 Een kaal voorbeeld . . . 43
10.2 Voorbeeldweergaven (previews) voor gegevens, uitvoer en andere resultaten . . . 44
10.2.1 Previews van (HTML)-uitvoer . . . 44
10.2.2 Voorbeeldweergaven (previews) van (geïmporteerde) gegevens . . . 45
10.2.3 Aangepaste voorbeeldweergaven (previews) . . . 46
10.3 Context-afhankelijke plugins . . . 46
10.3.1 Context voor X11-apparaat . . . 47
10.3.2 Context voor importeren van gegevens . . . 47
10.4 R bevragen naar informatie . . . 48
10.5 Naar het huidige object verwijzen . . . 49
10.6 Een (aantal) opties herhalen . . . 50
10.6.1 ˝Driven˝ optionsets . . . 51
10.6.2 Alternatieven: wanneer optionsets niet te gebruiken . . . 52

11 Omgaan met afhankelijkheden (dependencies) en compatibiliteitsproblemen. **54**
11.1 Versie-compatibiliteit van RKWard . . . 54
11.2 R versie-compatibiliteit . . . 55
11.3 Afhankelijkheden van R-pakketten . . . 56
11.4 Dependencies (afhankelijkheden) van andere RKWard `.pluginmaps` . . . 56
11.5 Een voorbeeld . . . 57

12 Plugin vertalingen **58**
12.1 Algemene beschouwingen . . . 58
12.2 i18n in RKWard's xml-bestanden . . . 58
12.3 i18n in RKWard's js-bestanden en secties . . . 59
12.3.1 i18n en aanhalingstekens . . . 60
12.3.2 i18n en backwards compatibility (terugwaartse compatibiliteit) . . . 61
12.4 Onderhoud van vertalingen . . . 61
12.5 Schrijven van vertalingen van plugins . . . 62

13 Auteur, licentie en versie-informatie **63**

14 Uw werk met anderen delen **65**
14.1 Externe plugins . . . 65
14.2 Waarom externe plugins? . . . 65
14.3 Structuur van een plugin-pakket . . . 66
14.3.1 Bestandshiërarchie . . . 66
14.3.1.1 Basis plugin-componenten . . . 67
14.3.1.2 Bijkomende informatie (optioneel) . . . 67
14.3.1.3 Geautomatiseerd testen van plugins (optioneel) . . . 68
14.4 Het compileren (build) van het plugin-pakket . . . 68

15 Plugin ontwikkelen met het rkwarddev pakket. **69**
15.1 Overzicht . . . 69
15.2 Praktijk-voorbeeld . . . 69
15.2.1 GUI beschrijving . . . 70
15.2.2 JavaScript code . . . 73
15.2.3 Plugin map . . . 74
15.2.4 Help-pagina . . . 74
15.2.5 De plugin-bestanden genereren . . . 75
15.2.6 Het volledige script . . . 75
15.3 Toevoegen van help-pagina's . . . 77
15.4 Plugins vertalen . . . 78

A Naslag **79**
A.1 Typen van Eigenschappen/mModifiers . . . 79
A.2 Elementen voor algemeen gebruik (general purpose) in elk XML-bestand (`.xml`, `.rkh`, `.pluginmap`) . . . 81
A.3 Te gebruiken elementen in de XML-beschrijving van de plugin . . . 81
A.3.1 Algemene elementen . . . 82
A.3.2 Interface-definities . . . 82
A.3.3 Elementen voor de indeling (layout) . . . 83

A.3.4 Actieve elementen 84
A.3.5 Logische sectie 91
A.4 Eigenschappen van plugin-elementen 94
A.5 Ingebedde plugins meegeleverd met de officiële uitgave van RKWard 99
A.6 Elementen voor gebruik in `.pluginmap`-bestanden 100
A.7 Te gebruiken elementen in .rkh (help) bestanden 104
A.8 Functies die in logische scripts voor de GUI kunnen worden gebruikt 106

B Problemen oplossen bij het ontwikkelen van een plugin **108**

C Licentie **109**

Lijst van tabellen

A.1 Standaard plugins die kunnen worden ingebed 100

Samenvatting

Dit is een gids voor het schrijven van plug-ins voor RKWard.

Hoofdstuk 1

Inleiding

> OPMERKING
> Documentation voor RKWard vanaf uitgave 0.6.4.

In dit document wordt beschreven hoe u plugins schrijft. Let er op dat op het moment van schrijven niet alle concepten als in steen zijn gebeiteld. U moet daarom dit document beschouwen als een introductie tot de huidige aanpak, en als basis voor discussie. Alle commentaar is welkom. Dit document is in de loop van de tijd wel wat groot geworden. Laat u dit niet afschrikken. We bevelen aan de vier basisstappen door te lezen (zoals hieronder beschreven), zodat u een idee krijgt hoe een en ander werkt. Hierna kunt u de inhoudstabel doorlezen om te zien welke gevorderde onderwerpen iets voor u zijn.

Voor vragen en commentaar, schrijf naar e-maillijst voor ontwikkeling van RKWard.

U hoeft dit niet te lezen om RKWard te kunnen gebruiken. Dit document gaat over het uitbreiden van RKWard. De doelgroep zijn geavanceerde gebruikers of mensen die willen helpen bij het verbeteren van RKWard.

Het schrijven van een plugin is in principe een proces in vier stappen:

- Een nieuwe Action plaatsen in de menu-hiërarchie
- Het uiterlijk en gedrag beschrijven van de GUI (grafische interface) van deplugin
- Het definiëren hoe R-code wordt gegenereerd, volgens de instellingen, die de gebruiker in de GUI maakt
- Het toevoegen van een help-pagina aan uw plugin

Deze worden elk op hun beurt besproken.

Enkele gevorderde onderwerpen kunnen in deze vier stappen worden gebruikt, maar worden in aparte hoofdstukken besproken, om het eenvoudig te houden:

- GUI-logica
- Plugins in plugins opnemen
- Bruikbare concepten voor het maken van vele series van soortgelijke plugins

Ook worden in deze hoofdstukken niet alle opties besproken, maar alleen de meest eenvoudige. Een volledige naslag van opties wordt apart ter beschikking gesteld.

Hoofdstuk 2

Vooraf: Wat zijn plugins in RKWard? Hoe werken zij?

Natuurlijk is de eerste vraag die u kunt hebben: Welke mogelijkheden van RKWard worden met behulp van plugins verwezenlijkt? Of: wat kun je met plugins doen?

Een antwoord is: Deselecteer alle `.pluginmap`-bestanden in **Instellingen** → **RKWard instellen** → **Plugins**, en ga na wat er dan ontbreekt. Een ietwat nuttiger antwoord is: de meeste beschikbare statistische functies worden gebruikt, worden via plugins gerealiseerd. Ook kunt u tamelijk flexibele GUI's aanmaken voor alle soorten van bewerkingen met behulp van plugins.

Het basisvoorbeeld van plugins in RKWard is die welke wij in dit document zullen doorwerken: een XML-bestand dat beschrijft hoe de GUI eruit ziet. Een JavaScript-bestand wordt daarnaast gebruikt voor het aanmaken van de R-code, volgens de instellingen van de GUI. Dit betekent dat de plugins niet werkelijk zelf de statistische berekeningen uitvoeren. In plaats daarvan genereren de plugins de R-syntaxis nodig voor het doen van deze berekeningen. De R-syntaxis wordt gestuurd naar de backend van R voor de uitwerking, en de uitvoer wordt typisch in het uitvoervenster getoond.

Lees in de volgende hoofdstukken hoe we dit gaan doen.

Hoofdstuk 3

Menu-ingangen aanmaken

Wanneer u een nieuwe plugin maakt, moet u dit aan RKWard vertellen. Dus is het eerste wat u moet doen, het schrijven van een `.pluginmap`-bestand (of het wijzigen van een bestaand bestand). Het formaat van een `.pluginmap`-bestand is XML. Ik loop met u door een voorbeeld (ook, natuurlijk, moet u er voor zorgen dat RKWard zo is ingesteld, dat het uw `.pluginmap` inleest -- **Instellingen** → **RKWard instellen** → **Plugins**):

> TIP
> Bekijk, na het lezen van dit hoofdstuk, ook het rkwarddev pakket. Hierin staan enkele R-functies waarmee u de meeste van de XML-tags kunt aanmaken in RKWard.

```
<!DOCTYPE rkpluginmap>
```

De doctype wordt niet werkelijk geïnterpreteerd, maar wel ingesteld op *´´rkpluginmap´´*.

```
<document base_prefix="" namespace="myplugins" id="mypluginmap">
```

Het *base_prefix*-attribuut kan worden gebruikt als al uw plugins in dezelfde directory zijn. U kunt dan die directory weglaten uit de hieronder genoemde bestandsnamen. U kunt die veilig *´´´´* laten blijven.

Zoals u hieronder zult zien, krijgen alle plugins een unieke naam, *id*. De *namespace* is een manier waaropIDs worden georganiseerd, zodat zij niet zo gauw per ongeluk dezelfde naam krijgen. In feite komt het erop neer dat de namespace (naamruimte) en daarna een'::' worden voorgevoegd aan alle namen die u opgeeft in deze `.pluginmap`. In het algemeen moet u, als u van plan bent uw plugins uit te geven in een R-pakket, de pakketnaam gebruiken als *namespace*-parameter. Plugins in het officiële pakket van de RKWard-distributie hebben *namespace=´´rkward´´*.

Het *id*-attribuut is optioneel, maar een id (een naam dus)voor uw `.pluginmap` opgeven maakt het anderen mogelijk hun `.pluginmap`s uw `.pluginmap`, automatisch te laten inlezen (zie de sectie over afhankelijkheden).

```
<components>
```

Componenten? We spreken toch over plugins? Ja, maar in de toekomst zijn plugins niets anders dan een speciale class van componenten. Wat we hier dus aan het doen zijn, is alle componenten/plugins te registreren in RKWard. Bekijken we een voorbeeld:

```
<component type="standard" id="t_test_two_vars" file="t_test_two_vars.xml"  ↩
    label="Two Variable t-Test" /> (Vert.: Door het hele document vertaal ik ↩
     dit niet, om de kans op verwarring, bij mij en de lezer, te  ↩
    minimaliseren. De vertaling hier zou zijn: <component type="standaard"  ↩
    id="t_test_twee_vars" file="t_test_twee_vars.xml" label="Twee variabelen ↩
     t-Test" />)
```

Eerst het *type*-attribuut: Laat dit voorlopig *´´standard´´* zijn. Andere typen zijn er nog niet. De *id* is al kort aan bod geweest. Elke component moet een (in zijn eigen namespace) unieke naam hebben. Kies er een die gemakkelijk herkenbaar is. Geen spaties, en geen speciale karakters. Mag wel, voorlopig, maar die kunnen een speciale betekenis hebben. Met het *file*-attribuut, geeft u de locatie op van de beschrijving van de actuele plugin zelf. Die is relatief ten opzichte van de directory waarin het `.pluginmap`-bestand is, en de *base_prefix* hierboven. Tenslotte, geef de component een naam. Deze naam wordt overal getoond waar de plugin in het menu wordt geplaatst (of in de toekomst ook elders).

Een `.pluginmap`-bestand bevat typisch een aantal componenten, dus hier zijn er nog een paar:

```
<component type="standard" id="unimplemented_test" file="means/ ↩
    unimplemented.xml" />
                <component type="standard" id="fictional_t_test" file=" ↩
                    means/ttests/fictional.xml" label="This is a fictional t ↩
                    -test" />
                <component type="standard" id="descriptive" file=" ↩
                    descriptive.xml" label="Descriptive Statistics" />
                <component type="standard" id="corr_matrix" file=" ↩
                    corr_matrix.xml" label="Correlation Matrix" />
                <component type="standard" id="simple_anova" file=" ↩
                    simple_anova.xml" label="Simple Anova" />
        </components>
```

OK, dit was het eerste (onvertaalde, tussen ´´ staan wat teksten) stapje. RKWard weet nu dat die plugins er zijn. Maar hoe die te gebruiken? Zij moeten in een menu-hiërarchie worden geplaatst:

```
<hierarchy>
                <menu id="analyse" label="Analysis">
```

Direct onder de **<hierarchy>** tag, begint u met de beschrijving van in welk **<menu>** uw plugins moeten komen. In de regel boven zegt u eigenlijk dat uw plugin moet komen in het menu **Analyse** (niet noodzakelijk direct daar, maar in een submenu). Het menu **Analyse** is standaard aanwezig in RKWard, dus hoeft het niet helemaal opnieuw te worden aangemaakt. Maar als het nog niet bestond, zou u met het *label*-attribuut zijn naam geven. Tenslotte zal met de *id* dit **<menu>** nog een naam krijgen. Dit is nodig, zodat diverse `.pluginmap`-bestanden hun plugins in dezelfde menus kunnen plaatsen. Dit gebeurt door te kijken naar een **<menu>** in de gegeven *id*. Indien de ID nog niet bestaat wordt een nieuw menu aangemaakt. En anders worden de ingangen toegevoegd aan het bestaande menu.

```
<menu id="means" label="Gemiddelden">
```

In principe hier hetzelfde: Nu definiëren we een submenu in het menu **Analyse**. Het moet de naam **Gemiddelden** krijgen.

```
<menu id="ttests" label="t-toetsen">
```

En een laatste level in de menu-hierarchie: een submenu van het submenu **Gemiddelden**.

```
<entry component="t_test_two_vars" />
```

Nu, eindelijk, is dit het menu waarin we de plugin willen plaatsen. De **<entry>** tag geeft aan dat dit het is,in plaats van weer een ander submenu. Het *component*-attribuut verwijst naar de *id* (naam) die u hierboven aan de component gaf.

```
<entry component="fictional_t_test" />
                                </menu>
                                <entry component="fictional_t_test" />
                        </menu>
                        <menu id="frequency" label="Frequency" index="2"/>
```

Voor als u het spoor bijster bent: Dit is nog een submenu van het menu **Analyse**. Zie het onderstaande schermbeeld. We slaan een en ander over dat niet zichtbaar is, gemarkeerd met [...].

```
[...]
                        </menu>
                        <entry component="corr_matrix"/>
                        <entry component="descriptive"/>
                        <entry component="simple_anova"/>
                </menu>
```

Dit zijn de uiteindelijke ingangen, zichtbaar in onderstaande schermbeelden.

```
<menu id="plots" label="Plots">
                        [...]
                </menu>
```

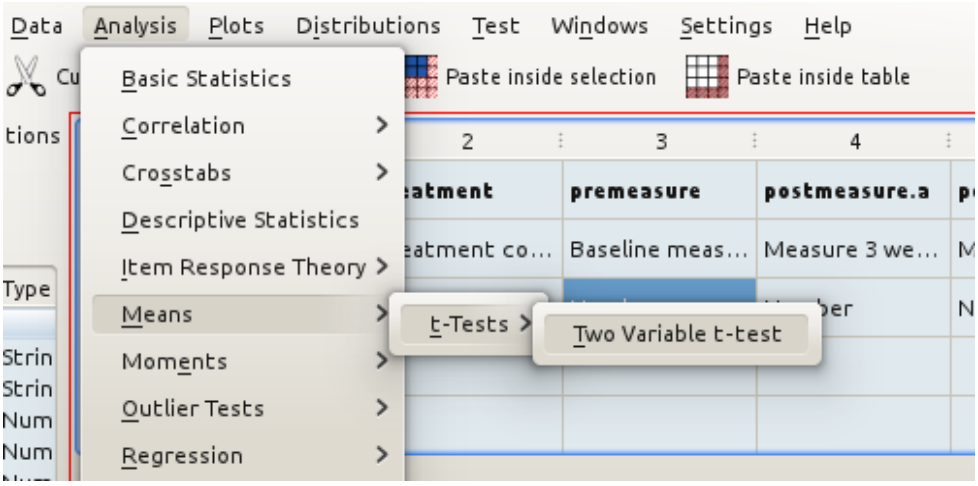
Natuurlijk kunt u uw plugins ook plaatsen in andere menu's dan het menu **Analyse**.

```
<menu id="file" label="Bestand">
                        [...]
                </menu>
```

Zelfs in standaard menu's zoals **Bestand**. Alles wat nodig is, is een correcte *id*.

```
</hierarchy>
</document>
```

Zo moet u het aanpakken. En dit schermbeeld toont het resultaat:



Duizelt het u? De beste manier om te beginnen, is mogelijk om uit te gaan van de bestaande `.pluginmap`-bestanden die met de distributie worden geleverd, en die te wijzigen daar waar u dit nodig vindt. En ook: indien u hulp wenst, aarzel dan niet te schrijven naar de development mailing list (Engels).

3.1 Het regelen van de volgorde van de menu-ingangen

Standaard wordt alles (ingangen / submenu's) in een menu automatisch alfabetisch gesorteerd. In *sommige* gevallen wilt u meer controle hierop hebben. In dat geval kunt u als volgt de elementen in groepen indelen:

- U kunt op deze manier groepen binnen een menu definiëren. Alle elementen binnen dezelfde groep worden bij elkaar gegroepeerd:

```
<group id="eengroep"/>
```

- Indien u de groep visueel wilt scheiden van andere ingangen, doet u dit:

```
<group id="eengroep" separated="true"/> (true betekent waar)
```

- Ingangen, menu's en groepen kunnen aan een groep worden toegevoegd met:

```
<entry component="..." group="eengroep"/>
```

- In feite is het ook mogelijk groepen (zonder scheidingslijnen) impliciet te definiëren:

```
<entry component="eerste" group="a"/>
                <entry component="derde"/>
                <entry component="tweede" group="a"/>
```

- Groepsnamen behoren bij hun eigen menu. Groep ˝a˝ in menu ˝Gegevens˝ botst niet met groep ˝a˝ in het menu ˝Analyse˝, bijvoorbeeld.
- Het meest komt voor dat groepen helemaal bovenin het menu worden gedefinieerd, of juist helemaal onderin. Hiervoor zijn reeds de groepen ˝top˝ (bovenin) en ˝bottom˝ (onderin) aangemaakt in elk menu.
- Ingangen binnen elke groep zijn alfabetisch gesorteerd. Groepen verschijnen in de volgorde van hun declaraties (behalve indien toegevoegd aan een andere groep, natuurlijk).
- Menu's en ingangen zonder groepspecificatie vormen logisch ook een groep (˝˝).

Hoofdstuk 4

Het definiëren van de GUI (Grafische interface)

4.1 Een dialoog definiëren

In het vorige hoofdstuk zag u hoe u met RKWard een plugin kunt registreren. Het belangrijkste was het opgeven van het pad naar een XML-bestand met een beschrijving van de plugin. In dit hoofdstuk leert u hoe u dit XML-bestand aanmaakt.

> TIP
> Bekijk, na het lezen van dit hoofdstuk, ook het rkwarddev pakket. Hierin staan enkele R-functies waarmee u de meeste van de XML-tags kunt aanmaken in RKWard.

We gaan weer met u door een voorbeeld wandelen. Dit voorbeeld is een (beetje vereenvoudigde) versie van de t-Test met twee variabelen.

```
<!DOCTYPE rkplugin>
```

De doctype wordt nog niet werkelijk geïnterpreteerd. Maar stel het toch maar in op *rkplugin*.

```
<document>
        <code file="t_test_two_vars.js"/>
```

Alle plugins genereren wat code. Op dit moment kan dit alleen met JS (JavaScript), zoals uitvoerig beschreven in het volgende hoofdstuk. Daarin staat waar u de JS-code kunt vinden. De bestandsnaam is relatief ten opzichte van de locatie van de plugin XML.

```
<help file="t_test_two_vars.rkh"/>
```

Het is meestal een goed idee om een help-pagina te maken voor uw plugin. De bestandsnaam van die help-pagina wordt hier gegeven, relatief ten opzichte van de locatie van de plugin XML. Het schrijven van help-pagina's wordt hier beschreven. Laat deze regel weg, als u geen help-pagina maakt.

```
<dialog label="Two Variable t-Test">
```

Zoals u weet kunnen plugins een dialoog bevatten, of een assistent, of beide. Hier beginnen we met het definiëren van een dialoogvenster. Het attribuut *label* definieert de koptekst van de dialoog.

```
<tabbook>
                        <tab label="Basisinstellingen">
```

GUI-elementen kunnen worden georganiseerd in een tabbook. Hier definiëren wij een tabbook als het eerste element in de dialoog. Definieer met **<tabbook>**[...]**</tabbook>** de tabbook en daarna elke pagina in de tabbook met **<tab>**[...]**</tab>**. Met het attribuut *label* in het **<tab>**-element kunt u een koptekst opgeven voor die pagina in de tabbook.

```
<row id="main_settings_row">
```

De tags **<row>** en **<column>** bepalen hoe de GUI-elementen worden geplaatst. Hierin bepaalt u dat u enkele elementen naast elkaar wilt plaatsen (links naar rechts). Het *id*-attribuut is niet direct nodig, maar we gaan het later gebruiken, wanneer we aan de plugin een assistent toevoegen. Het eerste in een rij te plaatsen element is:

```
<varselector id="vars"/>
```

Met deze eenvoudige tag maakt u een lijst waarin de gebruiker variabelen kan selecteren. U moet voor dit element een *id* (naam) opgeven, zodat RKWard het kan vinden.

> WAARSCHUWING
> Er mag GEEN punt (.) in de *id*-string voorkomen.

```
<column>
```

Vervolgens plaatsen we een **<column>** in de rij. Dit betekent dat we de volgende elementen boven elkaar plaatsen (van boven naar beneden), en alles komt dan rechts van de **<varselector>**.

```
<varslot types="getal" id="x" source="vars" required="true" label="compare ↩
    "/>
                                                <varslot types="getal" id=" ↩
                                                    y" source="vars"  ↩
                                                    required="true" label=" ↩
                                                    met" i18n_context=" ↩
                                                    vergelijken met"/>
```

Deze elementen zijn de tegenpartij van de **<varselector>**. Zij zijn 'plaatshouders' waarin de gebruiker variabelen kan plaatsen. U merkt dat de *source* wordt ingesteld op de zelfde waarde als de *id* van de **<varselector>**. Dit betekent dat de **<varslot>**s hun variabelen verkrijgen uit de varselector. De **<varslot>**s moeten ook een *id* (naam) krijgen. Mogelijk hebben zij een *label* (naam), en zijn die ingesteld op *required* (vereist). Dit betekent dat de knop **Indienen** niet werkt totdat de **<varslot>** een geldige waarde heeft. Tenslotte wordt het *type*-attribuut nog niet geïnterpreteerd, maar zorgt dat ervoor dat alleen geldige variabelen-typen worden toegestaan in de **<varslot>**.

Voor het geval dat u zich afvraagt wat het attribuut *i18n_context* betekent: Hiermee wordt de context geleverd die helpt bij de correcte vertaling van het woord ˝against˝, dat gebruikt wordt in de label van de **<varslot>**, maar het beïnvloedt niet direct de functionaliteit van de plugin. Hierover meer in een afzonderlijk hoofdstuk. (Vert.: ik heb ˝against˝ dat er eerst stond, vertaald met ˝met˝: vergelijken met)

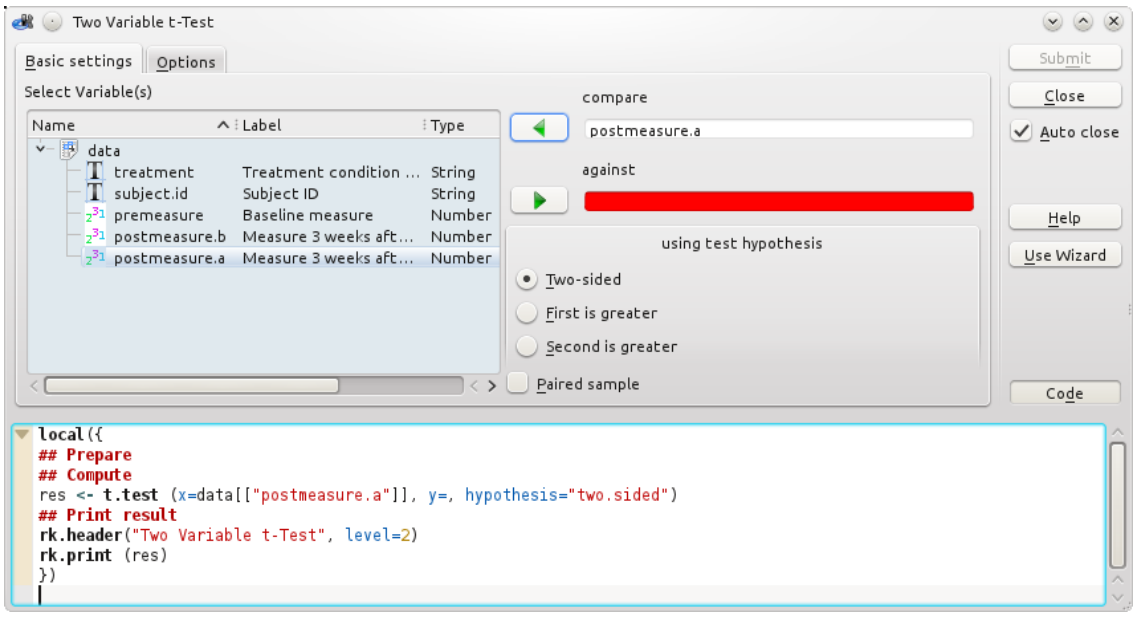
```
<radio id="hypothese" label="met test-hypothese">
                                                          <option value="two. ↩
                                                              sided" label=" ↩
                                                              Tweezijdig"/>
```

```
                                                    <option value=" ↩
                                                        groter" label=" ↩
                                                        Eerste is groter ↩
                                                        "/>
                                                    <option value=" ↩
                                                        kleiner" label=" ↩
                                                        Tweede is groter ↩
                                                        "/>
                                                </radio>
```

Hier definieert u een groep van **<radio>**-knoppen. De groep heeft een *label* en een *id*. Elke **<option>** (knop) heeft een *label* en is toegewezen aan een *waarde*. Deze waarde wordt door het **<radio>**-element teruggegeven wanneer de optie wordt geselecteerd.

```
</column>
                                        </row>
                                </tab>
```

Elke tag moet worden afgesloten. We hebben alle gewenste elementen geplaatst (de twee **<varslots>** en de **<radio>**) in de **<column>**. We plaatsten alle gewenste elementen (de **<varselector>** en de **<column>** waarin die elementen) in de **<row>**. En we plaatsten alle gewenste elementen in de eerste pagina in de **<tabbook>**. We zijn nog niet klaar met het definiëren van de **<tabbook>** (er komen nog wat meer pagina's), en natuurlijk moet er ook nog meer komen in de **<dialog>**. Maar in dit schermbeeldzien we in principe wat we al hebben gedaan:



Merk op dat we knoppen **Indienen**, **Afsluiten**, etc. of de programmacode hebben gespecificeerd. Die elementen worden automatisch aangemaakt. Maar natuurlijk moeten we nog de tweede pagina van de **<tabbook>** definiëren:

```
<tab label="Opties">
                                    <checkbox id="varequal" label="Aanname: ↩
                                        gelijke varianties" value=", var.equal= ↩
                                        TRUE"/>
```

Standaard worden elementen van boven naar beneden geplaatst, als in een **<column>**. Omdat dit precies is wat we hier willen, hoeven we niet expliciet een **<row>** of **<column>**-indeling op te geven. Het eerste element dat we definiëren is een keuzevakje. Net als de **<radio><optie>**s, heeft

het keuzevakje een *label* (naam) en een *value* (waarde). De *value* is wat wordt teruggegeven, wanneer het keuzevakje wordt geselecteerd. Natuurlijk heeft het keuzevakje ook een *id* (naam).

```
<frame label="Confidence Interval" id="frame_conf_int">
```

Hier is nog een ander indelingselement: om aan te geven dat de de twee elementen hieronder bij elkaar horen, tekenen we een **<frame>** (vak). Die frame kan een *label* (kopnaam) hebben. Omdat de frame slechts een passief indelingselement is, heeft die geen *id* nodig. Toch definiëren we er hier een, omdat we er later naar zullen verwijzen, wanneer we een extra interface maken voor de assistent.

```
<checkbox id="confint" label="druk vertrouwensinterval af" value="1"  ↩
    checked="true"/>
                                          <spinbox type="real" id="conflevel" ↩
                                               label="vertrouwensniveau" min ↩
                                              ="0" max="1" initial="0.95"/>
                                  </frame>
```

In de **<frame>** plaatsen we nog een **<keuzevakje>** (met *checked=´´true´´*, geven wij aan dat het keuzevakje standaard gekozen is), en een **<spinveld>**. In het spinveld kan de gebruiker een waarde kiezen tussen *´´min´´* en *´´max´´* met de standaard/begin-waarde *´´0.95´´*. Het *type* op *´´real´´* instellen betekent dat reële getallen (kommagetallen) worden geaccepteerd, in tegenstelling tot *type=´´integer´´* wat betekent dat alleen gehele getallen worden geaccepteerd..

> OPMERKING
> Het is ook mogelijk, en vaak zelfs beter, de **<frame>** zelf selecteerbaar te maken, in plaats van het erin toevoegen van een **<keuzevakje>**. Zie de naslag voor de details. Hier hebben we dit niet gedaan, voor illustratieve doeleinden.

```
</tab>
                </tabbook>
        </dialog>
```

Dat is alles voor de tweede pagina van de **<tabbook>**, alle pagina's in de **<tabbook>** en alle elementen in de **<dialog>**. We zijn klaar met het definiëren hoe de dialoog eruit zal zien.

```
</document>
```

Tenslotte sluiten we de **<document>**-tag af, en dat is het dan. De GUI is gedefinieerd. U kunt het bestand nu opslaan. Maar hoe wordt de syntaxis van R uit de instellingen van de GUI gegenereerd? Dit bespreken we in het volgende hoofdstuk. Maar eerst bekijken we het toevoegen van de interface voor een assistent (wizard), en geven we een aantal algemene beschouwingen.

4.2 Het toevoegen van een interface voor de assistent (wizard)

Eigenlijk hoeven we geen extra **<assistent>**-interface toe te voegen, maar we laten hier zien hoe het moet. Een interface toevoegen voor een assistent, doet u door een **<wizard>**-tag toe te voegen op hetzelfde menu-niveau als de **<dialog>**-tag:

```
<wizard label="Two Variable t-Test">
                        <page id="firstpage">
                                <text>Eerst kiezen we de twee met elkaar te ↩
                                     vergelijken variabelen.
```

```
                                    En geven aan welke volgens u de  ↩
                                        grootste is. Selecteer  ↩
                                        Tweezijdig,
                                    indien uw theorie niet vertelt  ↩
                                       welkevariabele groter is.</text>
                            <copy id="main_settings_row"/>
                    </page>
```

Hiervan is een en ander vanzelfsprekend: We voegen een**<wizard>**-tag toe met een *label* (nnam) voor de assistent. Omdat een assistent meerdere pagina's kan hebben die na elkaar worden getoond, gaan we eerst een **<page>** definiëren, waarin we een **<text>** plaatsen met uitleg. Daarna gebruiken we een **<copy>**-tag. Wat dit doet bespaart ons de noodzaak opnieuw te definiëren wat we al schreven voor de **<dialoog>**: de copy-tag zoekt naar een andere tag met dezelfde *id* eerder in de XML. Toevallig is die gedefinieerd in de **<dialoog>**-sectie, en is er een **<row>** waarin de **<varselector>**, **<varslots>** en de 'hypothese' **<radio>**-besturing. Dit alles wordt 1:1 gekopieerd en ingevoegd in het **<copy>**-element.

Nu de tweede pagina:

```
<page id="secondpage">
                            <text>Onder zijn wat gevorderde opties. Het ↩
                                 is meestal niet veilig aan te nemen dat
                                    variabelen dezelfde varianties  ↩
                                        hebben. Er wordt dan een  ↩
                                        geschikte correctie toegepast.
                                    Maar kiezen van "gelijke varianties ↩
                                         aannemen" kan de teststerkte  ↩
                                        verbeteren.</text>
                            <copy id="varequal"/>
                            <text>Soms helpt het hebben van een  ↩
                                schatting van het vertrouwensinterval  ↩
                                van
                                    het verschil van de gemiddelden.  ↩
                                        Onder kunt u opgeven of er een  ↩
                                        moet worden getoond, en
                                    welk vertouwensniveau moet worden  ↩
                                        toegepast (95% komt overeen met  ↩
                                        een 5%
                                    significantie-niveau.).</text>
                            <copy id="frame_conf_int"/>
                    </page>
            </wizard>
```

Veel van hetzelfde hier. we voegen wat teksten toe, en **<copy>** daarin wat secties van de dialoog-interface.

U kunt natuurlijk de assistent-interface heel anders ontwerpen dan die van de eenvoudige dialoog, en de **<copy>**-tag helemaal niet gebruiken. Maar zorg er dan wel voor dat overeenkomstige elementen in beide interfaces dezelfde *id* krijgen. Niet alleen voor het overbrengen van instellingen van de dialoog-interface naar die van de assistent, en terug, wanneer de gebruiker de andere interface wil gebruiken, (wat in de huidige versie van RKWard nog niet kan), maar het maakt het schrijven van de template (sjabloon) voor uw code eenvoudiger (zie hier onder).

4.3 Enige beschouwingen bij het ontwerpen van een GUI

In deze sectie zijn enkele algemene beschouwingen over welke GUI-elementen waar moeten worden gebruikt. Indien dit de eerste keer is dat u een plugin probeert te maken, kunt u deze sectie

overslaan, omdat het niet gaat over de basis van het maken van een GUI. Kom later terug, om te zien of u de GUI's van uw plugins misschien kunt verfijnen.

4.3.1 <radio> vs. <checkbox> vs. <dropdown>

De drie elementen **<radio>**, **<checkbox>** (keuzevakje), **<dropdown>** (neerklapmenu), hebben allemaal een zelfde soort functie: het laten kiezen uit een aantal opties. Natuurlijk kunt u met een keuzevakje alleen maar uit twee opties kiezen: gekozen, of niet gekozen, u kunt die dus niet gebruiken als er uit meer dan twee opties moet worden gekozen. Maar wanneer, welk van de elementen te gebruiken? Enkele vuistregels zijn:

Als u merkt dat u een **<radio>** of **<dropdown>** aanmaakt met slechts twee opties, vraag u dan af of de vraag eigenlijk een ja / nee type vraag is. Bv. een keuze tussen 'resultaten aanpassen' en 'resultaten niet aanpassen', of tussen 'ontbrekende waarden verwijderen' en 'ontbrekende waarden behouden'. In zo'n geval is een**<checkbox>** een betere keus: het gebruikt minder ruimte, er zijn minder woorden of labels (namen) nodig, en is gemakkelijker te begrijpen voor de gebruiker. Er zijn maar weinig situaties waarin u beter een **<radio>** kunt kiezen in plaats van een **<checkbox>**, als er slechts twee opties zijn. Een voorbeeld daarvan zou kunnen zijn: 'Berekeningsmethode: 'Pearson'/'Spearman''. Hier zijn meer methodes denkbaar, en zij zijn niet echt een stel tegengestelde methodes.

Kiezen tussen een **<radio>** en een **<dropdown>** hangt meestal af van de beschikbare ruimte. De **<dropdown>** heeft het voordeel minder ruimte nodig te hebben, zelfs als er veel opties zijn waaruit moet worden gekozen. Aan de andere kant heeft **<radio>** het voordeel dat de gebruiker alle mogelijke keuzes in een oogopslag kan zien, zonder te moeten klikken op een pijltje in een keuzelijst. Algemeen gesproken, als er zes of meer opties zijn waaruit moet worden gekozen, heeft **<dropdown>** de voorkeur. Met vijf of minder opties kunt u beter **<radio>** kiezen.

Hoofdstuk 5

Genereren van R code volgens de instellingen in de GUI

5.1 JavaScript gebruiken in plugins voor RKWard

We hebben nu een GUI gemaakt, maar moeten nu nog wat R-code hiervoor maken. Hiervoor hebben we nog een tekstbestand nodig, `code.js`, in dezelfde directory als de `description.xml`. Misschien kent u JavaScript al (of, technisch beter gezegd: ECMA-script). Documentatie voor JS is er in overvloed, zowel gedrukt, als op het Internet (bijv.: https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide). Maar in de meeste gevallen hoeft u niet echt veel kennis te hebben van JS, omdat we alleen een aantal erg eenvoudige eigenschappen zullen gebruiken.

> TIP
> Na het lezen van dit hoofdstuk, kunt u ook kijken op rkwarddev pakket. Hierin staan enkele R-functies voor het maken van JavaScript-code die veel in RKWard wordt gebruikt. Het kan ook ˝autodetect˝ (automatisch detecteren) variabelen die in een plugin XML-bestand worden gebruikt en heel eenvoudige JavaScript-code aanmaken, voor u om me te beginnen.

> OPMERKING
> Aangenomen wordt dat de .js-bestanden van plugins UTF-8 (een karakter-verzameling) gebruiken. Ga na of uw bewerkingsprogramma die ook gebruikt, indien u non-ascii karakters gebruikt.

Voor de t-test met twee variabelen, is het bestand `code.js` als volgt (met hierin commentaar):

5.1.1 preprocess()

```
function preprocess () {
}
```

Het JS-bestand is ingedeeld in drie afzonderlijke functies: `preprocess()`, `calculate()` (berekenen), en `printout()` (afdrukken). Dit omdat niet alle code overal nodig is. Op dit moment wordt de preprocess-functie op veel plaatsen niet echt gebruikt (eigenlijk kan u die geheel weggelaten).

5.1.2 calculate()

```
function calculate () {
        echo ('res <- t.test (x=' + getString ("x") + ', y=' + getString (" ↩
            y") + ', hypothesis="' + getString ("hypothese") + '"' +  ↩
            getString ("varequal"));
        var conflevel = getString ("conflevel");
        if (conflevel != "0.95") echo (', conf.level=' + conflevel);
        echo (')\n');
}
```

Deze functie genereert de actuele syntaxis in R die vanuit de GUI moet worden gestart. Bekijken we die eens nauwkeurig: De te gebruiken code wordt gegenereerd met de statement `echo()`. Als we de statement `echo()` stap voor stap bekijken, is het eerste deel:

```
res <- t.test (
```

als leesbare tekst. Vervolgens moeten we de waarde invullen, van de variabele die de gebruiker als eerste selecteerde. We halen die op met `getString (´´x´´)`, en plakken die aan de string die moet worden 'ge-echo'd'. Dit geeft de waarde van het GUI-element met de *`id=´´x´´`*: onze eerste **<checkbox>**. Vervolgens voegen we een', ' toe, en halen op dezelfde manier de waarde op van het element *`´´y´´`* - de tweede **<checkbox>**. Voor de hypothese (de **<radio>** groep), en de gelijke varianties **<checkbox>**, is de werkwijze in grote lijnen dezelfde.

Merk op dat u, in plaats van de uitvoer-stukjes met '+' aan elkaar plakt, u ook een aantal aparte `echo()` statements kunt gebruiken. Alles wordt op een enkele regel weergegeven. U kunt meerdere regels gebruiken door in de ge-echode string een ˝\n˝ (nieuwe regel) op te nemen. In theorie kunt met een echo-statement vele regels genereren, maar houdt u het alstublieft eenvoudig en beperk u tot slechts een (of minder) regels per `echo()`.

> OPMERKING
> Behalve `getString()`, zijn er functies `getBoolean()`, die een logische waarde probeert terug te geven (bruikbaar in een `if()`-statement), en `getList()`, die gegevens probeert terug te geven in een soort listin een JS-`Array()`. Later zien we hier wat voorbeelden van.
> Bij het beschouwen van bestaande plugins, ziet u dat in veel plugins`getValue()` wordt gebruikt, in plaats van `getString()`, en in feite zijn zij *bijna* identiek. Maar `getString()`, `getBoolean()` en `getList()` gebruiken wordt aanbevolen vanaf versie 0.6.1.

Voor de confidence level (de betrouwbaarheid) is het net wat lastiger. Vanwege de esthetica, willen we niet expliciet opgeven welk niveau we willen, als het met de standaardwaarde overeenkomt. Daarom, in plaats van de waarde zonder meer weer te geven, halen we die eerst op in een variabele. Daarna controleren we of die verschilt van *`´´0.95´´`*, en als dat zo is geven we een extra argument weer. Tenslotte, echo-en we een a afsluithaakje en een einde regel: *`´´)\n´´`*. Dat is alles wat de calculate (bereken) functie betref.

5.1.3 printout()

```
function printout () {
        echo ('rk.header (' + i18n ("Two Variable t-Test") + ')\n');
        echo ('rk.print (res)\n');
}
```

En dit is alles voor de printout (weergave) functie in de meeste gevallen. `rk.header()` drukt een standaard kopregel af voor de resultaten. Merk op dat in de .js-bestanden, u alle vertaalbare

strings met de handmoet markeren, met de opdracht **i18n()**, of andere. Hierover meer in het hoofdstuk over internationalisatie. U kunt, als u dit wilt, hieraan ook wat informatie toevoegen, bijv.:

```
function printout () {
        new Header (i18n ("Twee variabelen t-Test"))
                  .addFromUI ("varequal")
                  .add (i18n ("Vertrouwensniveau"), getString ("conflevel") ↩
                      )  // Merk op: zo geschreven als illustratie. Meer  ↩
                      automatisch:
        //        .addFromUI ("conflevel")
                  .print ();
echo ('rk.print (res)\n');
}
```

`rk.print()` gebruikt het R2HTML-pakket voor HTML- geformatteerde uitvoer. Nog een nuttige functie is `rk.results()`, die ook verschillende soorten tabellen kan uitvoeren. Maar bij twijfel gebruikt u gewoon `rk.print()`, en klaar is Kees. De JS class `Header` is een hulpfunctie in JS die `rk.header()` gebruikt (kijk maar naar de gegenereerdeR-code). Soms kunt u de functie `echo ('rk.header (...)')` direct aanroepen om een koptekst bij uw uitvoer af te drukken.

Merk op dat, tot dusver, intern de uitvoer gewoon een kaal HTML-document is. U kunt daarom in de verleiding komen om eigen HTML toe te voegen met `rk.cat.output()`. Dit werkt, maar doe dit liever niet. Het uitvoerformaat kan in de toekomst veranderen (bijv. naar ODF),dus is het beter geen HTML-specifieke code toe te voegen. Beter kunt u een en ander eenvoudig houden, met `rk.header()`, `rk.print()`, `rk.results()`, en -- wanneer nodig -- `rk.print.literal()`. Als dit niet genoeg is, en u meer nodig heeft, neem dan contact met ons op op de mailing list, voor hulp.

Gefeliciteerd! U heeft uw eerste plugin gemaakt. Lees verder in de volgende hoofdstukken voor meer gevorderde onderwerpen.

5.2 Conventies, policies, en achtergrond

Er zijn meerdere manieren waarop R-code kan worden geschreven voor een bepaalde taak, en er zijn zelfs nog meer manieren waarop deze R-code met JS kan worden gegenereerd. Hoe u dit precies aanpakt, is helemaal aan u. Maar er zijn een aantal overwegingen die u moet volgen, en achtergrondinformatie die u moet begrijpen.

5.2.1 De `local()` omgeving begrijpen

Vaker wel dan niet moet u een of meer tijdelijke R-objecten aanmaken in de code die door uw plugin wordt gegenereerd. Normaal gesproken wilt u die niet plaatsen in de werkruimte (workspace) van de gebruiker, waardoor zelfs misschien variabelen van de gebruiker kunnen worden overschreven. Daarom moet alle door de plugin gegenereerde code werken in een `local()` omgeving (zie R help-pagina voor functie `local()`). Dit betekent dat alle variabelen die u aanmaakt tijdelijk zijn, en niet permanent worden opgeslagen.

Als de gebruiker expliciet vraagt dat een variabele wordt opgeslagen, moet u dat object (een waarde) toekennen met `.GlobalEnv$objectname <- value`. Gebruik, in het algemeen gesproken, niet de operator `<<-` operator. Die kent in .GlobalEnv niet noodzakelijk (een waarde) toe.

Een grote valkuil is het gebruik van `eval()`. U moet hier beseffen dat eval standaard bij de evaluatie (waardebepaling) de huidige omgeving gebruikt, bijv. de lokale. Meestal werkt dat goed, maar niet altijd. Dus, als u `eval()` nodig heeft, moet u waarschijnlijk de *`envir`*-parameter opgeven: `eval(..., envir=globalenv())`.

5.2.2 Programma-code formatteren

Het belangrijkste is dat uw gegenereerde R-code werkt, maar die moet ook gemakkelijk te lezen zijn. Let er daarom op dat u dit ook goed formatteert. Enkele beschouwingen:

Normale top_niveau R-statements moeten links uitgelijnd zijn.

Statements in een lager blok moeten een tab inspringen (zie onderstaand voorbeeld)

Indien u erg complexe berekeningen doet, voeg dan hier en daar wat commentaar toe, vooral om logische secties te markeren. Merk op dat er een speciale functie **comment()** is, voor het toevoegen van vertaalbaar commentaar in de gegenereerde code.

Bij voorbeeld, kan de gegenereerde code er aldus uitzien. Dezelfde code zonder inspringen en commentaar zou behoorlijk lastig te lezen zijn, hoewel die maar tamelijk complex is:

```
# Bepaal eerst de wobble en de rotatie
my.wobble <- wobble (x, y)
my.rotation <- wobble.rotation (my.wobble, z)

# boggling-methode hangt af van de rotatie
if (my.rotation > wobble.rotation.limit (x)) {
        method <- "foo"
        result <- boggle.foo (my.wobble, my.rotation)
} else {
        method <- "bar"
        result <- boggle.bar (my.wobble, my.rotation)
}
```

5.2.3 Omgaan met complexe opties

Vele plugins doen meer dan een ding. Bij voorbeeld, de plugin 'Descriptive Statistics' (beschrijvende statistiek) kan gemiddelde, bereik, som, product, mediaan, lengte, etc. berekenen. Maar gewoonlijk zal de gebruiker alleen maar enkele hiervan berekenen. In zo'n geval, moet u de gegenereerde code zo eenvoudig mogelijk proberen te houden. Die moet alleen die gedeelten bevatten die overeenkomen met de werkelijk geselecteerde opties. U kunt dit bereiken, met een voorbeeld van een algemeen design pattern (ontwerppatroon) zoals u die zou gebruiken (hier, in JS, zouden ˝domean˝, ˝domedian˝, en ˝dosd˝ <checkbox> elementen zijn):

```
function calculate () {
        echo ('x <- <' + getString ("x") + ')\n');
        echo ('results <- list ()\n');

        if (getBoolean ("domean.state")) echo ("results$" + i18n (" ↩
            Gemiddelde waarde") + " <- mean (x)\n");
        if (getBoolean ("domedian.state")) echo ("results$" + i18n (" ↩
            Mediaan") + " <- median (x)\n");
        if (getBoolean ("dosd.state")) echo ("results$" + i18n ("Standaard ↩
            deviatie") + " <- sd (x)\n");
        //...
}
```

5.3 Tips en trucs

U ziet hier een aantal trucs die het schrijven van plugins kunnen vereenvoudigen:

Als u de waarde in een GUI-instelling op meerdere plaatsen in uw plugin nodig heeft, kunt u overwegen er in JS een waarde aan te geven, en die te gebruiken in plaats van die keer op keer op te halen met `getString()/getBoolean()/getList()`. Dit is sneller, leesbaarder, en u hoeft minder te typen:

```
function calculate () {
        var narm = "";        // na.rm=FALSE is standaard in alle functies ↩
            hieronder
        if (getBoolean ("remove_nas")) {
                $narm = ", na.rm=TRUE";
        }
        // ...
        echo ("results$foo <- foo (x" + narm + ")\n");
        echo ("results$bar <- bar (x" + narm + ")\n");
        echo ("results$foobar <- foobar (x" + narm "\n");
        // ...
}
```

De eenvoudige hulp-functie `makeOption()` kan het vaak eenvoudiger maken parameters weg te laten die een standaard waarde waarde hebben:

```
function calculate () {
        var options
        //...
        // Dit doet niets, als VALUE is 0.95 (standaard). Anders voegt het ↩
            ', conf.int=VALUE' toe aan options.
        options += makeOption ("conf.int", getString ("confint"), "0.95");
        //...
}
```

Hoofdstuk 6

Een help-pagina schrijven:

Als uw plugin in principe werkt, komt het moment dat u een help-pagina gaat maken. U wilt wellicht niet alle onderliggende concepten uitputtend uit de doeken doen, maar wel enige uitleg geven over sommige opties, en koppelen aan gerelateerde plugins en functies in R.

> TIP
> Na het lezen van dit hoofdstuk kunt u ook kijken op rkwarddev pakket. Hierin zijn enkele R-functies die de meeste XML-tags voor RKWard voor u kunnen aanmaken. Het kan, om te beginnen, ook eenvoudige kale help-bestanden voor u aanmaken uit bestaande XML-bestanden voor plugins.

U kunt zich misschien herinneren dat u dit in uw plugin-XML plaatste (als u dit nog niet heeft gedaan, doe het nu):

```
<document>
        [...]
        <help file="bestandsnaam.rkh" />
        [...]
</document>
```

Waarin u, natuurlijk, `bestandsnaam` vervangt door de juiste naam. Het is nu tijd het `.rkh`-bestand aan te maken. Hier volgt een voorbeeld dat voor zichzelf spreekt:

```
<!DOCTYPE rkhelp>
<document>
        <summary>
In deze sectie plaatst u wat erg korte basisinformatie over het doel van uw ↩
    plugin.
Deze sectie staat altijd helemaal bovenaan in de helppagina.
        </summary>

        <usage>
De gebruikssectie kan wat meer praktische informatie bevatten. Niet alle  ↩
    instellingen worden tot in detail uitgelegd (maar dat gebeurt in de  ↩
    sectie &#8220;instellingen&#8221;).

U kunt een nieuwe alinea beginnen met een lege regel, zoals u hierboven  ↩
    ziet.
Deze regel, daarentegen, is in dezelfde alinea.

In alle secties kunt u eenvoudige HTML-code invoegen, zoals <b> bold</b> of
```

```
<i>italic</i> text. Maar beperk dit formatteren tot het minimum nodige.

De gebruikssectie is altijd de tweede sectie in de help-pagina.
        </usage>

        <section id="sectionid" title="Algemene sectie" short_title= " ↩
            Algemeen">
Indien nodig kunt u secties toevoegen tussen de secties voor gebruiken  ↩
    instellingen.
Maar voor het documenteren van plugins heeft u dit gewoonlijk niet nodig.  ↩
    Het&#8220;id&#8221;-attribuut
geeft een ankerpunt voor het springen naar deze sectie vanuit het  ↩
    navigatiemenu.Het "short_title"
attribuut geeft een korte naam voor in de navigatiebalk. Dit is optioneel,  ↩
    standaard
wordt de hoofd"naam" gebruikt, zowel als kopnaam voor de sectie, als voor  ↩
    de naam van de koppeling
in de navigatiebalk.

In elke sectie kunt u koppelingen invoegen naar verdere informatie. U doet  ↩
    dit met het toevoegen van

<link href="URL">link name</link>

Waarin URL een externe koppeling kan zijn zoals http://rkward.kde.org .
Enkele speciale URL's worden in de help-pagina's ondersteund:

<link href="rkward://pagina/pad/pagina_id"/>

Dit koppelt naar een topniveau help-pagina in rkward (geen plugin).

<link href="rkward://component/[namespace/]component_id"/>

Dit koppelt naar de help-pagina van een andere plugin. Het [namespace/]  ↩
    deel kan worden weggelaten
(in dit geval wordt rkward aangenomen als standaard namespace, bv.:
<link href="rkward://component/import_spss"/> of
<link href="rkward://component/rkward/import_spss"/> zijn equivalent).
De component_id is dezelfde als die u opgaf in de .pluginmap.

<link href="rkward://rhelp/rfunction"/>

Kopppelt naar de help-pagina van R voor "rfunction".

Merk op dat de namen van de koppelingen van dit type koppelingen  ↩
    automatisch worden gegenereerd.
        </section>

        <settings>
                <caption id="id_of_tab_or_frame"/>
                <setting id="id_of_element">
Beschrijving van het GUI-element behorend bij de gegeven id
                </setting>
                <setting id="id_of_elementb" title="beschrijving">
Gewoonlijk komt de naam van het GUI-element automatisch uit de
XML-definitie van de plugin, Maar
voor sommige GUI-elementen is deze beschrijving onvoldoende voor  ↩
    betrouwbare identificatie.
```

```
In dat geval kunt u een expliciete naam toevoegen met het &#8220;title ↩
    &#8221;-attribuut.
                </setting>
                <setting id="id_of_elementc">
Beschrijving van het GUI-element behorend bij "id_of_elementc"
                </setting>
                [...]
        </settings>

        <related>
De bijbehorende sectie bevat typisch slechts enkele koppelingen, zoals:

<ul>
        <li><link href="rkward://rhelp/mean"/></li>
        <li><link href="rkward://rhelp/median"/></li>
        <li><link href="rkward://component/related_component"/></li>
</ul>
        </related>

        <technical>
De technische sectie (optioneel, altijd laatst) kan wat technische details ↩
    over de implementatie van
de plugin bevatten, die alleen van belang zijn voor RKWard-ontwikkelaars. ↩
    Dit is vooral van belang bij
plugins, die in veel andere plugins moeten worden ingebed, en kunnen ↩
    vermelden welke opties
er zijn voor het aanpassen van de ingebedde plugins, en welke ↩
    codesectieswelke
R code bevatten.
        </technical>
</document>
```

Hoofdstuk 7

Logische interacties tussen GUI-elementen

7.1 GUI-logica

Alle basisconcepten voor het maken van een plugin voor RKWard zijn beschreven in de vorige hoofdstukken. Dit zou voldoende moeten zijn voor veel -- zo niet de meeste -- gevallen. Maar soms wilt u toch meer controle hebben over het gedrag van de GUI van uw plugin.

Bij voorbeeld, stel dat u het t-testvoorbeeld in deze documentatie wilt uitbreiden om beide mogelijk te maken: het vergelijken van een variabele met een andere variabele (zoals getoond), en het vergelijken van een variabele met een constante waarde. Nu, een manier is een radioknop toe te voegen die tussen beide mogelijkheden schakelt, of het toevoegen van een spinveld voor de invoer van de constante waarmee moet worden vergeleken. Zie dit vereenvoudigde voorbeeld:

```
<!DOCTYPE rkplugin>
<document>
        <code file="code.js"/>

        <dialog label="T-Test">
                <row>
                        <varselector id="vars"/>
                        <column>
                                <varslot id="x" types="getal" source="vars" ↩
                                     required="true" label="vergelijken"/>
                                <radio id="mode" label="Vergelijken met">
                                        <option value="variabele" checked=" ↩
                                            true" label="een andere  ↩
                                            variabele (selecteer hieronder) ↩
                                            "/>
                                        <option value="constante" label=" ↩
                                            een constante (hieronder  ↩
                                            instellen)"/>
                                </radio>
                                <varslot id="y" types="getal" source="vars" ↩
                                     required="true" label="variable"  ↩
                                    i18n_context="Zelfst, nw.; een variabele ↩
                                    "/>
                                <spinbox id="constante" initial="0" label=" ↩
                                    constante" i18n_context="Zelfst. nw.;  ↩
                                    een constante"/>
```

```
                        </column>
                </row>
        </dialog>
</document>
```

Nou, dat is dan mooi, maar er zijn met deze GUI wel een paar problemen. Ten eerste zijn de varslot en het spinveld altijd allebei zichtbaar, terwijl maar één ervan werkelijk wordt gebruikt. Erger is dat er voor de varslot altijd een geldige sectie aanwezig moet zijn, zelfs als er met een constante wordt vergeleken. Het is duidelijk, dat als we een voor meerdere doeleinden geschikte GUI willen maken, we wat meer flexibiliteit wensen. Dit leidt tot: de **<logic>** sectie (ingevoegd op het zelfde niveau als **<code>**, **<dialog>**, of **<wizard>**).

```
[...]
        <code file="code.js"/>

        <logic>
                <convert id="varmode" mode="gelijk aan" sources="mode. ↩
                    string" standard="variable" />

                <connect client="y.visible" governor="varmode" />
                <connect client="constant.visible" governor="varmode.not"  ↩
                    />
        </logic>

        <dialog label="T-Test">
        [...]
```

De eerste regel in de logische sectie is een **<convert>**- tag (conversie, omzetten). In principe geeft dit een nieuwe boolean (aan of uit, true of false) eigenschap, die later kan worden gebruikt. Deze eigenschap (´´*varmode*´´) is waar wanneer de bovenste radioknop geselecteerd is, en false wanneer de onderste is geselecteerd. Hoe komt dat?

Ten eerste, onder *sources*, worden de broneigenschappen waarmee wordt gewerkt in een lijst genoemd (in dit geval voor elk een; u kunt meerderebronnen opnemen met *sources=´´mode.string;ietsanders´´*, dan is ´´*varmode*´´ alleen waar indien zowel ´´*mode.string*´´ als ´´*ietsanders*´´ gelijk zijn aan de ´´*variable*´´). Merk op dat in dit geval we niet alleen ´´*mode*´´ schrijven (zoals in `getString(´´mode´´)`), maar ´´*mode.string*´´. Dit is in principe hoe een radioknop werkt: die heeft een eigenschap 'string', die zijn string-waarde bevat.`getString(´´mode´´)` is alleen maar korter, en gelijk aan `getString(´´mode.string´´)`. Zie ook de beschrijving van alle eigenschappen van de GUI-elementen.

Ten tweede, stellen we de conversiemodus in op *mode=´´gelijk aan´´*. Dit betekent dat we willen controleren of de bron(nen) gelijk is (zijn) aan een bepaalde waarde. Tenslotte, is standaard de waarde waarmee we willen vergelijken, dus met *standard=´´variable´´* controleren we of de eigenschap ´´*mode.string*´´ gelijk is aan de string ´´*variable*´´ (de waarde van de bovenste radio-optie). Indien gelijk, is de eigenschap varmode true, en anders false.

Nu komt waar het allemaal om draait: We **<connect>** (verbinden) de eigenschap ´´*varmode*´´ met y.visible, die bepaalt of de varslot ´´*y*´´ visible (zichtbaar) is of niet. Merk op dat elk element dat onzichtbaar wordt gemaakt, impliciet niet vereist is. Dus als de bovenste radioknop wordt geselecteerd, is de varslot ´´*y*´´ vereist, en zichtbaar, en anders niet vereist en niet zichtbaar.

Voor het spinveld willen we precies het omgekeerde. Gelukkig hebben we hiervoor niet nog een **<convert>** nodig: Boolean eigenschappen kunnen zeer eenvoudig met ´´*not*´´ (dat is:niet), worden omgekeerd, dus **<connect>** *(verbinden) we* ´´*varmode.not*´´ met de zichtbaarheid-eigenschap van het spinveld. Hierdoor wordt of de varslot zichtbaar en vereist, *of* het spinveld - afhankelijk van welke optie werd gekozen met de radio-knoppen. De GUI past zichzelf aan afhankelijk van welke optie wordt gekozen. U kunt dit voorbeeld als u dit wilt, zelf uitproberen.

Voor een complete lijst van eigenschappen, zie de naslag. Er is echter nog een eigenschap, die alle GUI-elementen hebben: 'aangezet'. Dit is iets minder drastisch dan 'zichtbaar'. Het maakt

het GUI-element niet zichtbaar of niet, maar zet het alleen maar aan of uit. Uitgezette elementen worden gewoonlijk grijs getoond, en reageren niet op acties van de gebruiker.

> OPMERKING
> Behalve **<convert>** en **<connect>**, zijn er nog meer elementen die in de **<logic>** sectie kunnen worden gebruikt. Bijv. conditionele constructs kunnen ook worden geïmplementeerd met het **<switch>**-element. Zie de naslag voor logische elementen voor details.

7.2 GUI logica in scripts

Hoewel het verbinden van eigenschappen, zoals hierboven beschreven, vaak voldoende is, is het soms flexibeler of eenvoudiger, de GUI-logica in een script te beschrijven. Op die manier kan het bovenstaande voorbeeld worden herschreven naar:

```
[...]
        <code file="code.js"/>
'
        <logic>
                <script><![CDATA[
                        // ECMAScript-code in dit blok
                        // de top-level statement wordt maar een keer ↩
                            aangeroepen
                        gui.addChangeCommand ("mode.string", "modeChanged ↩
                            ()");

                        // deze functie wordt aangeroepen als "mode" wordt ↩
                            gewijzigd
                        modeChanged = function () {
                                var varmode = (gui.getString ("mode.string ↩
                                    ") == "variable");
                                gui.setValue ("y.enabled", varmode);
                                gui.setValue ("constante.enabled", !varmode ↩
                                    );
                        }
                ]]></script>
        </logic>

        <dialog label="T-Test">
        [...]
```

In de eerste regel staat dat RKWard de functie `modeChanged()` (modusGewijzigd)moet aanroepen als de waarde van de radioknop *id=´´mode´´* verandert. In deze functie definiëren we een hulpvariabele *´´varmode´´* die true is wanneer de modus *´´variabele´´* is, en false als die *´´constante´´* is. Daarna gebruiken we `gui.setValue()` om de eigenschappen op 'aan' te zetten van *´´y´´* en *´´constant´´*, op precies dezelfde manier als we dat eerder deden met de **<connect>** statements.

Het maken van scripts voor de GUI-logica is bijzonder nuttig wanneer u de beschikbare optie wilt aanpassen aan het type object dat door de gebruiker wordt geselecteerd. Zie welke functies er zijn in de naslag.

Merk op dat dit maken van scripts voor de GUI-logica kan samengaan met de statements **<connect>** en **<convert>** als u dat wilt.Ook maakt de **<script>**-tag het mogelijk de bestandsnaam van een script op te geven naast, of in plaats van, het Inlinen (invoegen) van de scriptcode. Het inlinen, zoals hierboven, is echter gewoonlijk het eenvoudigst.

Hoofdstuk 8

Plugins inbedden in Plugins

8.1 Voorbeelden van plugins in plugins

Als u plugins schrijft, merkt u vaak dat u een aantal plugins aan het maken bent die maar op enkele punten van elkaar verschillen, maar veel met elkaar gemeen hebben. Bijvoorbeeld, voor plotten, zijn er een aantal algemene R-opties, die kunnen worden gebruikt in bijna alle typen van plots. Moet u nu een GUI- en JS-sjabloon (template) voor elk apart aanmaken?

Het is duidelijk dat dat niet echt handig zou zijn. Gelukkig hoeft dat niet. U kunt beter de gemeenschappelijke functionaliteit één keer aanmaken, en dit later in andere plugins invoegen. Het is in feite mogelijk elke plugin in te voegen in elke andere plugin, ook als de originele schrijver er nooit aan gedacht zou hebben dat zijn plugin daarvoor zou worden gebruikt.

8.2 Invoegen in een dialoog

OK, genoeg. Hoe werkt dit? Eenvoudig: gebruik de **<embed>**-tag (embed is invoegen, opnemen, inbedden). Hier is een uitgekleed voorbeeld:

```
<dialog>
        <tabbook>
                <tab [...]>
                        [...]
                </tab>
                <tab label="Plot Opties" i18n_context="Opties voor de plot ↩
                    ">
                        <embed id="plotopties" component="rkward:: ↩
                            plot_options"/>
                </tab>
                <tab [...]>
                        [...]
                </tab>
        </tabbook>
</dialog>
```

Wat hier gebeurt, is dat de gehele GUI of de plugin met de plotopties (behalve natuurlijk voor de standaard elementen zoals de knop **Indienen** etc.) meteen in uw plugin worden opgenomen (probeer het maar!).

U ziet dat de syntaxis van de **<embed>**-tag tamelijk eenvoudig is. Zoals de meeste elementen heeft het een *id* (naam). De parameter-component geeft op welke plugin moet worden ingevoegd, zoals bepaald in het .pluginmap-bestand (*´´rkward::plot_opties´´* is het resultaat van

het aan elkaar plakken van de naamruimte (namespace) 'rkward', een scheidingsteken '::', en de naam van de component 'plot_opties').

8.3 Het genereren van code tijdens het invoegen

Mooi dan, maar hoe is het met de gegenereerde code? Hoe worden de codes van de invoegende en ingevoegde plugins in elkaar geschoven (merged)? In de JS code van de invoegende plugin schrijft u eenvoudig zoiets als het volgende:

```
function printout () {
        // ...
        echo ("myplotfunction ([...]" + getString ("plotopties.code. ↩
            printout"); + ")\n");
        // ...
}
```

Dus, in principe, halen we de code die gegenereerd is door de ingebedde plugin op, net zoals we elke andere GUI-instelling kunnen ophalen. Hier kan de string *´´plotopties.code.printout´´* worden ontrafeld tot: 'de af te drukken sectie van de gegenereerde code van het element met de *id* (naam) plotopties' (plotopties is de ID die we gaven aan de **<embed>**-tag hierboven). En ja, als u meer controle wilt, kunt u zelfs de waarden ophalen van de afzonderlijke GUI-elementen in deingebedde plugin (maar niet de andere kant op, omdat de ingebedde plugin niets weet over zijn omgeving).

8.4 Inbedden in een assistent (wizard)

Indien uw plugin een GUI voor een assistent heeft, gaat het inbedden op dezelfde manier. Gewoonlijk gebruikt u:

```
<wizard [...]>
                [...]
                <page id="page12">
                        [...]
                </page>
                <embed id="plotopties" component="rkward::plot_opties"/>
                <page id="pagina13">
                        [...]
                </page>
                [...]
        </wizard>
```

Indien de ingebedde plugin een assistent-interface heeft, worden de pagina's daarvan ingevoegd tussen de pagina's *´´pagina12´´* en *´´pagina13´´* van uw plugin. Indien de ingebedde plugin alleen een dialoog-interface heeft, wordt een enkele nieuwe pagina ingevoegd tussen die pagina's. De gebruiker zal hier niets van merken.

8.5 Minder ingebedde inbedding: knop voor verdere opties

Ook al is inbedden wel leuk, moet u toch oppassen dat niet te veel te doen. Teveel functies in een GUI maakt het moeilijk de relevante opties te vinden. Natuurlijk wilt u soms een groot aantal opties inbedden (zoals alle opties voor de `plot()`-functie), maar omdat die eigenlijk optioneel zijn, wilt u die niet prominent in uw GUI hebben.

Een alternatief voor inbedden is dit te doen 'als een knop':

```
<dialog>
        <tabbook>
                [...]
                <tab label="Opties">
                        [...]
                        <embed id="plotopties" component="rkward:: ↩
                            plot_opties" as_button="true" label="Plotopties  ↩
                            opgeven"/>
                </tab>
                [...]
        </tabbook>
</dialog>
```

In dit geval wordt een enkele drukknop aan uw plugin toegevoegd, met de naam **Plotopties opgeven**. Wanneer u op die knop drukt, komt er een afzonderlijke dialoog op, met alle opties van de ingebedde plugin. Zelfs als deze ingebedde GUI meest onzichtbaar is, kunt u de instellingen ervan ophalen zoals hierboven beschreven.

> LET OP
> Waarschijnlijk moeten 'knoppen' alleen worden worden gebruikt in plugins die nooit ongeldig kunnen zijn (door ontbrekende/ongeldige instellingen). Anders kan de gebruiker zijn code misschien niet indienen, en veel moeite hebben met het uitzoeken waarom dit niet kan, omdat de reden daarvan achter een of andere knop is verborgen.

8.6 Inbedden/definiëren van onvolledige plugins

Sommige plugins -- en in feite behoren de plotopties zoals in het voorbeeld hierboven daartoe -- zijn op zichzelf incompleet. Zij bevatten domweg niet de GUI-elementen nodig voor het selecteren van enkele belangrijke waarden. Zij zijn alleen bedoeld om ingebed te worden in andere plugins.

In hoeverre is de plugin voor plotopties incompleet? Wel, voor sommige plotinstellingen moet die de naam kennen van de objecten/expressies voor de x- en de y-as (eigenlijk is een daarvan voldoende). Maar er is geen mechanisme aanwezig voor het selecteren van die objecten, of die op enige andere manier in te voeren. Dus hoe moet de plugin die kennen?

In de logische sectie van de plugin voor plot_opties, zijn twee extra regels, die nog niet zijn besproken:

```
<logic>
                <external id="xvar" />
                <external id="yvar" />

                [...]
        </logic>
```

Hierin worden twee extra eigenschappen gedefinieerd in de plugin voor plot_opties, waarvan het enige doel is dat die worden verbonden (connected) aan enige (nu nog onbekende) eigenschap van de inbeddende plugin. In de plugin voor plot_opties worden die twee opties eenvoudigweg gebruikt zoals elke andere, en bijvoorbeeld zijn er aanroepen naar `getString(´´xvar´´)` in de JS sjabloon voor plot_opties.

Nu is er voor de incomplete plugin geen enkele manier waarop die weet waarin die zal worden ingebed, en welke relevante instellingen in de inbeddende plugin zullen worden gebruikt. Dus moeten we ook nog twee regels toevoegen aan de logische sectie van de inbeddende plugin:

```
<logic>
                [...]

                <connect client="plotoptions.xvar" governor="xvarslot. ↩
                    available" />
                <connect client="plotoptions.yvar" governor="yvarslot. ↩
                    available" />
        </logic>
```

Hier is niets nieuws, in principe, we hebben de **<connect>**-statements besproken in het hoofdstuk van de GUI logica. U verbindt (connect) de waarden in twee varslots (genaamd *´´xvarslot´´* en *´´yvarslot´´* in dit voorbeeld) met de 'externe' eigenschappen van de ingebedde plugin. En dat is het. Al het andere wordt verder automatisch geregeld.

Hoofdstuk 9

Werken met vele soortgelijke plugins

9.1 Overzicht van de verschillende werkwijzen

Soms wilt u misschien plugins ontwikkelen voor een aantal soortgelijke functies. Beschouw bijvoorbeeld de plots van verdelingsfuncties. Deze genereren tamelijk soortgelijke code, en het is natuurlijk wenselijk de grafische interfaces ervan op elkaar te laten lijken. En ook kunnen grote delen van de help-pagina's identiek zijn. In elke plugin zijn maar een klein aantal parameters verschillend.

De naïeve aanpak is één plugin te maken, en dan de gehele inhoud te kopiëren van de `.js`, `.xml`, en `.rkh`-bestanden, en daarna die paar onderdelen te wijzigen die verschillend zijn. Maar wat doet u als u later een spelfout ontdekt die in alle andere bestanden gekopieerd is? U zult ze allemaal stuk voor stuk moeten verbeteren. Een vermoeiend en taai werkje.

Een tweede aanpak is met behulp van inbedding. Maar dit is voor dit probleem soms niet erg handig, vooral omdat de 'brokken' die u in kunt bedden soms te groot zijn, en de indeling (layout) er door wordt beperkt.In deze gevallen kunnen concepten zoals het opnemen van `.js`-bestanden, opnemen van `.xml`-bestanden en korte stukjes erg nuttig zijn (maar zie gedachten over wanneer inbedden verkiesbaar is).

Een waarschuwing echter, voordat u begint te lezen: deze concepten kunnen het eenvoudiger maken met soortgelijke plugins te werken, en de onderhoudbaarheid en leesbaarheid van die plugins verbeteren (onderhouden is het eventueel later aanbrengen van verbeteringen). Maar teveel kan al spoedig leiden tot het omgekeerde effect. Gebruik dit dus voorzichtig.

9.2 Met behulp van de JS include statement

In RKWard-plugins kunt u gemakkelijk een scriptbestand in een ander includen (insluiten, opnemen). De waarde hiervan is onmiddellijk duidelijk, indien gedeelten van uw JS-code in plugins met elkaar overeenkomen. U definieert eenvoudig die gedeelten in een apart `.js`-bestand, en neemt die op in al die `.js`-bestanden van de plugins. Bijvoorbeeld, zoals in:

```
// Dit is een bestand met de naam "common_functions.js"

function doCommonStuff () {
        // haal mogelijk wat opties op, etc.
        // ...
        comment ("Dit is R-code voor gebruik in andere plugins\n");
        // ...
}
```

```
// Dit is een van uw gewone plugin .js bestanden

// includen (opnemen) van de gemene functies
include ("common_functions.js");

function calculate () {
        // doe iets
        // ...

        // invoegen gemene code
        doCommonStuff ();
}
```

Merk op dat het soms zelfs nog nuttiger is, dit om te keren en een 'kale' `preprocess()`, `calculate()`, en `printout()` -functies in een gemeenschappelijk bestand te definiëren, en deze die gedeelten aan te laten roepen (call), die in de diverse plugins verschillend zijn. Bijv.:

```
// Dit is een bestand met de naam "common_functions.js"

function calculate () {
        // doe dingen die in alle plugins hetzelfde zijn
        // ...

        // voeg iets toe dat verschillend is in de plugins
        getSpecifics ();

        // ...
}
```

```
// dit is een van uw gewone plugin .js-bestanden

// includen van de gemene functies
include ("common_functions.js");

// merk op: geen calculate()-functie wordt hier gegeven.
// maar in plaats daarvan in de common_functions.js.

function getSpecifics () {
        // enige R-code afdrukken
}
```

Een probleem waar u op moet letten, met deze techniek, is de scopes van de variabelen. Zie het handboek van JS voor scopes van variabelen. (de scope van een variabele is het codegebied waarin de variabele bestaat).

Deze techniek wordt veel gebruikt in de plugins voor (CLT)plots van verdelingsfuncties, dus kunt daar voorbeelden hiervan vinden.

9.3 Includen van `.xml`-bestanden

In principe is de zelfde mogelijkheid van het insluiten (include) van bestanden ook aanwezig voor de `.xml`, `.pluginmap` en `.rkh`-bestanden. Overal in deze bestanden kunt u een**<include>**-tag plaatsen, zoals u hieronder kunt zien. Het effect is dat de gehele inhoud van het XML-bestand (om precies te zijn: alles binnen de **<document>**-tag van dat bestand) letterlijk op dit punt in het bestand wordt ingesloten. Merk op dat u alleen een ander XML-bestand kunt includen.

```
<document>
        [...]
        <include file="een_ander_xml_bestand.xml"/>
        [...]
</document>
```

Het *file*-attribuut is de bestandsnaam relatief tot de directory waarin het huidige bestand zich bevindt.

9.4 <snippets> gebruiken

Waar met het includen van bestanden zoals in de vorige sectie aan bod kwam tamelijk veel mogelijk is, wordt het het nuttigst in combinatie met de **<snippets>**-opdracht. Snippets zijn kleinere secties die u kunt invoegen in een ander deel van het bestand. Een voorbeeld maakt dit het duidelijkst:

```
<document>
        <snippets>
                <snippet id="notitie">
                        <frame>
                                <text>
                Dit wordt op twee plaatsen in de GUI ingevoegd
                                </text>
                        </frame>
                </snippet>
        </snippets>
        <dialog label="test">
                <column>
                        <insert snippet="notitie"/>
                        [...]
                        <insert snippet="notitie"/>
                </column>
        </dialog>
</document>
```

Dus u definieert een snippet op een plaats bovenin het XML-bestand, en daarna **<insert>** (invoegen) u die op elke plaats die u wenst.

Hoewel dit voorbeeld op zichzelf niet bijster nuttig is, kunt u denken aan het combineren hiervan met een **<include>**d `.xml`-bestand. Merk op dat u in hetzelfde bestand ook snippets kunt plaatsen voor het `.rkh`-bestand. U **<include>** het bestand eenvoudig ook daar, en **<insert>** de relevante snippet:

```
<!-- Dit is een bestand genaamd "common_snippets.xml" -->
<document>
        <snippet id="common_opties">
                <spinbox id="iets" [...]/>
                [...]
        </snippet>
        <snippet id="common_notitie">
                <text>Een belangrijke notitie voor dit type plugin</text>
        </snippet>

        <snippet id="common_help">
                <setting id="iets">Dit doet iets</setting>
                [...]
```

```
        </snippet>
</document>
```

```
<!-- Dit is het .xml-bestand van de plugin -->
<document>
        <snippets>
                <!-- Importeer de common snippets -->
                <include file="common_snippets.xml"/>
        </snippets>

        <dialog label="test2">
                <insert snippet="common_notitie"/>
                <spinbox id="iets_plugin_specifiek" [...] />
                <insert snippet="common_opties"/>
        </dialog>
</document>
```

Gelijk aan inclusie in JS, is de omgekeerde aanpak vaak zelf nog nuttiger:

```
<!-- Dit is een bestand met de naam "common_layout.xml" -->
<document>
        <column>
                <insert snippet="notitie">
                [...]
                <insert snippet="plugin_parameters">
        </column>
        [...]
</document>
```

```
<!-- Dit is het .xml-bestand van de plugin -->
<document>
        <snippets>
                <snippet id="notitie">
                        <text>De notitie voor deze specifieke plugin</text>
                </snippet>

                <snippet id="plugin_parameters">
                        <frame label="Parameters specifiek voor deze plugin ↩
                            ">
                                [...]
                        </frame>
                </snippet>
        </snippets>

        <dialog label="test3">
                <include file="common_layout.xml"/>
        </dialog>
</document>
```

Het is tenslotte ook mogelijk snippets **<in te voegen>** in andere snippets, als: a) ze niet in elkaar zijn genest, en b) de **<snippets>**-sectie bovenin het bestand is geplaatst (voordat een geneste snippet wordt ingevoegd); dit omdat **<insert>**-statements worden gelezen van boven naar beneden.

9.5 <include> en <snippets> vs. <embed>

Op het eerste gezicht geven **<include>** en **<snippets>** mogelijkheden die nogal gelijk zijn aan inbedden: zij maken het mogelijk dat stukjes code in de plugins worden hergebruikt. Dus wat is het verschil tussen deze methoden van aanpak, en wanneer moet u welke gebruiken?

Het voornaamste verschil tussen deze concepten is dat in te bedden plugins een wat strakker geheel zijn. Zij combineren een complete GUI, code om R-code hieruit te genereren, en een helppagina. In contrast daarmee maken include en insert meer fijncontrole mogelijk, maar ten koste van minder modulariteit.

Dit betekent dat een plugin die een andere plugin inbedt niets hoeft te weten over de interne details van de ingebedde plugin. Een goed voorbeeld hiervan is de plugin met plot-opties. Plugins waarin deze plugin wordt ingebed, hoeven niet noodzakelijk alle beschikbare opties te kennen, of hoe die te gebruiken. Dit is goed, omdat anders, als een plugin met plot_opties wordt gewijzigd, het nodig zal zijn om alle plugins die de plugin hebben ingebed, hieraan moeten worden aangepast. In tegenstelling daarmee, tonen include en insert al hun interne details, en plugins die daarvan gebruik maken, moeten -- bijvoorbeeld -- alle ids (namen) en misschien zelfs typen kennen van de gebruikte elementen.

De vuistregel is daarom als volgt: include en insert zijn prachtig als de relevante opties slechts nodig zijn in een duidelijk beperkte groep plugins. Ingebedde plugins zijn beter wanneer de groep plugins waarvoor ze bruikbaar zijn niet helder afgebakend is, en als hun functionaliteit gemakkelijk kan worden verdeeld over modules. Nog een vuistregel: als de gemeenschappelijke gedeelten in een enkel 'brokstuk' kan worden gestopt, doe dat dan, en gebruik inbedden. Indien veel kleine snippets nodig zijn voor de gemeenschappelijke code -- wel -- gebruik dan **<snippets>**. Een laatste manier om het te bekijken: als alle plugins dezelfde *zeer* vergelijkbare functionaliteit hebben, zijn includes en inserts waarschijnlijk een goed idee. Indien zij slechts een of twee gemeenschappelijke 'modules' delen, is inbedden waarschijnlijk beter.

Hoofdstuk 10

Te gebruiken concepten voor gespecialiseerde plugins

Dit hoofdstuk bevat informatie over een aantal onderwerpen die alleen bruikbaar zijn voor bepaalde klassen van plugins.

10.1 Plugins die een plot maken

Het is gemakkelijk een plot te maken met een plugin. Maar er zijn een paar subtiele valkuilen die u moet vermijden, en ook wat handige algemene functionaliteit die u zou moeten kennen. In deze sectie vindt u de eenvoudigste concepten.Aan het eind is een eenvoudig voorbeeld dat u zou moeten volgen bij hetmaken van plot plugins.

10.1.1 Een plot tekenen in het uitvoervenster

Om een plot te tekenen in het uitvoervenster, gebruikt u `rk.graph.on()` direct voor het tekenen van een plot, en `rk.graph.off()`, direct daarna. Dit is vergelijkbaar met bijv. het aanroepen van `postscript()` en `dev.off()` in een normale R-sessie.

Maar het is belangrijk dat u *altijd* `rk.graph.off()` aanroept na de aanroep van `rk.graph.on()`. Anders wordt het uitvoerbestand beschadigd. Om er zeker van te zijn dat `rk.graph.off()` werkelijk wordt aangeroepen, moet u *alle* R-opdrachten tussen de twee statements inpakken in een `try()`-statement. Nooit van gehoord? Maak u geen zorgen, het is eenvoudig. Alles wat nodig is, is dat u het patroon volgt in onderstaand voorbeeld.

10.1.2 Toevoegen mogelijkheid voorbeeldweergave

> OPMERKING
> In deze sectie praten we over het toevoegen van de mogelijkheid van een voorbeeldweergave van plugins die plots produceren. Er zijn hier aparte secties over op voorbeelden (HTML) uitvoer, voorbeelden van (geïmporteerde) gegevens, en eigen voorbeelden. Het is echter aan te raden dit eerst te lezen, omdat de aanpak in alle gevallen dezelfde is.

Een erg nuttige eigenschap van alle plugins die plots/grafieken maken is dat ze een automatische voorbeeldweergave (preview) geven na een wijziging. Hiervoor moet u twee dingen doen: Een **<preview>** keuzevakje toevoegen aan de GUI-definitie, en de gegenereerde code voor de preview aanpassen.

Toevoegen van een **<preview>** keuzevakje is eenvoudig. Plaats het volgende ergens in uw GUI. Dit zorgt dan voor alles wat achter de schermen gebeurt wat nodig is voor het maken van een preview, waarbij deze wordt bijgewerkt wanneer er iets aan is gewijzigd, etc.. Voorbeeld:

> OPMERKING
> Te beginnen met de versie 0.6.5 van RKWard zijn de **<preview>** elementen speciale gevallen in plugin-dialogen (niet assistenten): zij worden geplaatst in de kolom voor knoppen, onafhankelijk van waar zij precies zijn gedefinieerd in de GUI. Het is toch nog een goed idee ze te definiëren op een logische plek in de indeling (layout), voor de compatibiliteit met oudere versies (backwards compatibility).

```
<document>
                [...]
                <dialog [...]>
                        [...]
                        <preview id="voorbeeld"/>
                        [...]
                </dialog>
                [...]
        </document>
```

En dat is het dan wat de GUI betreft.

Het aanpassen van de JS-sjabloon (template) is wat meer werk. U moet een nieuwe functie maken met de naam `preview()`, naast de functies `preprocess()`, `calculate()`, etc.. Deze functie moet code genereren die nodig is voor het maken van de plot, en alleen dat. Vooral geen kopteksten afdrukken, `rk.graphics.on()`, of dergelijke aanroepen. Zie het onderstaande voorbeeld, voor de typische manier waarop dit wordt gedaan:

10.1.3 Generieke plotopties

U zult gemerkt hebben dat de meeste plot-plugins in RKWard een groot aantal generieke opties hebben, bijv. voor as-aanpassingen en ruimte om de cijfers heen. Deze opties toevoegen aan uw plugin is eenvoudig. Zij worden verzorgd door een inbedbare plugin met de naam **rkward::plot_opties**. U kunt die zo in uw plugin inbedden:

```
<document>
                [...]
                <logic [...]>
                        <connect client="plotopties.xvar" governor="x. ↩
                            beschikbaar"/>
                        <set id="plotopties.toestaan_type" to="true"/>
                        <set id="plotopties.toestaan_ylim" to="true"/>
                        <set id="plotopties.toestaan_xlim" to="false"/>
                        <set id="plotopties.toestaan_log" to="false"/>
                        <set id="plotopties.toestaan_grid" to="true"/>
                </logic>
                <dialog [...]>
                        [...]
                        <embed id="plotopties" component="rkward:: ↩
                            plot_opties" as_button="true" label="Plot Opties ↩
                            "/>
                        [...]
```

```
                </dialog>
                [...]
        </document>
```

Hierdoor wordt een knop aan uw UI (User Interface) toegevoegd die een venster doet verschijnen met plotopties. De logische sectie is slechts een voorbeeld. U krijgt hiermee enige controle over de plugin voor de plotopties. U kunt hierover meer lezen op de help-pagina van de plugin voor plotopties (een link hiernaar is aanwezig in de help-pagina van alle plugins met de generieke opties).

Vervolgens moet u ervoor zorgen dat de code voor uw plotopties wordt toegevoegd aan de gegenereerde code voor uw plot. Dit doet u door de eigenschappen **code.preprocess**, **code.printout**, en **code.calculate** op te halen uit de ingebedde plugin voor plotopties, en in uw code in te voegen, zoals u kunt zien in het onderstaande voorbeeld.

10.1.4 Een kaal voorbeeld

U ziet hier een .JS-bestand als voorbeeld, dat u kunt gebruiken als sjabloon voor het maken van een plotplugin:

```
function preprocess () {
    // de "een_of_ander_pakket" is nodig om de plot aan te maken
    echo ("vereist (een_of_ander_pakket)\n");
  }

  function preview () {
    // we gebruiken alle stages van de algemene code. Alleen de functie  ↩
        printout ()  moet een klein beetje anders worden aangeroepen voor  ↩
        het plot-voorbeeld
    preprocess ();
    // calculate (); // in dit voorbeeld heeft de plugin geen functie  ↩
        calculate ().
    printout (true); // hier betekent 'true': Maak de plot aan, maar zonder ↩
          kop of andere uitvoer.
  }

  function printout (is_voorbeeld) {
    // If "is_voorbeeld"  false is, wordt de volledige code aangemaakt,  ↩
        inclusief kopteksten.
    // If "is_voorbeeld" true is, wordt allen het essentiële aangemaakt.

    if (!is_voorbeeld) {
      echo ('rk.header (' + i18n ("Een voorbeeldplot") + ')\n\n');
      echo ('rk.graph.on ()\n');
    }
    // Alleen de volgende sectie wordt aangemaakt als is_voorbeeld=true

    // onthoud: alles tussen rk.graph.on() en rk.graph.off() moet worden   ↩
        ingepakt in een try() statement:
    echo ('try ({\n');
    // voeg optie-instellende coderegels in, die moeten worden uitgevoerd  ↩
        voor de werkelijke plotopdrachten.
    // De code zelf  komt uit de ingebedde plugin met plot-opties.  ↩
        printIndentedUnlessEmpty() zorgt voor een mooi afdrukresultaat.
    printIndentedUnlessEmpty ('\t', getString ("plotoptions.code.preprocess ↩
        "), '', '\n');
```

```
    // maak de werkelijke plot. plotoptions.code.printout geeft het deel ↩
        van de algemene plot opties
    // dat moeten worden toegevoegd aan de plotaanroep zelf.
    echo ('plot (5, 5' + getString ("plotoptions.code.printout") + ')\n');

    // voeg optie-instellende coderegels toe die moeten worden uitgevoerd ↩
        na de werkelijke plot.
    printIndentedUnlessEmpty ('\t', getString ("plotoptions.code.calculate ↩
        "), '\n');
    echo ('})'\n);  // Het afsluiten van de try() statement

    if (!is_preview) {
      echo ('rk.graph.off ()\n');
    }
  }
```

10.2 Voorbeeldweergaven (previews) voor gegevens, uitvoer en andere resultaten

10.2.1 Previews van (HTML)-uitvoer

> OPMERKING
> In deze sectie bespreken we het toevoegen van preview functionaliteit voor plugins die uitvoer / HTML afdrukken. U wordt aangeraden eerst de aparte sectie over plot previews te lezen.

Het maken van een preview van HTML-uitvoer is bijna dezelfde procedure als het maken van een plot preview. In dit geval verzekert u zich ervan dat **preview()** de relevante **rk.print()/rk.results()** opdrachten genereert. Echter, in het algemeen is het handig de statements voor de kopregel in de preview weg te laten. Hier is een uitgekleed voorbeeld:

```
<!-- In the plugin's XML file -->>
        <dialog label="Importeer CSV-data" >
                <browser id="bestand" type="bestand" label="Bestandnaam"/>
                <!-- [...] -->>
                <preview id="preview" mode="output"/>
        </dialog>
>
```

Merk de specificatie op van *mode=´´output´´* in het **<preview>**-element.

```
// In het plugin's JS-bestand
        function preview () {
                // genereert code voor preview
                printout (true);
        }

        function printout (is_preview) {
                // genereert alleen een koptekst als is_preview==false
                if (!is_preview) {
                        new Header ("Dit is een koptekst").print ();
                }
                echo ('rk.print (result)');
        }
```

10.2.2 Voorbeeldweergaven (previews) van (geïmporteerde) gegevens

> OPMERKING
> In deze sectie bespreken we het toevoegen van preview functionaliteit voor plugins die gegevens aanmaken (importeren). U wordt aangeraden eerst de aparte sectie over plot previews te lezen.

Het maken van een preview van geïmporteerde gegevens (elk type gegevens dat door **rk.edit()** kan worden verwerkt), is bijna gelijk aan het maken van een plot preview. Hier is een uitgekleed voorbeeld van het maken van een preview van gegevens:

```
<!-- In het plugin's XML-bestand -->>
        <dialog label="Importeer CSV-data" >
                <browser id="bestand" type="bestand" label="Bestandnaam"/>
                <!-- [...] -->>
                <preview id="preview" active="true" mode="data"/>
        </dialog>
>
```

Merk op dat het **<preview>**-element deze keer de *mode=´´data´´* specificeert. *active=´´true´´* maakt eenvoudig de preview standaard actief.

```
// In het plugin's JS-bestand
        function preview () {
                // genereert de code voor preview
                calculate (true);
        }

        function calculate (is_preview) {
                echo ('imported <- read.csv (file="' + getString ("bestand ↩
                    ") /* [+ opties] */);
                if (is_preview) {
                        echo ('preview_data <- imported\n');
                } else {
                        echo ('.GlobalEnv$' + getString ("name") + ' >-  ↩
                            imported\n');
                }
        }

        function printout () {
                // [...]
        }
```

Ook nu weer genereert de functie **preview()** bijna dezelfde R-code als de functie **calculate()**, dus maken we een hulpfunctie **doCalcuate()**voor het ontbinden van de gemeenschappelijkeonderdelen. Het belangrijkste waar u op moet letten, is dat u de geïmporteerde gegevens moet toewijzen aan een object genaamd *preview_data* (binnen de huidige - lokale - omgeving). *Al het andere gebeurt automatisch* (grof gezegd, roept RKWard **rk.edit(preview_data)** aan, opgenomen in een aanroep van **.rk.with.window.hints()**).

OPMERKING
Hoewel previews een mooie eigenschap is, gebruiken zij wel hulpbronnen (resources). In het geval van voorbeeldweergaven van gegevens, kan het voorkomen dat het systeem dit niet aankan. Dit kan gebeuren bij het invoeren van enorme gegevensverzamelingen (datasets) (die te groot zijn om te kunnen worden geopend voor bewerking in het bewerkingsvenster van RKWard). Maar ook ˝normale˝ gegevensverzamelingen kunnen verkeerd worden geïmporteerd, waardoor een enorm aantal rijen of kolommen worden aangemaakt. *Het is sterk aan te raden dat u de preview_data* parameter beperkt tot dimensies die de preview mogelijk maakt, zonder dat het systeem merkbaar traag wordt (bijv. 50 rijen bij 50 kolommen zou meer dan genoeg zijn in de meeste gevallen).

10.2.3 Aangepaste voorbeeldweergaven (previews)

Het **<preview>**-element kan worden gebruikt voor het maken van previews voor elk type type ˝document˝-venster dat kan worden gekoppeld aan een workplace van RKWard. Behalve plots en gegevensvensters, zijn dit HTML -bestanden, R-scripts, en object opsommingsvensters (object summary windows). Voor deze laatste, moet u **<preview mode=˝custom˝>** gebruiken.

Als u de secties die plot preview en gegevens previews beschrijven heeft gelezen, heeft u al een algemene indruk van de werkwijze, maar ˝custom˝ (aangepast, eigen) previews eisen iets meer handwerk achter de schermen. De belangrijkste R-functie hier is **rk.assign.preview.data()**. De volgende korte code laat zien hoe uw gegenereerde (preview) R-code eruit kan zien voor een plugin die een tekstbestand als uitvoer genereert:

```
## Moet worden gegenereerd in de preview() code-sectie van een plugin
        pdata <- rk.get.preview.data("EENID")
        if (is.null (pdata)) {
                outfile <- rk.get.tempfile.name(prefix="preview", extension ↩
                    =".txt")
                pdata <- list(bestandnaam=outfile, on.delete=function (id)  ↩
                    {
                        unlink(rk.get.preview.data(id)$bestandnaam)
                })
                rk.assign.preview.data("SOMEID", pdata)
        }
        try ({
                cat ("Dit is een test", pdata$bestandnaam)
                rk.edit.files(file=pdata$bestandnaam)
        })
```

U moet hier de waarde *EENID* verkrijgen uit de *id*-eigenschap van het **<preview>**-element. Bv. met behulp van **getString (˝preview.id˝)** in het .js-bestand van de plugin.

10.3 Context-afhankelijke plugins

Tot dusverre hebben we aangenomen dat alle plugins altijd nuttig zijn, en allemaal geplaatst in het hoofdmenu. Echter, sommige plugins hebben alleen betekenis (of geven iets extra's), in een bepaalde context. Bijvoorbeeld een plugin die de inhoud van een R X11 graphics-apparaat exporteert, heeft duidelijk alleen nut wanneer die geplaatst wordt in het menu van een X11-apparaat, en niet in de hoofdmenubalk. Ook moet zo'n plugin het nummer kennen van het apparaat waarvoor het zou moeten werken, zonder ernaar te moeten vragen.

We noemen zulke plugins context-afhankelijk. In overeenstemming daarmee worden zij in het `.pluginmap`-bestand niet (of niet alleen) geplaatst in de hoofd **<hiërarchie>**, maar eerder in een **<context>**-element. Tot nu toe worden slechts twee soorten contexts ondersteund (later meer hierover): X11 en het importeren van bestanden. We zullen dit nog bespreken. Zelfs als u alleen

belangstelling heeft voor de context voor importeren, moet u ook even kijken naar die voor X11, aangezien die iets bewerkelijker is.

10.3.1 Context voor X11-apparaat

Om een plugin in de context van een X11-apparaat te gebruiken - dit betekent het plaatsen ervan in de menubalk van het venster dat u ziet na het in de console aanroepen van `x11()`, moet u het eerst als gebruikelijk declareren in het `.pluginmap`-bestand:

```
<document [...]>
        <components>
                [...]
                <component id="mijn_x11_plugin" file="mijn_x11_plugin.xml"  ↩
                    label="Een X11 context-plugin"/>
                [...]
        </components>
```

Echter, u hoeft die niet te definiëren in de hiërarchie (het kan wel, indien die ook nuttig is als top-level plugin):

```
<hierarchy>
                [...]
        </hierarchy>
```

In plaats daarvan, voeg een definitie toe van de ˝x11˝-context, en voeg het toe aan de menu's daar:

```
<context id="x11">
                [...]
                <menu id="bewerken">
                        [...]
                        <entry id="mijn_x11_plugin"/>
                </menu>
        </context>
</document>
```

In de logische sectie van de plugin xml, kunt u nu twee **<externe>** eigenschappen declareren: *`devnum`* en *`context`*. *`context`* (indien gedeclareerd) wordt ingesteld op *´´x11´´* als de plugin in deze context wordt geactiveerd. *`devnum`* wordt ingesteld op het nummer van het graphics-apparaat waarin het moet werken. En dat is alles.

10.3.2 Context voor importeren van gegevens

Voordat u deze sectie leest, is het handig eerst de sectie te lezen over de context voor X11-apparaat, omdat hierin de grondbeginselen staan.

De context *´´import´´* wordt gebruikt voor het declareren van filter plugins voor import-bestanden . U plaatst die gewoon in een context met *id=´´import´´* in het `.pluginmap`-bestand. Maar er is nog een probleempje bij het declareren van deze plugins: om een algemene selectiedialoog voor alle ondersteunde bestandstypen aan te bieden, moet u nog een beetje extra informatie declareren over uw component:

```
<document [...]>
        <components>
                [...]
```

```
                <component id="mijn_xyz_import_plugin" file=" ↩
                    mijn_xyz_import_plugin.xml" label="Importeren XYZ- ↩
                    bestanden">
                        <attribute id="format" value="*.xyz *.zyx" label=" ↩
                            XYZ-gegevensbestanden"/>
                </component>
                [...]
        </components>
        <hierarchy>
                [...]
        </hierarchy>
        <context id="importeren">
                [...]
                <menu id="importeren">
                        [...]
                        <entry id="mijn_xyz_import_plugin"/>
                </menu>
        </context>
        [...]
</document>
```

In de attribuutregel staat eenvoudig, dat bijbehorende extensies voor bestandsnamen `*.xyz` of `*.zyx` zijn, en de filterdialoog moet de naam 'XYZ-gegevensbestanden' hebben in de dialoog voor het filteren van bestanden.

U kunt twee **<externe>** eigenschappen declareren in uw plugin. *`bestandnaam`* wordt ingesteld op de geselecteerdebestandnaam, en *`context`* wordt ingesteld op *´´importeren´´*.

10.4 R bevragen naar informatie

In enkele gevallen wilt u misschien verdere informatie uit R ophalen, die dan in de UI van uw plugin wordt weergegeven. U wilt, bijvoorbeeld, een selectie aanbieden van de levels van een factor die de gebruiker heeft geselecteerd voor analyse. Vanaf versie 0.6.2 van RKWard is dit mogelijk. Voordat we beginnen, is het belangrijk dat u op de hoogte bent van een aantal valkuilen:

R-code, die gestart wordt vanuit de logica van een UI van een plugin, wordt verwerkt tijdens de event loop van R, wat betekent dat die kunnen lopen *terwijl* andere berekeningen aan de gang zijn. Dit is om er zeker van te zijn, dat de UI van uw plugin bruikbaar is, ook al is R druk bezig met andere dingen. Maar, dit maakt het werkelijk belangrijk dat uw code geen bijeffecten heeft, in het bijzonder:

- Maak *geen* toewijzingen (assignments) in .GlobalEnv of welke andere niet-lokale omgeving dan ook.
- Stuur *niets* naar het uitvoerbestand.
- Plot *niets* op het scherm.
- Doe, in het algemeen, *niets* dat zij-effecten heeft. Uw code kan *informatie inlezen*, maar niets *¨doen¨*.

Met dit in gedachten, is hier het algemene patroon. U gebruikt dit in een sectie van ge-scripte UI-logica:

```
<script><![CDATA[
                                last_command_id = -1;
                                gui.addChangeCommand ("variable", "update  ↩
                                    ()");
```

```
                                update = function () {
                                        gui.setValue ("selector.enabled",  ↩
                                            0);
                                        variable = gui.getValue ("variabele ↩
                                            ");
                                        if (variabele == "") return;

                                        last_command_id = doRCommand (' ↩
                                            levels (' + variabele + ')', " ↩
                                            commandFinished");
                                }

                                commandFinished = function (result, id) {
                                        if (id != last_command_id) return;  ↩
                                             // een ander antwoord is op  ↩
                                            komst
                                        als (typeof (result) == "undefined ↩
                                            ") {
                                                gui.setListValue ("selector ↩
                                                    .available", Array (" ↩
                                                    ERROR"));
                                                return;
                                        }
                                        gui.setValue ("selector.enabled",  ↩
                                            1);
                                        gui.setListValue ("selector. ↩
                                            available", result);
                                }
                ]]></script>
```

Hierin is, *variabele* een eigenschap met een objectnaam (bijv. in een **<varslot>**). Wanneer die verandert, wilt u het tonen van levels in de **<valueselector>**, met de naam *selector* bijwerken. De sleutelfunctie hier is **doRCommand()**, met als eerste parameter de opdracht welke string te starten, en als tweede parameter de naam van een aan te roepen functie, na de opdracht. Let erop dat de opdracht asynchroon wordt gedaan, wat de zaak wat complexer maakt. Ten eerste moet u er voor zorgen dat uw **<valueselector>** uit blijft staan, zolang het geen bijgewerkte informatie bevat. En verder kan het zijn dat er mogelijk meerdere opdrachten in een wachtrij staan, voordat u de eerste resultaten krijgt. Daarom krijgt elke opdracht een ˝id˝, die we opslaan in *last_command_id* voor een later gebruik.

Als de opdracht klaar is, wordt de daarin opgegeven functie aangeroepen (in dit geval dus *commandFinished*), met twee parameters: het antwoord zelf, en de id van bijbehorende opdracht. Het type van het antwoord is vergelijkbaar met dat in R, bijv. een numerieke array, als het antwoord numeriek (een getal) is. Het kan zelfs een **list()** in R zijn, maar in dit geval wordt het een JS **Array()** zonder namen.

Let erop dat zelfs dit voorbeeld een beetje eenvoudiger is gemaakt. In werkelijkheid moet u verdere voorzorgsmaatregelen treffen, bijv. het voorkomen van een extreem aantal levels in de selector. Het goede nieuws is dat u dit waarschijnlijk niet allemaal zelf moet doen. Bovenstaand voorbeeld komt uit de **rkward::level_select**-plugin, bijvoorbeeld, die u eenvoudig kunt inbedden in uw eigen plugin. U kunt zelfs een andere te starten expressie opgeven in plaats van **levels()**.

10.5 Naar het huidige object verwijzen

In veel plugins is het gewenst te werken met het 'huidige' object. Bijvoorbeeld zou een 'sorteer'-plugin de dataframe die op dit ogenblik wordt bewerkt voor sorteren, alvast kunnen selecteren.

De naam van het huidige object is beschikbaar voor plugins als een voorgedefinieerde eigenschap met de naam *current_object*. U kunt met deze eigenschap op de gebruikelijke manier verbinden. Is er geen huidig object, dan wordt de eigenschap een lege string.

Op dit ogenblik kan de *current_object* alleen maar van de class `data.frame` zijn, maar vertrouw hier aub. niet op, omdat dit in de toekomst zal worden uitgebreid tot andere gegevenstypen. Indien u alleen belangstelling heeft voor `data.frame`-objecten, verbindt u in plaats daarvan met de eigenschap *current_dataframe*. Of anders kunt u type-vereisten forceren door de juiste beperkingen te gebruiken voor uw **<varslot>**s, of met behulp van scripts voor de GUI-logica.

10.6 Een (aantal) opties herhalen

Soms wilt u een aantal opties herhalen voor een willekeurig aantal items. Bijv. stel dat u een plugin voor het sorteren van een data.frame wilt maken. U zou dit kunnen doen met het sorteren van een willekeurig aantal kolommen (in het geval dat de eerste kolommen met elkaar overeenkomen). Dit kan eenvoudig door de gebruiker meerdere variabelen te laten selecteren in een **<varslot>** met *multi=´´true´´*. Maar als u dit uitbreidt, bijv. door de gebruiker voor elke variabele te laten opgeven of die naar karakter/numeriek wordt omgezet, of dat het sorteren in de ene of in de andere richting moet, dan heeft u meer flexibiliteit nodig. Andere voorbeelden zouden zijn het plotten van meerdere lijnen in een plot (met keuze van object, lijnstijl, lijnkleur, etc., voor elke lijn), of een afbeelding (mapping) op te geven voor het terug-coderen van een verzameling oude naar nieuwe waarden.
Dit leidt tot de **<optionset>**. Laten we eerst een eenvoudig voorbeeld bekijken:

```
<dialog [...]>
        [...]
        <optionset id="verzameling" min_rows="1">
                <content>
                        <row>
                                <input id="voornaam" label="Voornamen" size ↩
                                    ="klein">
                                <input id="achternaam" label="Familienaam"  ↩
                                    size="small">
                                <radio id="geslacht" label="Geslacht">
                                        <optioncolumn label="Mannelijk"  ↩
                                            value="m"/>
                                        <optioncolumn label="Vrouwelijk"  ↩
                                            value="f"/>
                                </radio>
                        </row>
                </content>

                <optioncolumn id="voornamen" label="Voornamen" connect=" ↩
                    firstname.text">
                <optioncolumn id="achternaam" label="Familienaam" connect=" ↩
                    lastname.text">
                <optioncolumn id="geslacht" connect="geslacht.string">
        </optionset>
        [...]
</dialog>
```

Hier hebben we een UI gemaakt voor het opgeven van aantal personen (bijv. auteurs). De UI vereist de invoer van tenminste een persoon (*min_rows=´´1´´*). In het **<optionset>**-element beginnen we met het opgevenvan de **<content>**, bijv. die elementen die behoren tot de verzameling van opties. U kent de meeste elementen wel in de **<content>** al.

Vervolgens geven we de gewenste variabelen op die we willen lezen uit de optieverzameling (optionset)in ons JS-bestand. Om dat we te maken zullen hebben met een willekeurig aantal

items, kunnen we ons niet beperken tot het lezen van `getString (´´voornaam´´)` in JS. In plaats daarvan moeten we voor elke gewenste waarde, een **<optioncolumn>** opgeven. Voor de eerste optiekolom in het voorbeeld, betekent **<connect=˝voornaam.text˝>** dat de inhoud van het **<input>**-element ˝voornaam˝ wordt gelezen voor elk item. **<optioncolumn>**s waarvoor een *label* (naam) is gegeven, worden getoond, in een kolom met die naam. In JS kunnen we nu de voornamen ophalen van alle auteurs met behulp van `getList(´´verzameling.voornaam´´)`, `getList (´´set.achternamen´´)` voor de achternamen, en `getList(´´set.geslacht´´)` voor een array van ˝m˝/˝v˝strings.

Merk op dat er geen beperkingen zijn voor wat u in een **<optionset>** kunt plaatsen. U kunt zelfs ingebedde-componenten gebruiken. Net als met elk ander element is alles wat u moet doen het verzamelen van de gewenste uitvoervariabelen in een **<optioncolumn>**-specificatie. In het geval van ingebedde plugins, is dit vaak een sectie van de ˝code˝-eigenschap. Bijv.:

```
<dialog [...]>
        [...]
        <optionset id="verzameling" min_rows="1">
                <content>
                        [...]
                        <embed id="kleur" component="rkward::kleur_kiezer"  ↩
                            label="Kleur"/>
                </content>

                [...]
                <optioncolumn id="kleur_params" connect="kleur.code.afdruk ↩
                    ">
        </optionset>
        [...]
</dialog>
```

Natuurlijk kunt u ook UI-logica in een optionset gebruiken. Hiervoor zijn er twee manieren: U kunt dit zoals gebruikelijk doen door het maken van een verbinding (of script) in de hoofdsectie **<logic>** van uw plugin. Echter, u krijgt toegang tot de UI-elementen in het contents-gebied, met (bijv.) ˝verzameling.contents.voornaam.XYZ˝. Let op het voorvoegsel (prefix) ˝verzameling˝ (de *id* die u heeft toegekend aan de set (verzameling) en ˝contents˝ (inhoud). En anders kunt u een aparte **<logic>**-sectie als child element van uw **<optionset>** toevoegen. In dit geval zijn *id*s relatief ten opzichte van het contents-gebied, bijv. ˝voornaam.XYZ˝. Alleen het **<script>**-element is niet toegestaan in de sectie van een optionset. Als u een script wilt maken, moet u ook de **<logic>**-sectie gebruiken van de hoofdsectie **<logic>** van de plugin.

> OPMERKING
> Bij het maken van logische scripts in een optionset, kunt u alleen toegang krijgen tot de *huidige* content region (inhoudsgebied). Het is dus alleen mogelijk om elementen binnen een content region met elkaar te verbinden. Het verbinden van een eigenschap buiten de **<optionset>** met een eigenschap binnen de content region kan misschien nuttig zijn bij de initialisatie. Maar het wijzigen van de contents region na de initialisatie zal *niet* van toepassing zijn voor elementen die door de gebruiker al zijn gedefinieerd. Slechts de huidig geselecteerde item in de set (verzameling).

10.6.1 ˝Driven˝ optionsets

Tot dusver hebben we een **<optionset>** beschouwd, die knoppen levert voor het toevoegen / verwijderen van items. In sommige gevallen, echter, is het natuurlijker items te selecteren buiten de **<optionset>**, en alleen opties te leveren voor het aanpassen van sommige aspecten van elk item in een **<optionset>**. Bijv. stel dat u de gebruiker wilt toestaan een aantal objecten in een plot te plotten. Voor elk object moet de gebruiker in staat zijn de kleur op te geven. U *zou* dit kunnen oplossen door het plaatsen van een **<varselector>** en **<varslot>** in het **<content>**-gebied,

waardoor de gebruiker de items een voor een kan selecteren. De gebruiker zal echter veel minder hoeven te klikken als u in plaats daarvan **<varslot multi=˝true˝>** *buiten* de **<optionset>** gebruikt. Daarna verbindt u deze selectie van objecten aan een zogenoemde ˝driven˝ optionset. Hier volgt hoe u dit doet:

```
<dialog [...]>
        <logic>
                <connect client="verzameling.vars" governor="vars. ↩
                    beschikbaar"/>
                <connect client="verzameling.varnamen" governor="vars. ↩
                    available.shortname"/>
        </logic>
        [...]
        <varselector id="varsel"/>
        <varslot id="vars" label="Te plotten objecten"/>
        <optionset id="verzameling" keycolumn="var">
                <content>
                        [...]
                        <embed id="kleur" component="rkward::kleur_kiezer"  ↩
                            label="Lijnkleur"/>
                </content>

                [...]
                <optioncolumn id="vars" external="true">
                <optioncolumn id="varnamen" external="true" label=" ↩
                    Variabele">
                <optioncolumn id="kleur_params" connect="kleur.code.afdruk ↩
                    ">
        </optionset>
        [...]
</dialog>
```

We beginnen met het bekijken van het voorbeeld onderaan. U ziet dat twee **<optioncolumn>** specificaties *external=´´true´´* hebben. Dit vertelt RKWard dat deze bestuurd bestuurd worden van buiten de **<optionset>**. Hier is het enige doel van de ˝varnamen˝-option column (optiekolom) is het leveren van gemakkelijk leesbare namen bij het tonen van de optionset (het is verbonden met de ˝shortname˝ modifier(veranderen in korte naam) van de eigenschap die de geselecteerde objecten inhoudt). Het doel van de ˝vars˝-option column is te dienen als de ˝key˝-column (sleutelkolom), zoals opgegeven door **<optionset keycolumn=˝vars˝...>**. Dit betekent dat voor elk item in de lijst de verzameling een van de beschikbare opties zal aanbieden, en opties hier logisch verbonden zijn met deze items. Deze kolom is verbonden met de eigenschap die de geselecteerde objecten inhoudt in **<varslot>**. Dus voor elk element dat daar wordt gekozen, kan met de **<optionset>** de kleur worden opgegeven.

> OPMERKING
> Een externe kolom kan ook worden *verbonden* met eigenschappen in het **<content>**-gebied. Maar het is belangrijk op te merken dat optioncolumns die *external=´´true´´* zijn gedeclareerd, nooit binnen **<optionset>** mogen worden gewijzigd, en optioncolumns die *external=´´false´´* (de standaard) zijn gedeclareerd, nooit buiten de **<optionset>** mogen worden gewijzigd..

10.6.2 Alternatieven: wanneer optionsets niet te gebruiken

Optionsets zijn een krachtig gereedschap, maar zij kunnen soms meer kwaad doen dan goed, omdat zij een enorme complexiteit toevoegen, zowel voor een plugin-ontwikkelaar, als voor een gebruiker. Denk dus twee keer na voordat u die gebruikt. Hier volgt wat advies:

- In eenvoudige gevallen, is het**<matrix>**-element misschien een nuttig lichtgewicht alternatief.
- Laat uw plugin niet te veel doen. We gaven een voorbeeld van het gebruik van een optionset (verzameling van opties) voor een plugin voor het tekenen van een aantal lijnen in een plot. Maar in het algemeen is het geen goed idee een plugin te maken voor aparte plots voor elk item in een optionset. Beter is het dat de plugin een plot produceert, die de gebruiker meerdere keren kan aanroepen.
- Indien u niet meer dan twee of drie items in een set verwacht, overweeg dan de opties met de hand te herhalen.

Hoofdstuk 11

Omgaan met afhankelijkheden (dependencies) en compatibiliteitsproblemen.

11.1 Versie-compatibiliteit van RKWard

We doen ons best, ervoor te zorgen dat een plugin, gemaakt voor een oude versie van RKWard, ook werkt in latere versies. Echter, het omgekeerde is niet altijd het geval, omdat nieuwe eigenschappen zijn toegevoegd. Omdat niet iedereen de laatste versie van RKWard gebruikt, betekent dit dat uw plugin niet voor iedereen werkt.

Indien u weet heeft van zulke problemen met compatibiliteit, moet u dit melden in de documentatie in uw `.pluginmap`-bestand, met het **<dependencies>** (afhankelijkheden)-element. De **<dependencies>** kunnen ofwel worden opgegeven als `.pluginmap`'s <document>-element, of als een child element van afzonderlijke **<component>**-definities. In het eerste geval zijn afhankelijkheden van toepassing op *alle* plugins in de map. In het andere geval alleen maar op het/de afzonderlijke **<component>**(en). U kunt ook top ˝globale˝ en ˝specifieke˝ afhankelijkheden mixen. In dat geval worden de ˝globale˝ afhankelijkheden toegevoegd aan die van de afzonderlijke component.

Bekijken we een klein voorbeeld:

```
<document ...>
        <dependencies rkward_min_version="0.5.0c" />
        <components ...>
                <component id="mijnplugin" file=" ↩
                    gereduceerde_versie_van_mijnplugin.xml" ...>
                        <dependencies rkward_max_version="0.6.0z" />
                </component>
                <component id="mijnplugin" file=" ↩
                    fantasie_version_van_mijnplugin.xml" ...>
                        <dependencies rkward_min_version="0.6.1" />
                </component>
                ...
x        </components ...>
</document>
```

In dit voorbeeld werken alle plugins alleen in versies 0.5.0c of later van RKWard. Een plugin, met *id=´´mijnplugin´´* wordt beschikbaar gesteld in twee alternatieve varianten. De eerste,

uitgeklede, versie wordt gebruikt in versies van RKWard eerder dan 0.6.1. De tweede gebruikt eigenschappen die nieuw zijn in versies 0.6.1 van RKWard.

Het beschikbaar maken van alternatieve varianten als deze is een erg gebruiksvriendelijke manier van het gebruik maken van nieuwe eigenschappen, terwijl toch de eerdere versies van RKWard bruikbaar blijven. Alternatieve versies moeten de zelfde `id` delen (anders komen er waarschuwingen), en kunnen alleen *in hetzelfde* `.pluginmap`-bestand worden gedefinieerd.

Plugins die niet compatibel zijn met (niet werken in) de gebruikte versie van RKWard, en die geen alternatieve versie bevatten, worden genegeerd met een waarschuwing.

> OPMERKING
> In werkelijkheid is RKWard 0.6.1 de eerste versie die sowieso afhankelijkheden herkent - en die meldt met foutmeldingen - . Dus, in tegenstelling tot wat het voorbeeld misschien suggereert, zal het opgeven van eerdere versies in de afhankelijkheden niet direct effect hebben (maar toch wel een goed idee zijn voor documentatie-doeleinden).

Soms is het zelfs mogelijk om met incompatibiliteitsproblemen van versies om te gaan *in* een enkel `.pluginmap`-bestand met het **<dependency_check>**-element, dat in de volgende sectie wordt beschreven.

11.2 R versie-compatibiliteit

Net zoals `rkward_min_version` en `rkward_max_version`, maakt het **<dependencies>**-element het mogelijk de attributen `R_min_version` en `R_max_version` op te geven. Maar er zijn de volgende verschillen:

- Plugins die niet voldoen aan de vereisten voor de R-versie worden op dit moment *niet* overgeslagen bij het inlezen van het `.pluginmap`-bestand. De gebruiker kan nog steeds de plugin aanroepen (call) en krijgt niet direct een waarschuwing krijgen (in toekomstige versies komt er waarschijnlijk wel een waarschuwing).
- Als gevolg daarvan is het ook *niet* mogelijk alternatieve versies van een plugin te definiëren, afhankelijk van de gebruikte versie van R.
- Echter, het is vaak eenvoudig terugwaartse compatibiliteit (backwards compatibility) te bereiken, zoals u hieronder kunt zien. Indien u compatibiliteitsproblemen van R onderkent, kunt u, als u dit wilt, deze methode gebruiken, in plaats van het definiëren van een afhankelijkheid (dependency) van een afzonderlijke versie van R.

In veel gevallen is het eenvoudig mogelijk een verminderde functionaliteit beschikbaar te stellen, indien een bepaalde eigenschap niet beschikbaar is in de gebruikte versie van R. Zie het volgende korte voorbeeld van een .xml-bestand van een plugin:

```
<dialog [...]>
        <logic>
                <dependency_check id="ris210" R_min_version="2.10.0"/>
                <connect client="compression.xz.enabled" governor="ris210 ↩
                    "/>
        </logic>
        [...]
        <radio id="compressie" label="Compressie-methode">
                <option label="Geen" value="">
                <option label="gzip" value="gzip">
                <option id="xz" label="xz" value="xz">
        </radio>
        [...]
</dialog>
```

In dit voorbeeld wordt de compressie-optie ˝xz˝ eenvoudig uitgezet indien de gebruikte versie van R ouder is dan 2.10.0 (waarin xz-compressie niet wordt ondersteund). Het **<dependency_check>**-element ondersteunt hetzelfde **<dependencies>**-element als in `.pluginmap`-bestanden. Het maakt een booleaanse eigenschap aan, die true is, indien aan de opgegeven afhankelijkheden wordt voldaan, en anders false.

11.3 Afhankelijkheden van R-pakketten

Afhankelijkheden van specifieke R-pakketten kunnen worden gedefinieerd, maar vanaf RKWard 0.6.1 worden deze afhankelijkheden automatisch noch gecontroleerd, noch geïnstalleerd / ingelezen. Maar zij zijn te zien in de plugin help-bestanden. Hier is een voorbeeld van een definitie:

```
<dependencies>
                <package
                        name="heisenberg"
                        min_version="0.11-2"
                        repository="http://rforge.r-project.org"
                />
        </dependencies>
```

> OPMERKING
> Zorg er altijd voor dat u de geëigende `require()` aanroepen (calls) toevoegt wanneer voor uw plugin bepaalde pakketten nodig zijn.

> OPMERKING
> Indien u uw `.pluginmap` als een R-pakket uitgeeft, en alle plugins van één bepaald pakket afhankelijk zijn, dan moet u die afhankelijkheid definiëren op het niveau van het R-pakket. Het definiëren van afhankelijkheden van R-pakketten op het niveau van de RKWard-`.pluginmap` is het nuttigst, indien alleen enkele van uw plugins die nodig hebben, de afhankelijkheid niet in CRAN beschikbaar is, of wanneer uw `.pluginmap` niet uitgegeven wordt als een R-pakket.

11.4 Dependencies (afhankelijkheden) van andere RKWard `.pluginmaps`

Indien uw plugins afhankelijk zijn van plugins in een andere `.pluginmap` (die *geen* onderdeel is van uw pakket) kunt u deze dependency als volgt definiëren:

```
<dependencies>
                <pluginmap
                        name="heisenberg_plugins"
                        url="http://eternalwondermaths.example.org/hsb"
                />
        </dependencies>
```

Op dit moment wordt niet ingelezen, noch geïnstalleerd, en zelfs niet gewaarschuwd voor ontbrekende `.pluginmaps`, maar u krijgt u tenminste informatie over dependencies (en waar die te verkrijgen) in de help-pagina van de plugin. U hoeft niet (en moet niet) dependencies declareren van `.pluginmaps` die mee worden geleverd met de officiële RKWard-distributie, of van `.pluginmaps` in uw eigen pakket.Verder, indien een vereiste `.pluginmap` wordt uitgegeven als een R-pakket,

moet u een dependency van het pakket declareren (zoals getoond in de vorige sectie), en niet van de map.

Om er zeker van te zijn dat de vereiste `.pluginmap`s werkelijk wordeningelezen, moet u de **<require>**-tag gebruiken (zie de naslag voor details).

11.5 Een voorbeeld

Om uit te leggen hoe definities van dependencies (afhankelijkheden) door elkaar kunnen worden gebruikt, is hier een gecombineerd voorbeeld:

```
<document ...>
        <dependencies rkward_min_version="0.5.0c">
                <package
                        name="heisenberg"
                        min_version="0.11-2"
                        repository="http://rforge.r-project.org"
                />
                <package
                        name="DreamsOfPi"
                        min_version="0.2"
                />
                <pluginmap
                        name="heisenberg_plugins"
                        url="http://eternalwondermaths.example.org/hsb"
                />
        <dependencies>

        <require map="heisenberg::heisenberg_plugins"/>

        <components ...>
                <component id="mijnplugin" file=" ↩
                    gereduceerde_versie_van_mijnplugin.xml" ...>
                        <dependencies rkward_max_version="0.6.0z" />
                </component>
                <component id="mijnplugin" file=" ↩
                    fantasie_versie_van_mijnplugin.xml" ...>
                        <dependencies rkward_min_version="0.6.1" />
                </component>
                ...
x        </components ...>
</document>
```

Hoofdstuk 12

Plugin vertalingen

Tot dusver hebben we in het voorbijgaan een paar concepten gebruikt met betrekking tot vertalingen (of kortweg ˝i18n˝: ˝internationalization˝ dat met 18 karakters tussen de i en de n wordt geschreven). In dit hoofdstuk vertellen we meer over de i18n-functionaliteit in plugins voor RKWard. Meestal heeft u die *niet* allemaal nodig voor uw plugins. Maar het kan toch wel handig zijn dit hoofdstuk in zijn geheel aandachtig te lezen, omdat het begrip van deze concepten u in staat kan stellen plugins te maken die goed vertaalbaar zijn, en een hoge kwaliteit mogelijk maakt van de vertalingen.

12.1 Algemene beschouwingen

Een belangrijk aspect van software-vertalingen, dat u moet begrijpen, in tegenstelling tot dat van teksten, is dat het voor vertalers vaak heel lastig is een compleet plaatje te hebben van *wat* ze eigenlijk vertalen. Vertalingen van software berusten noodzakelijkerwijs op korte tekstfragmenten: elke label (naam) die u aan een **<option>** in een **<radio>** geeft, elke string die u markeert voor vertaling in een aanroep (call) van de **i18n()**-functie, vormen een aparte ˝vertaaleenheid˝. In essentie wordt elk zo'n fragment geïsoleerd aan de vertaler aangeboden. Wel, niet helemaal geïsoleerd, omdat we proberen de vertaler zoveel mogelijk betekenisvolle context aan te bieden als dat maar automatisch mogelijk is. Maar in een aantal gevallen moeten vertalers extra context hebben, om een string te kunnen begrijpen, vooral bij korte strings.

12.2 i18n in RKWard's xml-bestanden

In RKWard's xml-bestanden, is i18n meestal geen probleem. Als u uw eigen **.pluginmap** schrijft (bijv. voor een externe plugin), moet u een `po_id` opgeven naast de pluginmap's `id` (naam). Dit definieert de te gebruiken zogenoemde ˝message catalog˝ (berichtencatalogus). In het algemeen moet die gelijk zijn aan de `id` van uw **.pluginmap**, maar indien u meerdere **.pluginmap**s heeft en wilt regelen hoe de message catalogs moeten worden opgesplitst , dan kunt u meerder po_ids gebruiken. De `po_id` wordt inherited (geërfd, doorgegeven aan) door elke **.pluginmap** die u bijsluit, behalve als u die een andere `po_id` declareert, en door alle plugins plugins die erin zijn gedeclareerd.

Voor plugins en help-pagina's hoeft u RKWard niet te vertellen welke strings vertaald moeten worden, omdat dit gewoonlijk uit hun gebruik blijkt. Maar, zoals hier boven uitgelegd, moet u letten op strings die op meer manieren kunnen worden begrepen, of waar wat uitleg bij nodig is voor de juiste vertaling. `i18n_context` zoals dit:

```
<checkbox id="schaal" label="Schaal" i18n_context="Schaal tonen"/>
<checkbox id="schaal" label="Schaal" i18n_context="De plot schalen"/>
```

Door *i18n_context* op te geven worden de twee strings afzonderlijk vertaald. Bovendien ziet de vertaler de context. Het *i18n_context*-attribuut wordt ondersteund in alle elementen die ergens vertaalbare strings bevatten, met inbegrip van elementen waarbinnen tekst aanwezig is (bijv. **<text>**-elementen).

In andere gevallen heeft de te vertalen string een enkele niet-dubbelzinnige betekenis, maar is er toch nog wat uitleg gewenst. In dat geval kunt u commentaar toevoegen die zichtbaar is voor vertalers. Voorbeelden hiervan kunnen zijn:

```
<!-- i18n: Nee, dit is geen typefout voor screen plot! -->
<component id="scree_plot" label="Scree plot"/>

<!-- i18n: Houd zo mogelijk deze string kort. Meer dan  zo'n 15 karakters
ziet er hier niet goed uit, en de betekenis moet duidelijk zijnvoor de
gebruiker (selectie uit lijst met waarden naast dit element) -->
<valueslot id="geselecteerd" label="Een kiezen"/>
```

Merk op dat zulk commentaar vooraf moet gaan aan het element waarop het betrekking heeft, en moet beginnen met of ˝i18n:˝ of ˝VERTALERS:˝

Tenslotte, in zeldzame gevallen, wilt u dat bepaalde strings niet vertaald zullen worden. Dit is begrijpelijk, bijvoorbeeld, als u laat kiezen tussen functienamen met behulp van **<radio>**-rondjes. U wilt dan niet dat deze worden vertaald, (maar afhankelijk van de context, moet u duidelijke namen overwegen):

```
<radio id="transformatie" label="toe te passen R-functie">
  <option id="as.list" noi18n_label="as.list()"/>
  <option id="as.vector" noi18n_label="as.vector()"/>
  [...]
</radio>
```

Merk op dat u dan het *label*-attribuut weglaat, en in plaats daarvan *noi18n_label* opgeeft. Ook, dat in tegenstelling tot *i18n_context* en commentaar, het gebruiken van *noi18n_label* uw plugin incompatibel maakt met versies van RKWard eerder dan 0.6.3.

12.3 i18n in RKWard's js-bestanden en secties

In tegenstelling tot de .xml-bestanden vereist het vertaalbaar maken van de js-bestanden van een plugin meer aanpasarbeid. Het grote verschil is hier dat er geen goede automatische manier is, aan te geven dat een string moet worden weergegeven als een voor een mens leesbare tekst, of als een stuk code. Dus zult u dit zelf moeten aanmerken. We hebben hier gaandeweg al voorbeelden van gezien. Hier volgt een completere beschrijving van de i18n-functies in js-code, en een aantal tips voor complexere gevallen:

i18n (msgid, [...])

De belangrijkste functie. Merkt de string aan om vertaald te worden. De string (vertaald of niet) wordt teruggegeven tussen dubbele aanhalingstekens (˝˝). Een willekeurig aantal plaatshouders (open plaatsen, place holders) kunnen in de string worden gebruikt zoals hieronder te zien is. Zulke plaatshouders in plaats van het aan elkaar plakken van kleine substrings is veel eenvoudiger voor vertalers:

```
i18n ("Vergelijken objecten %1 en %2", getString ('x'), getString ('y') ↩
    );
```

i18nc (msgctxt, msgid, [...])

Zelfde als **i18n()**, maar geeft ook nog eencontext voor de message:

```
i18nc ("eigen naam, niet geestestoestand", "Mood test"); (Vert.: Engels ↩
    : Mood is stemming, humeur, én er bestaat een statistische test met ↩
     de naam Mood test)
```

i18np (msgid_singular, msgid_plural, n, [...])

Zelfde als **i18n()**, maar voor messages die in enkelvoud en meervoud verschillend zijn (en in sommige talen zijn er zelfs meer numerieke vormen). Merk op dat precies zoals met **i18n()**, u een willekeurig aantal vervangingen (place holders) kunt gebruiken, maar de eerste ('%1') is verplicht, en moet een geheel getal zijn.

```
i18np ("Vergelijken enkel paar", "Vergelijken "%1" verschillende paren ↩
    ", n_paren);
```

i18ncp (msgctxt, msgid_singular, msgid_plural, n, [...])

i18np() met toegevoegde message context.

comment (comment, [indentation])

Echo't een commentaar bij de code, gemerkt voor vertaling. In tegenstelling tot de andere i18n()-functies, niet tussen aanhalingstekens, maar met een '#' voor iedere commentaarregel.

```
comment ("Transponeer de matrix");
                        echo ('x <- t (x)\n');
```

Voor het toevoegen van commentaar voor de vertalers (zie boven voor een discussie over de verschillen tussen commentaar en context), voeg commentaar toe dat begint met ˝i18n:˝ of ˝translators:˝ onmiddellijk boven de **i18n()**-aanroep voor commentaar. Bijv.:

```
// i18n: Spelling is correct: Scree plot.
               echo ('rk.header (' + i18n ("Scree plot") + ')\n');
```

12.3.1 i18n en aanhalingstekens

Meestal hoeft u zich geen zorgen te maken over het gedrag van i18n() met betrekking tot aanhalingstekens. Omdat vertaalbare strings meestal string literals (stringconstanten) zijn, is het aanhalen ervan prima, en bespaart dit u veel typewerk. En ook, in functies zoals **makeHeaderCode()/Header()** die gewoonlijk hun argumenten aanhalen, worden ge-i18n()'de strings beschermd tegen dubbel aanhalen.In wezen werkt dit door het sturen van de vertaalde string, eerst door **quote()** (om het aangehaald te maken), daarna door **noquote()** (om te voorkomen dat die nog een keer wordt aangehaald). Mocht u een vertaalbare string nodig hebben zonder aanhalingstekens, dan gebruikt u **i18n(noquote (˝Mijn bericht˝))**. Indien u een vertaalbare string nodig heeft die nog een keer moet worden aangehaald, dan stuurt u die *twee keer* door **quote()**.

Dat gezegd hebbende, is het gewoonlijk geen goed idee om zaken zoals functienamen of namen van variabelen vertaalbaar te maken. Ten eerste is R, de programmeertaal in wezen Engels, en is de taal zelf niet geïnternationaliseerd. Commentaar bij code is iets geheel anders, maar daarvoor moet u de **comment()**-functie gebruiken. Tweedens, het vertaalbaar maken van syntactisch relevante stukken van de gegenereerde code betekent dat vertalingen uw plugin onbruikbaar kunnen maken. Bijv. als een nietsvermoedende vertaler een variabelenaam vertaalt in twee aparte woorden, met daar tussen een spatie (Vert.: ik heb de stukjes programmacode zo voorzichtig mogelijk vertaald, maar ik ben me ervan bewust, dat dit wel eens mis kan zijn gegaan. Maar die stukjes programmacode zijn ook bedoeld als illustratie, en dat kan het best in het Nederlands. Elders in dit document worden alle stukjes programmacode als één groot geheel weergegeven, en daar heb ik NIET aangezeten).

12.3.2 i18n en backwards compatibility (terugwaartse compatibiliteit)

Een ongelukkig aspect van het niet aanwezig zijn van ondersteuning voor **i18n()** in RKWard versies tot aan 0.6.2, is dat toevoegen van **i18n()** aanroepen (calls) maakt dat voor uw plugin RKWard versie 0.6.3 nodig is. Indien uw plugin buiten RKWards officiële uitgave om is ontwikkeld, kan dit een probleem zijn. Hier zijn een paar mogelijke opties om hier mee om te gaan:

- Lever de plugin in twee versies, voor RKWard >= 0.6.3 en RKWard < 0.6.3, zoals beschreven in het hoofdstuk met dependencies omgaan
- Gewoon de strings in het .js-bestand nog niet vertalen. Het is duidelijk dat dit wel een gemakkelijke, maar niet erg elegante oplossing is.
- Voeg enige ondersteuningscode toe aan uw .js-bestand(en), zoals hieronder:

```
// js-functie "comment" is niet aanwezig voor 0.6.3
                        if (typeof (comment) == 'undefined) {
                                // definieer functie i18n(), en andere  die ↩
                                    u nodig heeft. Merk op dat uw  ↩
                                    implementatie misschien eenvoudiger is  ↩
                                    dan
                                // hier getoond, bijv. als u geen gebruik  ↩
                                    maakt van placeholders.
                                i18n = function (msgid) {
                                        var ret = msgid;
                                        for (var i = 1; i < arguments. ↩
                                            length; i++) {
                                                ret = ret.replace(new  ↩
                                                    RegExp("%" + i, 'g'),  ↩
                                                    arguments[i]);
                                        }
                                        if (msgid.noquote) {
                                                ret.noquote = msgid.noquote ↩
                                                    ;
                                                return (ret);
                                        }
                                        return (noquote (quote (ret)));
                                }
                        }
```

12.4 Onderhoud van vertalingen

Nu uw plugin vertaalbaar is gemaakt, hoe kunt u het werkelijk laten vertalen? In het algemeen hoeft u zich daar alleen zorgen over te maken bij het ontwikkelen van eenexterne plugin. Voor plugins in de hoofd-repository van RKWard wordt al deze magie voor u gedaan. Hier is in principe hoe dit werkt. Let op dat u de ˝gettext˝-hulpmiddelen moet hebben geïnstalleerd:

- Merk alle strings aan, en geef context op en commentaar waar nodig
- Laat **python scripts/update_plugin_messages.py --extract-only /pad/naar/mijn.pluginmap** werken. scripts/update_plugin_messages.py is thans niet aanwezig in de bronuitgaven (source releases), maar kan worden gevonden in de repository voor bronbestanden.
- Geef het resulterende **rkward__*POID*.pot**-bestand uit aan uw vertalers. Voor externe plugins wordt aangeraden ze te plaatsen in een submap ˝po˝ in inst/rkward.

- De vertaler opent het bestand in een vertaalhulpmiddel zoals **lokalize**. Eigenlijk zou u, ook als u niet van plan bent een vertaling voor te bereiden, deze stap zelf eens moeten doen. Bekijk de strings en zoek naar mogelijke problemen / dubbelzinnigheden.
- De vertaler slaat de vertaling op als **rkward__*POID*.*xx*.po** (waarin *xx* de taalcode is, Nederlands is nl), en stuurt het aan u terug.
- Kopieer **rkward__*POID*.*xx*.po** naar uw bronbestanden, naast **rkward__*POID*.pot**. Laat **python scripts/update_plugin_messages.py /pad/naar/mijn.pluginmap** werken. (Merk op: nu zonder `--extract-only`). Hierdoor wordt de vertaling samengevoegd (merge) met alle tussentijdse veranderingen in de strings, de vertaling gecompileerd, en geïnstalleerd in ***DIR_OF_PLUGINMAP*/po/*xx*/LC_MESSAGES/rkward__*POID*.mo** (waarin *xx* weer de taalcode (Nederlands: nl) is).
- U moet ook de niet-gecompileerde vertaling (bijv. **rkward__*POID*.*xx*.po**) bijsluiten in uw distributie, in de ˝po˝ subdirectory.
- Voor elke update van uw plugin, laat u **python scripts/update_plugin_messages.py /pad/naar/mijn.pluginmap** werken, zodat het .pot-bestand wordt bijgewerkt, maar ook de bestaande .po-files, en de catalogus van gecompileerde messages (berichten).

12.5 Schrijven van vertalingen van plugins

We nemen aan dat u kunt vertalen, of ergens anders hierover wilt lezen. Maar niettemin nog een paar opmerkingen over het vertalen van plugins voor RKWard:

- Plugins van RKWard waren onvertaalbaar tot aan versie 0.6.3, en werden voordien niet geschreven met i18n in het achterhoofd. Dus ontmoet u wat meer ambigue strings en andere i18n-problemen, dan in andere, gerijpte projecten. Werk hier aub. niet stilletjes omheen, maar laat ons (of de plugin maintainers) dit weten, zodat we er wat aan kunnen doen.
- Veel plugins van RKWard verwijzen naar zeer gespecialiseerde termen, zoals data handling (omgaan met gegevens) en statistieken, maar ook uit andere wetenschapsgebieden. In veel gevallen is er voor vertalen, hiervan tenminste enige kennis nodig. In sommige gevallen *is* er domweg geen goede vertaling van een technische term, en de beste optie is de term niet te vertalen, of de Engelse term er tussen haakjes achter te zetten. Probeer niet de strings 100% letterlijk te vertalen, maar geef liever een goede vertaling, ook al betekent dit dat er soms wat wordt overgeslagen, (of zelfs het overslaan van gehele message catalogs). Andere gebruikers zijn misschien wel in staat de gaten in de vertaalde technische termen aan te vullen.
- Op het moment dat dit wordt geschreven, gaat het hosten van het RKWard-project veranderen, en dit beïnvloedt ook de werkwijze van de vertaling. Lees het commentaar in de bijgaande .pot-bestanden, over hoe bij het vertalen te werk te gaan. Bij twijfel is het nooit verkeerd uw werk op te sturen naar de rkward-devel mailing list, of daar te vragen naar de laatste instructies.

Hoofdstuk 13

Auteur, licentie en versie-informatie

U heeft dus een aantal plugins geschreven, en u wilt uw werk ter beschikking stellen aan anderen. Om er zeker van te zijn dat gebruikers weten waar uw werk over gaat, onder welke voorwaarden u het uitgeeft, en bij wie ze moeten zijn bij problemen of met ideeën, moet u wat extra informatie geven *over* uw plugins. Dit kan met het **<about>**-element, of in de `.pluginmap` of in afzonderlijke plugin .xml-bestanden (in beide gevallen als een directe child van de document-tag). Wanneer dit in de `.pluginmap` wordt opgegeven, geldt het voor alle plugins. Indien **<about>** op beide plaatsen wordt opgegeven, heeft **<about>** -informatie in het plugin .xml-bestand voorrang boven die in het `.pluginmap` -bestand. U kunt ook een **<about>**-element toevoegen aan .rkh-pagina's, die niet bij een plugin behoren, indien dat nodig is.

Hier volgt een voorbeeld van een `.pluginmap`-bestand met slechts een beetje informatie. In geval van twijfel kan er meer informatie beschikbaar zijn in de naslag (reference).

```
<document
        namespace="rkward"
        id="SquaretheCircle_rkward"
>
        <about
                name="Square the Circle"
                shortinfo="Cirkel-kwadratuur met Heisenberg compensatie."
                version="0.1-3"
                releasedate="2011-09-19"
                url="http://eternalwondermaths.example.org/23/stc.html"
                license="GPL"
                category="Meetkunde"
        >
                <author
                        given="E.A."
                        family="Dölle"
                        email="doelle@eternalwondermaths.example.org"
                        role="aut"
                 />
                <author
                        given="A."
                        family="Assistant"
                        email="alterego@eternalwondermaths.example.org"
                        role="cre, ctb"
                 />
        </about>
        <dependencies>
                ...
        </dependencies>
```

```
        <components>
                ...
        </components>
        <hierarchy>
                ...
        </hierarchy>
</document>
```

Het meeste is vanzelfsprekend, dus bespreken we niet elk tag-element. Maar laten we enkele details bekijken waar wat meer uitleg bij nodig zou kunnen zijn (SquaretheCircle : ˝Kwadratuur van de cirkel˝, een van de drie grote meetkundige problemen uit de oudheid, onoplosbaar met passer en liniaal).

Het *category*-element in **<about>** kan wat ruim worden gedefinieerd, maar moet betekenisvol zijn, omdat de bedoeling is dat het wordt gebruikt om plugins in groepen in te delen. Alle andere attributen in deze openingstag zijn verplicht en moeten worden ingevuld met een redelijke inhoud.

Ook moet minstens een **<author>** met een geldig emailadres en de role 'aut' ('author') worden opgegeven. Voor het geval dat er een probleem is met uw plugin, of als iemand u wilt bedanken, moet het eenvoudig zijn even contact met u op te nemen. Voor verdere informatie over andere valide rollen, zoals 'ctb' voor mensen die code hebben bijgedragen (contribute), of 'cre' voor pakketonderhoud (creator), zie R-documentatie over person().

> OPMERKING
> Onthoud dat u **<include>** en / of **<insert>** kunt gebruiken voor het herhalen van informatie in diverse .xml-bestanden (bijv. informatie over een auteur die betrokken was bij verschillende plugins). Verdere informatie.

> TIP
> U hoeft deze XML-code niet zelf te schrijven. Als u de functie `rk.plugin.skeleton()` gebruikt uit het rkwarddev-pakket en alle nodige informatie aanlevert via de optie *about*, wordt er voor u automatisch een `.pluginmap`-bestand aangemaakt met een werkende <about>-sectie.

Hoofdstuk 14

Uw werk met anderen delen

14.1 Externe plugins

Vanaf versie 0.5.5 biedt RKWard een gemakkelijke manier voor het installeren van extra plugins van derden, die niet behoren tot het kernpakket zelf. Deze worden 'externe plugins' genoemd. Zij komen in de vorm van een R-pakket, en kunnen direct vanuit het gebruikelijke pakketbeheer van R en/of RKWard worden beheerd.

Deze sectie van de documentatie beschrijft hoe externe plugins tot een pakket moeten worden gemaakt (packaged), zo dat ze met RKWard kunnen worden gebruikt. Het maken van de plugins zelf is natuurlijk hetzelfde als in de vorige secties. U moet dus waarschijnlijk eerst een werkende plugin maken, en daarna hier zien hoe die beschikbaar te maken voor anderen (distribute it).

Omdat externe plugins relatief iets nieuws zijn, kunnen in toekomstige uitgaven de details mogelijk veranderen. Wij horen gaarne van u als u nieuwe ideeën heeft waarmee het proces kan worden verbeterd.

> TIP
> In deze docs worden de details uitgelegd van externe plugins, opdat u kunt leren hoe zij werken. Bovendien kunt u een blik werpen op het rkwarddev-pakket, dat bedoeld is om veel van het ontwikkelingsproces te automatiseren.

14.2 Waarom externe plugins?

Het aantal pakketten waarmee de functionaliteit van R wordt uitgebreid, is al immens, en wordt steeds groter. Aan de ene kant moedigen we aan plugins te maken voor zelfs de meest gespecialiseerde taken die u moet oplossen. Aan de andere kant moet de gemiddelde gebruiker niet verdwalen in een oerwoud van menu's vol met onbekende statistische termen. Daarom leek het verstandig het beheren van plugins in RKWard heel modulair te houden. Het team van RKWard onderhoudt zijn eigen publieke repository van pakketten op http://files.kde.org/rkward/R, bestemd voor het bewaren van uw externe plugins.

Een vuistregel hiervoor is dat plugins die een wijdverspreid doel dienen (bijv. t- Tests) deel zouden moeten uitmaken van het kernpakket (core package), en die alleen voor een heel beperkt aantal mensen met gespecialiseerde belangstelling nuttig zijn, moeten worden geleverd als een optioneel pakket. Voor u als een schrijver van plugins, is het het beste als u met een externe plugin begint.

14.3 Structuur van een plugin-pakket

Opdat externe plugins goed installeren en werken, is het nodig dat zij voldoen aan enkele structurele regels met betrekking tot hun bestandshiërarchie.

14.3.1 Bestandshiërarchie

Kijken we naar de prototype bestandshiërarchie van een bewerkelijk pluginarchief. U heeft niet alle genoemde directories en/of bestanden nodig om het te laten werken (lees verder voor wat werkelijk nodig is). Beschouw dit als een 'best practice'-voorbeeld:

```
plugin_name/
                inst/
                        rkward/
                                plugins/
                                        plugin_naam.xml
                                        plugin_naam.js
                                        plugin_naam.rkh
                                        ...
                                po/
                                        ll/
                                                LC_MESSAGES/
                                                        rkward__plugin_naam_rkward ↩
                                                            .mo
                                        rkward__plugin_naam_rkward.ll.po
                                        rkward__plugin_naam_rkward.pot
                                tests/
                                        testsuite_naam/
                                                RKTestStandards. ↩
                                                    eentest_naam.rkcommands. ↩
                                                    R
                                                RKTestStandards. ↩
                                                    eentest_naam.rkout
                                                ...
                                        testsuite.R
                                plugin_name.pluginmap
                                ...
        ChangeLog
        README
        AUTHORS
        LICENSE
        DESCRIPTION
```

> **OPMERKING**
> In dit voorbeeld moeten alle voorkomens van plugin_naam, testsuite_naam en eentest_naam worden vervangen door de juiste namen. Ook is ll een plaatshouder voor een taalcode (bijv., 'nl' (Nederlands), 'en' of 'es').

> **TIP**
> U hoeft deze bestandshiërarchie niet zelf te maken. Indien u de functie `rk.plugin.skeleton()` uit het rkwarddev-pakket gebruikt, maakt het automatisch alle nodige bestanden en directories voor u aan, behalve de `po`-directory, die wordt aangemaakt en beheerd door de vertaalscript.

14.3.1.1 Basis plugin-componenten

Het is verplicht ten minste drie bestanden bij te sluiten: een `.pluginmap`, een plugin .xml -beschrijving en een plugin .js-bestand. Dat betekent dat zelfs de ˝plugins˝-directory optioneel is. Het kan helpen als u uw bestandenenigszins ordent, zeker als u meerdere plugins/dialogen in het archief opneemt, wat natuurlijk geen probleem is. U kunt net zoveel directories gebruiken voor de pluginbestanden als u nodig vindt, zij hoeven slechts te lijken op de `.pluginmap`. Het is ook mogelijk diverse `.pluginmap`-bestanden bij te sluiten, indien dit nodig is, maar u moet ze in dat geval allemaal opnemen in 'plugin_naam.pluginmap'.

Elk R-pakket moet een geldig `DESCRIPTION`-bestand (bestand met een BESCHRIJVING) hebben, dat ook cruciaal is voor RKWard en waarin staat dat het een plugin bevat. De meeste informatie erin is ook nodig in de plugin Meta-informatie (informatie over de plugin) en mogelijke dependencies, maar in een ander format (de R-documentatie verklaart het DESCRIPTION-bestand in detail).

Naast de algemene inhoud van een `DESCRIPTION`- bestand, moet u ook zorgen voor de regel 'Enhances: rkward' (verbetert rkward). Hierdoor zal RKWard automatisch het pakket naar plugins doorzoeken bij installatie. Een voorbeeld van een `DESCRIPTION`-bestand ziet er als volgt uit: this:

```
Package: SquaretheCircle
        Type: Package
        Title: Square the circle
        Version: 0.1-3
        Date: 2011-09-19
        Author: E.A. Dölle <doelle@eternalwondermaths.example.org>
        Maintainer: A. Assistant <alterego@eternalwondermaths.example.org>
        Enhances: rkward
        Description: Cirkelkwadratuur met Heisenberg compensation.
        License: GPL
        LazyLoad: yes
        URL: http://eternalwondermaths.example.org/23/stc.html
        Authors@R: c(person(given="E.A.", family="Dölle", role="aut",
                        email="doelle@eternalwondermaths.example.org"),
                        person(given="A.", family="Assistant", role=c("cre ↩
                            ",
                        "ctb"), email="alterego@eternalwondermaths.example. ↩
                            org"))
```

> **TIP**
> U hoeft dit bestand niet zelf te schrijven. Indien u de functie `rk.plugin.skeleton()` uit het rkwarddev-pakket gebruikt en alle nodige informatie geeft via de 'about'-optie, dan maakt het automatisch een werkend `DESCRIPTION`-bestand aan voor u.

14.3.1.2 Bijkomende informatie (optioneel)

`ChangeLog`, `README`, `AUTHORS`, `LICENSE` moeten voor zichzelf spreken en zijn geheel optioneel. In feite worden die niet geïnterpreteerd door RKWard, dus zijn die alleen bedoeld voor het doorgeven van informatie die van belang kan zijn voor bijv. distributeurs. De meeste informatie (over de auteurs, licentiebepalingen etc.) worden toch wel gegeven in de plugin-bestanden (zie de sectie over meta-informatie). Merk op dat al deze bestanden ook kunnen worden geplaatst ergens in de ˝inst˝-directory, indien u wilt dat ze niet alleen in het bronarchief (source archive), maar ook in het geïnstalleerde pakket aanwezig zijn.

14.3.1.3 Geautomatiseerd testen van plugins (optioneel)

Een andere optionele directory is ˝tests˝, met daarin bestanden die nodig zijn voor het automatisch testen van plugins. Deze tests zijn nuttig voor het snel controleren of uw plugins nog wel werken met de nieuwe versies van R of RKWard. Indien u deze tests wilt gebruiken, dan moet u u heel goed houden aan het het hier beschreven namenschema en hiërarchie. Dit betekent dat de tests in een directory met de naam `tests` moeten zijn, waaronder een bestand `testsuite.R` en een map met teststandaarden die genoemd zijn naar de bijbehorende test-suite. U kunt echter meerdere test-suites beschikbaar stellen. In dat geval hoeft u ze niet allemaal in één `testsuite.R`-bestand op te nemen. U kunt ze opsplitsen in bijv. een bestand voor elke test-suite en een `testsuite.R` aanmaken met `source()`-aanroepen voor elk bestand in de suite. In beide gevallen moet u aparte subdirectories aanmaken met teststandaarden voor elke gedefinieerde suite. Een suite (zeg: swiet) is een aantal bij elkaar behorende programma's.

Het nut van het zich houden aan deze structuur is dat het testen van plugin eenvoudig kan worden gedaan met `rktests.makplugintests()` in het rkwardtests-pakket zonder extra argumenten. Zie ook de online-documentatie Automatisch plugin-testen voor nadere details.

14.4 Het compileren (build) van het plugin-pakket

Zoals eerder is uitgelegd, zijn externe RKWard-plugins feitelijk R-pakketten, en dus is het proces van het maken van pakketten identiek. In tegenstelling tot ˝echte˝ R-pakketten, heeft een zuiver plugin-pakket geen verdere R-code (hoewel u natuurlijk ook RKWard-plugins aan gewone R-pakketten kunt toevoegen, op dezelfde manier als hier uitgelegd). Het wordt hierdoor nog eenvoudiger een werkend pakket te maken, als u maar zorgt voor een geldig `DESCRIPTION`-bestand (BESCHRIJVING), en u u houdt aan de bestanden hiërarchie zoals uitgelegd in de vorige secties.

De gemakkelijkste manier voor het daadwerkelijk compileren (build) en testen van uw plugin, is met de opdracht R op de opdrachtregel, bijvoorbeeld:

```
R CMD build SquaretheCircle
R CMD INSTALL SquaretheCircle_0.1-3.tar.gz
```

> TIP
> U hoeft dit compileren van het pakket niet op de opdrachtregel te doen, met de functie `rk.build.package()` in het rkwarddev pakket, kunt u uw plugin-pakket compileren en/of controleren.

Hoofdstuk 15

Plugin ontwikkelen met het rkwarddev pakket.

15.1 Overzicht

Het schrijven van plugins houdt het schrijven in van bestanden in drie talen (XML, JavaScript en R) en het aanmaken van een standaard hiërarchie van directories. Om dit voor degenen die plugins willen ontwikkelen (developers), eenvoudiger te maken, leveren wij het rkwarddev-pakket. Hierin zijn een aantal eenvoudige R-functies aanwezig, voor het aanmaken van XML-code voor alle dialoog-elementen zoals tabbooks, keuzevakjes, keuzelijsten of bestands-browsers, naast functies voor het aanmaken van JavaScript-code en help-bestanden voor RKWard, om er maar een paar te noemen. De functie `rk.plugin.skeleton()` maakt een verwachte directory-boom aan en alle nodige bestanden, daar waar je die zou verwachten (skeleton is zoiets als geraamte, al het kale nodige).

Dit pakket wordt niet standaard geïnstalleerd, maar moet zelf worden geïnstalleerd uit RKWard's eigen repository. U kunt dit doen met (**Instellingen** → **Pakketten instellen**) in de GUI (Grafisch), of in een lopende R-sessie.

```
install.packages("rkwarddev", repos="http://files.kde.org/rkward/R")
                library(rkwarddev)
```

rkwarddev is afhankelijk van een klein pakket met de naam 'XiMpLe', dat een heel eenvoudig programma is voor het lezen (parser) en aanmaken van XML-bestanden, en dat ook aanwezig is in dezelfde repository.

De volledige documentatie in PDF-formaat kunt u daar ook vinden. Een gedetailleerdere inleiding in het werken met het pakket vindt u in de rkwarddev vignette.

15.2 Praktijk-voorbeeld

Om u een idee te geven van wat het 'scripten van een plugin' inhoudt, in vergelijking met de directe aanpak die u in voorgaande hoofdstukken heeft gezien, maken we opnieuw een volledige t-test plugin aan -- deze keer met alleen de R-functies van het rkwarddev pakket.

TIP
Het pakket voegt een nieuwe GUI-dialoog toe aan RKWard in het menu **Bestand** → **Exporteren** → **RKWard plugin-script aanmaken**. Uit de naam blijkt, dat u kale (skeleton) plugins kunt aanmaken, die nader kunnen worden uitgewerkt. Deze plugin werd op zijn beurt gegenereerd door een rkwarddev-script die u kunt vinden in de 'demo'-directory van het geïnstalleerde pakket en pakketbronnen (sources), als een extra voorbeeld. U kunt het ook laten werken met de aanroep (calling) `demo(´´skeleton_dialog´´)`

15.2.1 GUI beschrijving

U zult onmiddellijk merken dat de manier van werken heel anders is: In tegenstelling tot het direct schrijven van de XML-code, begint u niet met de **<document>**-definitie, maar met de plugin- elementen die u in de dialoog wilt hebben. U kunt elk interface- element toekennen -- of het nu keuzevakjes, neerklapmenu's, slots voor variabelen of wat anders zijn -- aan individuele R-objecten, en daarna deze objecten combineren met de actuele GUI. Het pakket heeft functies voor elke XML tag die kan worden gebruikt voor het definiëren van de plugin-GUI, en de meeste daarvan hebben dezelfde naam, maar dan met het voorvoegsel `rk.XML.*`. Bijvoorbeeld kan het definiëren van een **<varselector>** en twee **<varslot>**-elementen voor de variabelen *´´x´´* en *´´y´´* van het t-test voorbeeld worden gedaan met:

```
variables <- rk.XML.varselector(id.name="vars")
var.x <- rk.XML.varslot("compare", source=variables, types="number",  ↩
    required=TRUE, id.name="x")
var.y <- rk.XML.varslot("against", source=variables, types="number",  ↩
    required=TRUE, id.name="y")
```

Het interessantste hierbij is waarschijnlijk *source=variables*: een prominente eigenschap van het pakket is dat alle functies automatische ID's kunnen genereren, zodat u niet zelf *id*-waarden hoeft te bedenken, of eraan te denken dat ze naar een specifiek plugin-element moeten verwijzen. U kunt eenvoudig verwijzen naar de R-objecten, omdat alle functies die een ID nodig hebben van enig ander element die ook uit deze objecten kunnen lezen. `rk.XML.varselector()` is een beetje bijzonder, omdat die gewoonlijk geen inhoud (content) heeft waaruit een ID kan worden verkregen (het is mogelijk, maar alleen als u een naam opgeeft), dus moeten we zelf een ID-naam instellen. Maar `rk.XML.varslot()` zou hier geen *id.name* argumenten nodig hebben, en dus is dit voldoende:

```
variables <- rk.XML.varselector(id.name="vars")
var.x <- rk.XML.varslot("compare", source=variables, types="number",  ↩
    required=TRUE)
var.y <- rk.XML.varslot("against", source=variables, types="number",  ↩
    required=TRUE)
```

Om alle voorbeeldcode precies zo opnieuw aan te maken zou u alle ID-waarden handmatig moeten instellen. Maar omdat het pakket het ons gemakkelijker moet maken, maken we ons daarover geen zorgen meer.

TIP
rkwarddev kan veel automatiseren bij het compileren van uw plugins. Maar het kan de voorkeur hebben dit niet volledig te gebruiken. Als u als doel heeft niet alleen werkende code te produceren, maar ook dat die door een mens gemakkelijk kan worden gelezen en vergeleken met uw genererende script, moet u overwegen altijd zelfverklarende ID's (namen) in te stellen met *id.name*. Als u uw R-objecten dezelfde namen geeft helpt dit ook met het verkrijgen van script-code die gemakkelijk(er) te begrijpen is.

Indien u wilt zien hoe de XML-code van het gedefinieerde element eruit ziet als u het naar een bestand exporteert, moet u het object met zijn naam aanroepen (call by name). Dus, als u nu 'var.x' aanroept in uw R-sessie, moet u iets zien dat er ongeveer zo uitziet:

```
<varslot id="vrsl_vergelijk" label="vergelijken" source="vars" types=" ↩
    number" required="true" />
```

Sommige tags zijn alleen nuttig in de context van andere. Daarom, bijvoorbeeld, vindt u geen functie voor de **<option>**-tag. In plaats daarvan worden beide radioknoppen en neerklapmenu's gedefinieerd met inbegrip van hun opties als een lijst met naam (named list) waarin de namen gelijk zijn aan de de in de dialoog getoonde namen, en hun waardes een vector zijn met naam (named vector) die twee ingangen kan hebben, *val* voor de waarde van een optie, en de boolean *chk* om op te geven of deze optie standaard aan is.

```
test.hypothesis <- rk.XML.radio("met test-hypothese",
        options=list(
                "Twee-zijdig"=c(val="two.sided"),
                "Eerste is groter"=c(val="greater"),
                "Tweede is groter"=c(val="less")
        )
)
```

Het resultaat ziet er als volgt uit:

```
<radio id="rad_usngtsth" label="met test-hypothese">
        <option label="Twee-zijdig" value="two.sided" />
        <option label="Eerste is groter" value="greater" />
        <option label="Tweede is groter" value="less" />
</radio>
```

Alles wat ontbreekt voor de elementen van het tabblad 'Basisinstellingen' is het keuzevakje voor gepaarde voorbeelden, en het structureren van al deze elementen in rijen en kolommen:

```
check.paired <- rk.XML.cbox("Gepaard voorbeeld", value="1", un.value="0")
basic.settings <- rk.XML.row(variables, rk.XML.col(var.x, var.y, test. ↩
    hypothesis, check.paired))
```

`rk.XML.cbox()` is een zeldzame uitzondering, waarin de functienaam niet de volledige tag-naam bevat, om wat typewerk uit te sparenvoor dit veelgebruikte element. De `basic.settings` (basis-instellingen) bevat nu:

```
<row id="row_vTFSPP10TF">
        <varselector id="vars" />
        <column id="clm_vrsTFSPP10">
                <varslot id="vrsl_compare" label="vergelijk" source="vars"  ↩
                    types="number" required="true" />
                <varslot id="vrsl_against" label="met" i18n_context=" ↩
                    vergelijken met" source="vars" types="number" required=" ↩
                    true" />
                <radio id="rad_usngtsth" label="met test-hypothese">
                        <option label="Twee-zijdig" value="two.sided" />
                        <option label="Eerste is groter" value="greater" />
                        <option label="Tweede is groter" value="less" />
                </radio>
                <checkbox id="chc_Pardsmpl" label="Gepaard voorbeeld" value ↩
                    ="1" value_unchecked="0" />
        </column>
</row>
```

Op een soortgelijke manier maken de volgende regels R-objecten aan voor de elementen van het tabblad 'Opties', voor functies voor spinvelden, frames en stretch:

```
check.eqvar <- rk.XML.cbox("gelijke varianties aannemen", value="1", un. ↩
    value="0")
conf.level <- rk.XML.spinbox("vertrouwensniveau", min=0, max=1, initial ↩
    =0.95)
check.conf <- rk.XML.cbox("druk vertrouwensniveau af", val="1", chk=TRUE)
conf.frame <- rk.XML.frame(conf.level, check.conf, rk.XML.stretch(), label ↩
    ="Vertrouwensinterval")
```

Hierna is alles wat we nog moeten doen de objecten samen in een tabbook zetten en dat in een dialoogsectie plaatsen:

```
full.dialog <- rk.XML.dialog(
        label="t-Test met twee variabelen",
        rk.XML.tabbook(tabs=list("Basisinstellingen"=basic.settings, " ↩
            Opties"=list(check.eqvar, conf.frame)))
)
```

We kunnen ook nog de assistent-sectie aanmaken met zijn twee pagina's met dezelfde objecten, dus hun ID's worden gehaald voor de **<copy>**-tags:

```
full.wizard <- rk.XML.wizard(
                label="Twee-variabelen t-Test",
                rk.XML.page(
                        rk.XML.text("Ten eerste, selecteer de twee te  ↩
                            vergelijken variabelen. En geef op, welke  ↩
                            volgens uw theorie de grootste is. Selecteer  ↩
                            twee-zijdig,
                                als uw theorie niet uitwijst, welke  ↩
                                    variabele de grootste is."),
                        rk.XML.copy(basic.settings)),
                rk.XML.page(
                        rk.XML.text("Hieronder zijn wat gevorderde opties.  ↩
                            Het  is gewoonlijk veilig niet aan te nemen dat
                                de varianties van de variabelen gelijk zijn ↩
                                    . Een toepasselijke correctie wordt in  ↩
                                    dat geval toegepast.
                                Maar het kiezen van  \"gelijke varianties  ↩
                                    aannemen\" kan de sterkte van de test  ↩
                                    gunstig beïnvloeden."),
                        rk.XML.copy(check.eqvar),
                        rk.XML.text("Soms is het verkrijgen nuttig van een  ↩
                            schatting van het betrouwbaarheidsinterval van
                                het verschil van de gemiddelde waarden.  ↩
                                    Hieronder kunt u opgeven of er een moet  ↩
                                    worden getoond, en
                                welk vertrouwensniveau moet worden  ↩
                                    toegepast (95% komt overeen met een 5%
                                significantie-niveau)."),
                        rk.XML.copy(conf.frame)))
```

Voor zover de GUI. Het gehele document wordt tenslotte gecombineerd met `rk.plugin.skeleton()`.

15.2.2 JavaScript code

Tot dusver heeft het gebruiken van het rkwarddev-pakket naar het lijkt niet zo veel nut gehad. Dit gaat nu echt veranderen.

Ten eerste, precies zoals we niet hoefden te zorgen voor ID's voor elementen bij het maken van de indeling van de GUI, hoeven we nu niet te zorgen voor de namen van variabelen in JavaScript in de volgende stap. Indien u wat meer controle wilt, kunt u gewone JavaScript-code schrijven, en dit plakken in het gegenereerde bestand. Maar het is waarschijnlijk efficiënter het op de rkwarddev- manier te doen.

U hoeft werkelijk geen enkele variabele zelf te definiëren, omdat de `rk.plugin.skeleton()` uw XML-code kan scannen en automatisch alle variabelen kan definiëren die u mogelijk nodig heeft -- bijvoorbeeld, u hoeft niet te zorgen voor een keuzevakje als u die later toch niet nodig heeft. Dus kunnen we direct beginnen met het schrijven van de actuele R-code voor het genereren van JS:

> TIP
> De functie `rk.JS.scan()` kan ook bestaande XML-bestanden scannen, op zoek naar variabelen.

Het pakket heeft ook een aantal functies voor JS code constructs die gewoonlijk worden gebruikt in RKWard-plugins, zoals de `echo()`-functie of `if() {...} else {...}` condities. Er zijn enkele verschillen tussen JS en R, bijv., bij `paste()` in R wordt de komma gebruikt voor het aan elkaar plakken van karakter-strings, waar bij `echo()` in JS daarvoor de '+' wordt gebruikt, en regels moeten worden afgesloten met een puntkomma. Door R-functies te gebruiken, kunt u deze verschillen bijna vergeten en R-code blijven schrijven.

Deze functies hebben verschillende classes van invoerobjecten nodig: of gewone tekst, R-objecten zoals hier boven met XML-code, of op hun beurt resultaten van enige andere JS-functies in het pakket. Uiteindelijk roept u altijd `rk.paste.JS()` aan, die zich net zo gedraagt als `paste()`, maar afhankelijk van de invoerobjecten vervangt het die door hun XML-ID, variabelenaam in JavaScript of zelfs complete JavaScript code blocks.

In het t-test voorbeeld hebben we twee JS-objecten nodig: een voor de berekening van de resultaten, en een om die af te drukken met de `printout()` functie:

```
JS.calc <- rk.paste.JS(
        echo("res <- t.test (x=", var.x, ", y=", var.y, ", hypothesis=\"",  ↩
            test.hypothesis, "\""),
        js(
                if(check.paired){
                        echo(", paired=TRUE")
                },
                if(!check.paired && check.eqvar){
                        echo(", var.equal=TRUE")
                },
                if(conf.level != "0.95"){
                        echo(", conf.level=", conf.level)
                },
                linebreaks=TRUE
        ),
        echo(")\n"),
        level=2
)

JS.print <- rk.paste.JS(echo("rk.print (res)\n"), level=2)
```

Zoals u ziet geeft rkwarddev ook een R- implementatie van de `echo()` functie. Die geeft precies één karakterstring terug met een geldige JS-versie van zichzelf. U merkt misschien ook op dat

hierin alle R-objecten dezelfde zijn als die we eerder hebben aangemaakt. Die worden automatisch vervangen door hun variabelenamen, dus moet dit heel intuïtief zijn. Telkens wanneer u alleen deze vervanging wenst, kan de functie `id()` worden gebruikt, die ook precies één karakterstring teruggeeft voor alle erin ingevoerde objecten (u zou kunnen zeggen dat die zich net zo gedraagt als `paste()`, met een zeer specifieke object-substitutie.

De `js()`-functie is een wrapper (functie waarin een andere functie is ˝verpakt˝) waarmee u R's **if(){...} else {...}** condities op dezelfde manier kunt gebruiken zoals u gewend bent. Die wordt direct naar JS-code vertaald. Hierin worden ook enkele operators zoals **<**, **>=** of **||** behouden, zodat u uw R-objecten logisch met elkaar kunt vergelijken zonder de noodzaak ze steeds weer tussen aanhalingstekens te plaatsen. Bekijken we het resulterende 'JS.calc'-object, dat nu een karakterstring met inhoudt bevat:

```
echo("res <- t.test (x=" + vrslCompare + ", y=" + vrslAgainst + ",  ↩
    hypothesis=\"" + radUsngtsth + "\"");
        if(chcPardsmpl) {
                echo(", paired=TRUE");
        } else {}
        if(!chcPardsmpl && chcAssmqlvr) {
                echo(", var.equal=TRUE");
        } else {}
        if(spnCnfdnclv != "0.95") {
                echo(", conf.level=" + spnCnfdnclv);
        } else {}
        echo(")\n");
```

> OPMERKING
> U kunt ook voor `if()` condities, genest in `js()`, de functie `ite()` gebruiken, die zich net zo gedraagt als R's `ifelse()`. Echter, conditie-statements gemaakt met `ite()` zijn gewoonlijk moeilijk leesbaar, en zouden waar mogelijk door `js()` moeten worden vervangen.

15.2.3 Plugin map

Deze sectie is heel kort: we hoeven helemaal geen `.pluginmap` te schrijven, omdat die automatisch kan worden gegenereerd door `rk.plugin.skeleton()`. De menu-hiërarchie kan worden opgegeven via de *`pluginmap`*-optie:

```
[...]
        pluginmap=list(
                name="Twee-variabele t-Test",
                hierarchy=list("analyse", "gemiddelden", "t-Test"))
        [...]
```

15.2.4 Help-pagina

Ook deze sectie is heel kort: `rk.plugin.skeleton()` kan geen volledige help-pagina aanmaken met de beschikbare informatie. Maar die kan wel het XML-document scannen naar elementen die mogelijk thuishoren in een help-pagina, en automatisch een sjabloon (template) maken voor de help-pagina van onze plugin. We moeten daarna voor elk daarvan wat regels tekst toevoegen.

> TIP
> De functie `rk.rkh.scan()` kan ook bestaande XML- bestanden scannen en daarna een kaal help-bestand aanmaken.

15.2.5 De plugin-bestanden genereren

Nu komt de laatste stap, waarin we alle gegenereerde objecten overdragen aan `rk.plugin.skeleton()`:

```
plugin.dir <- rk.plugin.skeleton("t-Test",
        xml=list(
                dialog=full.dialog,
                wizard=full.wizard),
        js=list(
                results.header="Twee-variabele t-Test",
                calculate=JS.calc,
                printout=JS.print),
        pluginmap=list(
                name="Twee-variabele t-Test",
                hierarchy=list("analyse", "gemiddelden", "t-Test")),
        load=TRUE,
        edit=TRUE,
        show=TRUE)
```

Standaard worden de bestanden aangemaakt in een tijdelijke directory. De laatste drie opties zijn niet nodig, maar wel erg handig: *`load=TRUE`* plaatst automatisch de nieuwe plugin in RKWards configuratie (omdat het in een tijdelijke directory is en dus ophoudt te bestaan als RKWard wordt afgesloten, wordt het automatisch weer verwijderd door RKWard bij de volgende start), *`edit=TRUE`* opent alle aangemaakte bestanden voor bewerking in RKWard's editor tabs, en *`show=TRUE`* probeert direct de plugin te starten, zodat u die kunt bekijken zonder een enkele klik. U zou kunnen overwegen *`overwrite=TRUE`* toe te voegen, indien u uw script herhaaldelijk wilt starten (bijv. na wijzigingen in de code), omdat standaard bestanden niet worden overschreven.

Het resulterende object 'plugin.dir' bevat het pad naar de directory waarin de plugin is aangemaakt. Dit kan nuttig zijn, samen met de functie `rk.build.package()`, voor het aanmaken van een actueel R-pakket om uw plugin met anderen te delen -- bijv. door die te sturen naar het RKWard development team, zodat die kan worden toegevoegd aan uw plugin repository.

15.2.6 Het volledige script

Om al het bovenstaande samen te vatten, volgt hier het volledige script voor een werkend voorbeeld van de t-test. Behalve de reeds uitgelegde code, leest het ook het pakket in, indien nodig, en gebruikt het de `local()` omgeving, zodat niet alle aangemaakte objecten terecht komen in uw huidige workspace (behalve 'plugin.dir'). Hierin is door de mij, vertaler, NIETS gewijzigd, zodat u dit ook als het origineel kunt beschouwen van de wel vertaalde delen:

```
require(rkwarddev)

local({
        variables <- rk.XML.varselector(id.name="vars")
        var.x <- rk.XML.varslot("compare", source=variables, types="number ↩
            ", required=TRUE)
        var.y <- rk.XML.varslot("against", source=variables, types="number ↩
            ", required=TRUE)
        test.hypothesis <- rk.XML.radio("using test hypothesis",
                options=list(
                        "Two-sided"=c(val="two.sided"),
                        "First is greater"=c(val="greater"),
                        "Second is greater"=c(val="less")
                )
        )
```

```
check.paired <- rk.XML.cbox("Paired sample", value="1", un.value ↩
    ="0")
basic.settings <- rk.XML.row(variables, rk.XML.col(var.x, var.y,  ↩
    test.hypothesis, check.paired))

check.eqvar <- rk.XML.cbox("assume equal variances", value="1", un. ↩
    value="0")
conf.level <- rk.XML.spinbox("confidence level", min=0, max=1,  ↩
    initial=0.95)
check.conf <- rk.XML.cbox("print confidence interval", val="1", chk ↩
    =TRUE)
conf.frame <- rk.XML.frame(conf.level, check.conf, rk.XML.stretch() ↩
    , label="Confidence Interval")

full.dialog <- rk.XML.dialog(
        label="Two Variable t-Test",
        rk.XML.tabbook(tabs=list("Basic settings"=basic.settings, " ↩
            Options"=list(check.eqvar, conf.frame)))
)

full.wizard <- rk.XML.wizard(
                label="Two Variable t-Test",
                rk.XML.page(
                        rk.XML.text("As a first step, select the  ↩
                            two variables you want to compare  ↩
                            against
                                each other. And specify, which one  ↩
                                    you theorize to be greater.  ↩
                                    Select two-sided,
                                if your theory does not tell you,  ↩
                                    which variable is greater."),
                        rk.XML.copy(basic.settings)),
                rk.XML.page(
                        rk.XML.text("Below are some advanced  ↩
                            options. It's generally safe not to  ↩
                            assume the
                                variables have equal variances. An  ↩
                                    appropriate correction will be  ↩
                                    applied then.
                                Choosing \"assume equal variances\" ↩
                                     may increase test-strength,  ↩
                                    however."),
                        rk.XML.copy(check.eqvar),
                        rk.XML.text("Sometimes it's helpful to get  ↩
                            an estimate of the confidence interval  ↩
                            of
                                the difference in means. Below you  ↩
                                    can specify whether one should  ↩
                                    be shown, and
                                which confidence-level should be  ↩
                                    applied (95% corresponds to a 5% ↩
                                     level of
                                significance)."),
                        rk.XML.copy(conf.frame)))

JS.calc <- rk.paste.JS(
        echo("res <- t.test (x=", var.x, ", y=", var.y, ",  ↩
            hypothesis=\"", test.hypothesis, "\""),
```

```
                js(
                        if(check.paired){
                        echo(", paired=TRUE")
                        },
                        if(!check.paired && check.eqvar){
                        echo(", var.equal=TRUE")
                        },
                        if(conf.level != "0.95"){
                        echo(", conf.level=", conf.level)
                        },
                        linebreaks=TRUE
                ),
                echo(")\n"), level=2)

        JS.print <- rk.paste.JS(echo("rk.print (res)\n"), level=2)

        plugin.dir <<- rk.plugin.skeleton("t-Test",
                xml=list(
                        dialog=full.dialog,
                        wizard=full.wizard),
                js=list(
                        results.header="Two Variable t-Test",
                        calculate=JS.calc,
                        printout=JS.print),
                pluginmap=list(
                        name="Two Variable t-Test",
                        hierarchy=list("analysis", "means", "t-Test")),
                load=TRUE,
                edit=TRUE,
                show=TRUE,
                overwrite=TRUE)
})
```

15.3 Toevoegen van help-pagina's

Indien u een help-pagina voor uw plugin wilt schrijven, is de meest directe manier het toevoegen van de betreffende instructies aan de definities van de bijbehorende XML-elementen:

```
variables <- rk.XML.varselector(
  id.name="vars",
  help="Selecteer het te analyseren gegevensobject.",
  component="Data"
)
```

De tekst in de *help*-parameter kan dan worden opgehaald door `rk.rkh.scan()` en worden geschreven naar de help-pagina van deze plugin-component. Maar om dit technisch te laten werken moet `rk.rkh.scan()` weten welke R-objecten tot één plugin-component behoren. Daarom moet u ook de *component*-parameter opgeven, en er voor zorgen dat die hetzelfde is voor alle bij elkaar behorende objecten.

Omdat u gewoonlijk vele objecten in een dialoog zult willen combineren, en misschien ook objecten zult willen hergebruiken, zoals de **<varslot>** in meerdere componenten van uw plugins, is het mogelijk globaal een component te definiëren met de `rk.set.comp()`. Indien ingesteld, wordt aangenomen dat alle volgende objecten in uw script tot die component behoren, totdat `rk.set.comp()` opnieuw wordt aangeroepen met een andere componentnaam. Daarna kunt u de *component* parameter weglaten:

```
rk.set.comp("Data")
variables <- rk.XML.varselector(
  id.name="vars",
  help="Selecteer het te analyseren gegevensobject."
)
```

U kunt globale secties zoals **<summary>** (opsomming) of **<usage>** (gebruik) aan de help-pagina toevoegen met functies zoals `rk.rkh.summary()` of `rk.rkh.usage()`. De resultaten daarvan worden dan gebruikt voor het instellen van de list elements zoals *`summary`* of *`usage`* in de *`rkh`*-parameter van `rk.plugin.component()`/`rk.plugin.skeleton()`.

15.4 Plugins vertalen

Met het rkwarddev-pakket kunnen externe plugins worden gemaakt met volledige ondersteuning voor i18n. Bijvoorbeeld hebben alle relevante functies die XML-objecten genereren een optionele parameter voor het opgeven van *`i18n_context`* of *`noi18n_label`*:

```
varComment <- rk.XML.varselector(id.name="vars", i18n=list(comment="Hoofd ↩
    variabele-selector"))
varContext <- rk.XML.varselector(id.name="vars", i18n=list(context="Hoofd ↩
    variabele-selector"))
cboxNoi18n <- rk.XML.cbox(label="Macht", id.name="macht", i18n=FALSE)
```

Bovenstaande voorbeelden hebben uitvoer als volgt:

```
# varComment
<!-- i18n: Hoofd variable-selector -->
  <varselector id="vars" />

# varContext
<varselector id="vars" i18n_context="Hoofd variable-selector" />

# cboxNoi18n
<checkbox id="macht" noi18n_label="Macht" value="true" />
```

Er is ook ondersteuning voor vertaalbare JS-code. In feite probeert het pakket standaard `i18n()`-aanroepen toe te voegen op plaatsen waar dit gewoonlijk nuttig is. De functie `rk.JS.header()` is hiervan een goed voorbeeld:

```
jsHeader <- rk.JS.header("Test-resultaten")
```

Dit produceert de volgende JS-code:

```
new Header(i18n("Test-resultaten")).print();
```

Maar u kunt ook zelf strings in uw JS-code markeren als vertaalbaar, met de functie `i18n()`, precies zoals wanneer u het JS-script direct zou schrijven.

Bijlage A

Naslag

A.1 Typen van Eigenschappen/mModifiers

Op enkele plaatsen in de inleiding hebben we gesproken over 'eigenschappen' van GUI-elementen en andere. In feite zijn er meerdere verschillende typen van eigenschappen. Gewoonlijk hoeft u zich hier niet om te bekommeren, omdat we met wat gezond verstand elke eigenschap met elke andere kunnen verbinden. Maar intern zijn er verschillende typen van eigenschappen. Dit is van belang wanneer er sommige speciale waarden worden opgehaald in het JS-sjabloon (JS template). In getString (˝id˝)/getBoolean (˝id˝)/getList (˝id˝)-statements kunt u ook enkele zogenoemde 'modifiers' (veranderaars) opgeven zoals `getString (´´id.modifier´´)`. Deze modifier heeft invloed op hoe de waarde wordt afgedrukt. Lees verder voor de lijst van eigenschappen en de modifiers die elk ervan beschikbaar maakt:

String-eigenschappen

Het eenvoudigste type eigenschap, gebruikt voor het bevatten van een stuk tekst. Modifiers:

Geen modifier (˝˝)
De string zoals gedefinieerd / ingesteld.

met aanhalingstekens
De string met aanhalingstekens (geschikt voor doorgeven aan R als karakter).

Booleaanse eigenschappen

Eigenschappen die aan of uit kunnen zijn, waar of onwaar, true of false. Bijvoorbeeld de eigenschappen die worden aangemaakt door <convert>-tags (conversie, omzetten), en ook de eigenschap bij een <keuzevakje> (zie onder). De volgende waarden worden teruggegeven, volgens de gegeven modifier:

Geen modifier (˝˝)
Standaard geeft de eigenschap 1 terug als die true is (waar), en anders 0. De aanbevolen manier voor het ophalen van booleaanse waarden is met `getBoolean()`. Merk op dat bij `getString()` de string ˝0˝ wordt teruggegeven als de eigenschap false is (onwaar). De waarde van deze string is in JS true, en niet false.

˝labeled˝ (met naam)
Geeft de string ˝true˝ terug indien true, ˝false˝ indien false, of welke eigen opgegeven strings ook (typisch in een <keuzevakje>).

˝waar˝
Geef de string terug als true, ook al is de eigenschap false

˝onwaar˝
Geef de string terug als false, ook al is de eigenschap true

˝niet˝
Dit geeft een andere booleaanse eigenschap terug, die het omgekeerde is van de huidige (bijv. false indien true, true indien false)

˝numeriek˝
Verouderd, dient voor achterwaartse compatibiliteit (backwards compatibility) Zelfde als de modifier ˝˝. Geeft ˝1˝ terug als de eigenschap true is, of ˝0˝ als die false is.

Integer eigenschappen (integers zijn gehele getallen)

Een eigenschap die een geheel getal moet bevatten (maar natuurlijk toch een numerieke karakterstring teruggeeft aan de JS-sjabloon (JS template)). Er zijn hier geen modifiers voor. Wordt in <spinveld>en gebruikt (zie onder)

Eigenschappen voor real numbers (kommagetallen)

Een eigenschap die een kommagetal moet bevatten (maar natuurlijk toch een numerieke karakterstring teruggeeft aan de JS-sjabloon (JS template)). Wordt in <spinveld>en gebruikt (zie onder)

Geen modifier (˝˝)
Voor `getValue() / getString()`, geeft dit hetzelfde terug als ˝formatted˝. In volgende versies zal het mogelijk zijn in plaats hiervan een numerieke weergave te verkrijgen.

˝formatted˝
Geeft het getal geformatteerd terug (als een string).

RObject eigenschappen

Een eigenschap, bedoeld als een selectie van een of meer R-objecten. Het meest gebruikt in varselectors en varslots. De volgende waarden worden teruggegeven volgens de opgegeven modifier:

Geen modifier (˝˝)
Standaard geeft de eigenschap de volledige naam terug van het geselecteerde object. Indien meerdere objecten zijn geselecteerd, worden de objectnamen gescheiden door einde regels (˝\˝).

˝shortname˝ (korte naam)
Net als hierboven, maar nu worden alleen korte namen van de objecten teruggegeven. Bijvoorbeeld, een object in een lijst krijgt alleen de naam die het in de lijst heeft, zonder de naam van de lijst.

˝label˝ (naam)
Zoals hierboven, maar nu worden de RKWard-label(s) (namen) van object(en) teruggegeven. (als er geen label beschikbaar is, is die gelijk aan de shortname).

Eigenschappen van string lists

Deze eigenschap bevat een list van strings (lijst met karakterrijen).

Geen modifier (˝˝)
`getValue()/getString()` geeft alle strings terug gescheiden door ˝\n˝. Alle ˝\n˝ karakters in elke item worden ge-escaped met ˝\n˝. Echter, aanbevolen wordt de waarde op te halen met `getList()`, die een array van strings teruggeeft.

˝joined˝
Geeft een lijst terug als een enkele string, waarin de items verbonden (joined) zijn door ˝\n˝. In tegenstelling tot geen modifier (˝˝), worden de individuele strings _not_ escaped (niet_escaped).

Code-eigenschappen (code is programmacode)

Een eigenschap van plugins die code hebben gegenereerd. Dit is van belang voor het inbedden (embedding) van plugins, om de code, gegenereerd door de ingebedde plugin, in te bedden in de code gegenereerd door de inbeddende (bovenliggende niveau) plugin. De volgende waarden worden teruggegeven volgens de opgegeven modifier:

Geen modifier (″″)
Geeft de volledige code terug, bijv. de secties ″preprocess″, ″calculate″ (berekenen), ″printout″ (afdrukken), en samengevoegd tot een string (maar niet ″preview″ (voorbeeld)).

″preprocess″ (onvertaalbaar, want vaste programmeerterm, maar zoiets als voorbewerking)
Geeft alleen de preprocess-sectie terug van de code.

″calculate″
Geeft alleen de berekenen-sectie terug van de code

″printout″
Geeft alleen de afdruk-sectie terug van de code

″preview″ (voorbeeld, voorvertoning)
Geeft alleen de voorbeeld-sectie terug van de code

A.2 Elementen voor algemeen gebruik (general purpose) in elk XML-bestand (`.xml`, `.rkh`, `.pluginmap`)

<snippets> (brokstukjes, snippertjes)

Toegestaan als een directe child van de <document>-node en daar alleen. Moet bovenaan in het bestand worden geplaatst. Zie sectie over snippets. Er mag slechts één <snippets>-element aanwezig zijn. Optioneel, geen attributen.

<snippet>

Definieert een enkele snippet. Alleen toegestaan als een directe child van het <snippets/>-element. Attributen:

<id>
Een identificerende string voor de snippet. Vereist.

<insert> (invoegen)

Voegt de inhoud in van een <snippet>. Overal toegestaan. Attributen:

<snippet>
De in te voegen identificerende string van de snippet. Vereist.

<include> (opnemen, bijsluiten, inclusief maken)

Voegt de inhoud toe van een ander XML-bestand (alles in het <document>-element van dat bestand). Overal toegestaan. Attributen:

<file> (bestand)
De bestandsnaam, relatief tot de directory waarin het huidige bestand zich bevindt. Vereist.

A.3 Te gebruiken elementen in de XML-beschrijving van de plugin

Eigenschappen van de elementen staan worden genoemd in een aparte sectie.

A.3.1 Algemene elementen

<document>

Moet aanwezig zijn in elk beschrijvend .xml-bestand als de root-node. Geen speciale functie. Geen attributen

<about> (over)

Informatie over deze plugin (auteur, licentie, etc.) Dit element is toegestaan in zowel een afzonderlijk .xml-bestand van een plugin, als in `.pluginmap`-bestanden. Zie de naslag `.pluginmap`-bestanden voor details, het hoofdstuk over 'about'-informatie voor een inleiding.

<code> (programmacode)

Definieert waar een JS-sjabloon (template) voor de plugin kan worden gevonden. Gebruik dit slechts een keer per bestand, als een directe child van de document-tag. Attributen:

bestand
Bestandsnaam van het JS-sjabloon, relatief tot de directory waarin de plugin-xml is

<help>

Definieert waar het help-bestand van de plugin kan worden gevonden. Gebruik dit slechts een keer per bestand, als een directe child van de document-tag. Attributen:

bestand
Bestandsnaam van het help-bestand, relatief tot de directory waarin de plugin-xml is

<copy> (kopieer)

Kan worden gebruikt als een child (direct of indirect) van de elementen in de hoofdindeling, bijv. <dialog> en <wizard>. Dit wordt gebruikt om een geheel blok van 1:1-attributen van een xml-element te kopiëren. Attributen:

id
De te zoeken ID. De <copy> tag zoekt naar een eerder XML-element dat dezelfde ID heeft gekregen, en kopieert dit met alle daarvan afgeleide elementen.

copy_element_tag_name
In enkele gevallen wilt u een bijna letterlijke kopie hebben, maar de tag-naam van het gekopieerde element wijzigen. Het belangrijkste voorbeeld hiervan is wanneer u een hele <tab> wilt kopiëren uit een dialoog-interface naar de <pagina> van een assistent-interface. In dat geval stelt u copy_element_tag_name=˝page˝ in om deze conversie automatisch te doen.

A.3.2 Interface-definities

<dialog>

Definieert een dialoog-type interface. Plaats de GUI-definitie in deze tag. Gebruik dit slechts een keer per bestand, als een directe child van de document-tag. Ten minste een van de ˝dialog˝ (dialoog) of ˝wizard˝ (assistent) -tags is vereist voor een plugin. Attributen:

label
Koptekst van de dialoog

aanbevolen
Moet de dialoog gebruikt worden als de ˝aanbevolen˝ interface (bijv. de interface die standaard wordt getoond, tenzij de gebruiker in RKWard een specifieke standaard interface heeft ingesteld)? Dit attribuut heeft op dit moment geen effect, omdat het impliciet ˝true˝ is, tenzij de assistent (wizard) aanbevolen is.

<wizard> (assistent)

Definieert een assistent-achtige interface. Plaats de GUI-definitie in deze tag. Gebruik dit slechts een keer per bestand, als een directe child van de document-tag. Ten minste een van de ˝dialog˝ of ˝wizard˝ -tags is vereist voor een plugin. Accepteert alleen <page> of <embed>-tags als directe children. Attributen:

label
Koptekst voor de assistent

aanbevolen
Moet de dialoog gebruikt worden als de ˝aanbevolen˝ interface (bijv. de interface die standaard wordt getoond, tenzij de gebruiker in RKWard een specifieke standaard interface heeft ingesteld)? Optioneel, standaard is ˝false˝.

A.3.3 Elementen voor de indeling (layout)

Alle elementen in deze sectie accepteren een attribuut id=˝naamstring˝.Dit attribuut is optioneel voor alle elementen. Het kan bijvoorbeeld worden gebruikt om het hele indelingselement te tonen/verbergen, samen met alle elementen die erin zijn (zie hoofdstuk GUI-logica). De id-naamstring mag geen ˝.˝ (punt) of ˝;˝ (puntkomma) bevatten, en moet algemeen gesproken worden beperkt tot alfanumerieke karakters en de lage streep (˝_˝). Alleen de extra attributen worden genoemd:

<page> (pagina)

Definieert een nieuwe pagina in een assistent. Alleen toegestaan als directe child van een <wizard>-element.

<row> (rij)

Alle directe children van een ˝row˝-tag worden van links naar rechts geplaatst.

<column>

Alle directe children van een ˝column˝-tag worden van boven naar beneden geplaatst.

<stretch> (uitrekken)

Standaard nemen elementen van een GUI alle beschikbare ruimte in beslag. Bijvoorbeeld, als u twee kolommen naast elkaar heeft, de linkse volgepakt met elementen, en de rechtse bevat alleen maar een eenzame **<radio>**, zal de **<radio>** verticaal uit worden gerekt, ook al heeft die niet werkelijk al die ruimte nodig, en ziet het er niet fraai uit. In dit geval heeft u eigenlijk lege ruimte nodig onder de **<radio>**. Dan gebruikt u het <stretch>-element. Dat zal alleen maar wat ruimte in beslag nemen. Gebruik dit element niet al teveel, Meestal is het goed dat elementen alle beschikbare ruimte krijgen, slechts een enkele keer niet. Voor het <stretch>-element zijn geen argumenten nodig, zelfs geen ïd˝ (naam). U kunt ook geen children plaatsen in het <stretch> element (met andere woorden: u gebruikt het alleen maar als ˝<stretch>˝.)

<frame> (onvertaalbaar, zoiets als raamwerk)

Tekent een lijst/vakje rondom zijn directe children. Kan worden gebruikt voor het visueel in groepen plaatsen van bij elkaar behorende opties. De indeling binnen een frame is van boven naar beneden, tenzij u er een <row> in plaatst. Attributen:

label
Koptekst voor de frame (optioneel)

activeerbaar
Frames kunnen activeerbaar (checkable) worden gemaakt. In dat geval worden alle elementen erin uitgeschakeld, wanneer de frame niet wordt geactiveerd, en aangezet wanneer die wel wordt geactiveerd. (optioneel, standaard is ˝false˝)

geactiveerd
: Alleen voor activeerbare frames. Moet een frame standaard worden geactiveerd? Standaard is ˝true˝. Niet geïnterpreteerd voor niet-activeerbare frames.

<tabbook> (aantal tabbladen)
: Organiseert elementen in een tabbook. Accepteert alleen <tab>-tags als directe children.

<tab> (tabblad)
: Definieert een pagina in een tabbook. Plaats de GUI-definitie voor de tab in deze tag. Kan alleen worden gebruikt als een directe child van een <tabbook>-tag. Een <tabbook> moet minstens twee gedefinieerde tabs hebben. Attributen:

label
: Koptekst voor de tab-pagina (vereist)

<text> (tekst)
: Toont de tekst in deze tag in het GUI-element. Wat eenvoudige HTML-markeringen worden ondersteund (zoals <b>, <i>, <p>, en
). Maar beperk u hierbij tot een minimum. Het invoegen van een lege regel resulteert in een einde-regel (line break). Attributen:

type
: Type van de tekst. Een van ˝normal˝, ˝warning˝ (waarschuwing) of ˝error˝ (fout). Dit heeft invloed op hoe de tekst eruit ziet (optioneel, normal is standaard)

A.3.4 Actieve elementen

Alle elementen in deze sectie accepteren een attribuut id=˝naamstring˝. Dit attribuut is vereist voor alle elementen. Alleen de extra attributen worden genoemd. De id-string mag geen ˝.˝ (punten) bevatten.

<varselector> (variabelekiezer)
: Geeft een lijst van beschikbare objecten waaruit de gebruiker een of meer van kan kiezen. Heeft alleen nut als er een of meer <varslot>s zijn. Attributen:

label
: Naam van de varselector (optioneel, standaard is ˝Select variable(s)˝)

<varslot>
: Wordt samen gebruikt met een ˝varselector˝ om het mogelijk te maken een of meer variabelen te selecteren. Attributen:

label
: Naam van de varslot (aangeraden, standaard is ˝Variable:˝)

source
: De bron waaruit de varslot de selectie moet ophalen (vereist, behalve als u zelf de verbinding maakt of met behulp van source_property (bron_eigenschap))

source_property (bron_eigenschap)
: Een willekeurige bron_eigenschap waaruit waarden moeten worden gekopieerd wanneer er op de selecteer-knop wordt geklikt. Indien opgegeven wordt het ˝source˝-attribuut genegeerd.

vereist
: Of - voor het indienen van de code - het vereist is dat deze varslot een geldige waarde bevat. Zie See vereist-eigenschap (optioneel, standaard is false)

multi
: Of de varslot slechts een (standaard, ˝false˝) of meerdere objecten bevat

allow_duplicates (duplicaten_ toestaan)
: Of de varslot alleen unieke objecten (standaard, ˝false˝) mag bevatten, of dat hetzelfde object er meerdere keren er aan kan worden toegevoegd.

min_vars
: Alleen zinvol als multi=˝true˝. Minimum aantal variabelen dat gekozen moet worden voor een geldige selectie (optioneel, standaard is ˝1˝)

min_vars_if_any (minimum_aantal_variabelen_indien_aanwezig)
: Alleen zinvol als multi=˝true˝: Sommige varslots kunnen als geldig worden beschouwd, als bijvoorbeeld de varslot of leeg is, of tenminste twee waarden bevat. Hiermee wordt opgegeven hoeveel variabelen er moeten worden geselecteerd, als er sowieso variabelen worden geselecteerd (2 in het voorbeeld). (optioneel, standaard is ˝1˝)

max_vars
: Alleen zinvol als multi=˝true˝. Maximum aantal variabelen dat gekozen kan worden (optioneel, standaard is ˝0˝, wat betekent geen maximum)

classes (klassen, maar onvertaalbaar, want term uit de programmeertaal)
: Indien u een of meer R-class-namen opgeeft (gescheiden door spaties (˝ ˝), accepteert de varslot alleen variabelen die tot die classes behoren (optioneel, *gebruik dit heel voorzichtig)*. De gebruiker moet niet beperkt worden in het maken van geldige keuzes, en R heeft *heel veel* verschillende classes!)

typen
: Indien u een of meer variabelen-typen opgeeft (gescheiden door spaties (˝ ˝), accepteert de varslot alleen objecten met deze types. Geldige typen zijn ˝unknown˝ (onbekend), ˝number˝ (getal), ˝string˝, ˝factor˝, ˝invalid˝ (ongeldig). Optioneel, *gebruik dit heel voorzichtig)*. De gebruiker moet niet beperkt worden in het maken van geldige keuzes, en RKWard kent niet altijd het type van een variabele)

num_dimensions (aantal_dimensies)
: Het aantal dimensies dat een object moet hebben. ˝0˝ (standaard) betekent dat elk aantal dimensies wordt geaccepteerd. (optioneel, standaard is ˝0˝)

min_length
: De kleinste lengte die een object mag hebben. (optioneel, standaard is ˝0˝)

max_length
: De grootste lengte die een object kan hebben. (optioneel, standaard is die het grootste gehele getal dat in het systeemgeheugen kan worden opgeslagen)

<valueselector> (waarde selector)

Geeft een lijst van beschikbare strings (geen R-objecten) die kunnen worden geselecteerd in een of meer bijgaande <valueslot>s.String-opties kunnen worden gedefinieerd met <option>-tags als directe children (zie onder), of door instellen van dynamische eigenschappen. Attributen:

label
: Naam van de varselector (optioneel, standaard is geen naam (label))

<valueslot>

Gebruikt samen met een <valueselector> zodat de gebruiker een of meer string items kan selecteren. Dit element is voor een groot deel identiek aan <varslot>, en deelt dezelfde attributen, behalve die die naar eigenschappen verwijzen van de toegestane items (bijv. classes, types, num_dimensions, min_length, max_length).

<radio> (radioknoppen, selectierondjes)

Definieert een groep van knoppen voor radio alleen (slechts een kan op enig moment geselecteerd zijn). Vereist minstens twee <option>-tags als directe children. Andere tags zijn niet toegestaan als children. Attributen:

label
: Naam van de radioknoppen (aanbevolen, standaard is ˝selecteer een:˝)

<dropdown> (neerklapmenu)

Definieert een groep opties waarvan er een en slechts een tegelijk kan worden geselecteerd, in een keuzelijst (dropdown list). Dit is functioneel gelijk aan een <radio>, maar ziet er anders uit. Vereist minstens twee <option>-tags als directe children. Geen andere tags zijn toegestaan als children. Attributen:

label
Naam van de keuzelijst (aanbevolen, standaard is ˝Selecteer een:˝)

<select> (kies, selecteer)

Geeft een lijst van beschikbare strings waaruit de gebruiker een of meer kan kiezen. String-opties kunnen worden gedefinieerd met <option>-tags als directe children (zie onder) of door dynamische eigenschappen in te stellen. Attributen:

label
Naam voor de <select> (optioneel, standaard is geen naam)

<option> (optie)

Kan alleen worden gebruikt als een directe child van een <radio>, <dropdown>, <valueselector> of <select>-element. Omdat <option>-elementen altijd deel uitmaken van een van de selectie-elementen, hebben zij gewoonlijk zelf geen ˝id˝ (naam), maar zie onder. Attributen:

label
Naam voor de optie (vereist)

waarde
De stringwaarde die het parent-element teruggeeft indien deze optie wordt gekozen/geselecteerd (vereist)

geactiveerd
Of de optie standaard ˝true˝ of ˝false˝ moet worden gekozen/geselecteerd. In een <radio> of <dropdown>, kan slechts een optie worden ingesteld op`checked=´´true´´`, en indien er geen optie is ingesteld, wordt het eerste element automatisch gekozen/ geselecteerd. In een <select>, kan elk aantal opties worden ingesteld op gekozen. (optioneel, standaard is ˝false˝)

id
Opgeven van de ˝id˝-parameter (naam) voor de <option>-elementen is optioneel (en feitelijk wordt het instellen van een ˝id˝ afgeraden, tenzij u er werkelijk een nodig heeft). Maar het opgeven van een ˝id˝ maakt het wel mogelijk <option>s dynamisch aan/uit te zetten, door te verbinden met de booleaanse eigenschap `id_of_radio.id_of_optionX.enabled`. Op dit moment werkt dit alleen voor opties in <radio> of <dropdown>-elementen; <valueselector> en <select>-opties ondersteunen op dit moment geen ids.

<checkbox> (keuzevakje)

Definieert een keuzevakje. bijv. een enkele optie die alleen kan worden ingesteld op aan of uit. Attributen:

label
Naam voor het keuzevakje (vereist)

waarde
De waarde die door het keuzevakje wordt teruggegeven indien gekozen (vereist)

value_unchecked
De waarde die door het keuzevakje wordt teruggegeven indien niet gekozen (optioneel, standaard is ˝˝, bijv. een lege string)

geactiveerd
Of de optie standaard ˝true˝ moet zijn of ˝false˝ (optioneel, standaard is ˝false˝)

<frame> (onvertaalbaar, zoiets als raamwerk)

Het frame-element wordt gewoonlijk zuiver gebruikt als een layout (indelings) element, en wordt genoemd in de sectie over indelings elementen. Echter, het kan ook kiesbaar worden gemaakt, en dus tegelijkertijd optreden als een eenvoudig keuze-element.

<input> (invoer)

Definieert een vrij veld voor tekstinvoer. Attributen:

label
Naam voor het invoerveld (vereist)

aanvankelijk
Aanvangstekst voor het tekstveld (optioneel, standaard is ˝˝, bijv. een lege string)

grootte
Te kiezen uit ˝small˝ (klein), ˝medium˝ (middelgroot), of ˝large˝ (groot). ˝large˝ definieert een invoerveld met meerdere regels. ˝small˝ en ˝medium˝ velden met maar een regel (optioneel, standaard is ˝medium˝)

vereist
Of het - voor het indienen van de code - vereist is dat deze invoer niet leeg is. Zie vereist-eigenschap (optioneel, standaard is false)

<matrix>

Een tabel voor het invoeren van matrix-gegevens (of vectoren) in de GUI.

> OPMERKING
> Dit invoer-element is *niet* geoptimaliseerd voor het invoeren/ bewerken van grote hoeveelheden gegevens. Hoewel er geen strikte grens is aan de grootte van een <matrix>, moet die in het algemeen niet groter zijn dan tien rijen/kolommen. Als u meer gegevens verwacht, kunt u het mogelijk maken die als een R-object te selecteren (wat een goed idee kan zijn als alternatieve optie, in bijna *alle* gevallen dat een matrix-element nodig is).

Attributen:

label
Naam van de tabel (vereist)

modus
Een van ˝integer˝ (geheel getal), ˝real˝ (kommagetal) of ˝string˝ (rij karakters tussen aanhalingstekens). Het type van gegevens die in de tabel worden geaccepteerd (vereist)

min
Kleinste acceptabele waarde (voor matrices van het type ˝integer˝ of ˝real˝) (optioneel, standaard is kleinste getal dat kan woorden weergegeven)

max
Grootste acceptabele waarde (voor matrices van het type ˝integer˝ of ˝real˝) (optioneel, standaard is grootste getal dat kan woorden weergegeven)

ontbrekende waarden toestaan
Of ontbrekende (lege) waarden in de matrix worden toegestaan. Dit is bedoeld voor matrices met modus ˝string˝ (Optioneel, standaard is false).

allow_user_resize_columns
Indien ingesteld op true, kan de gebruiker kolommen toevoegen door in de meest rechtse (inactieve) cellen te typen (optioneel, standaard is true).

allow_user_resize_rows
Indien ingesteld op true, kan de gebruiker rijen toevoegen door in de onderste (inactieve) cellen te typen (optioneel, standaard is true).

rows
Aantal rijen in de matrix. Heeft geen effect op allow_user_resize_rows=˝true˝.

> OPMERKING
> Dit kan ook worden geregeld door instellen van de ˝rows˝-eigenschap.

(optioneel, standaard is 2).

columns
Aantal kolommen in de matrix. Heeft geen effect op allow_user_resize_columns=˝true˝.

> OPMERKING
> Dit kan ook worden geregeld door instellen van de ˝columns˝-eigenschap.

(optioneel, standaard is 2).

min_rows
Kleinste aantal rijen in de matrix. De matrix kan niet tot onder deze waarde worden ingekrompen. (optioneel, standaard is 0: zie ook: `allow_missings`.).

min_columns
Kleinste aantal kolommen in de matrix. De matrix kan niet tot onder deze waarde worden ingekrompen. (optioneel, standaard is 0: zie ook: `allow_missings`.).

fixed_height (vaste hoogte)
Zodat het GUI-element zijn originele hoogte behoudt. Gebruik dit niet samen met matrices, waarin het aantal rijen op enige manier kan veranderen. Nuttig, speciaal bij het aanmaken van een vector invoer-element (columns = ˝1˝) Als deze optie ingesteld is op true, is er geen horizontale schuifbalk, zelfs als de matrix te breed is (omdat dit de hoogte zou beïnvloeden). (optioneel, standaard is false).

fixed_width (vaste breedte)
Een beetje misplaatste naam. Neem aan dat het aantal kolommen niet verandert. De laatste (of enige) kolom wordt uitgerekt om alle beschikbare ruimte op te vullen. Gebruik dit niet samen met matrices, waarin het aantal kolommen op enige manier kan veranderen. Nuttig, vooral bij het aanmaken van een vector invoer-element (rijen is ˝1˝), (optioneel, standaard is false).

horiz_headers
Te gebruiken strings voor de horizontale koptekst, gescheiden door ˝;˝. Geen koptekst wordt getoond, indien ingesteld op ˝˝. (optioneel, standaard is kolomnummer).

vert_headers
Te gebruiken strings voor de verticale koptekst, gescheiden door ˝;˝. Geen koptekst wordt getoond, indien ingesteld op ˝˝. (optioneel, standaard is kolomnummer).

<optionset> (verzameling van opties)

Een UI voor het herhalen van een aantal opties voor een willekeurig aantal items (introductie voor optionsets). Attributen:

min_rows
Indien opgegeven, wordt de set gemarkeerd als ongeldig, tenzij die minstens dit aantal rijen heeft (optioneel, integer).

min_rows_if_any
Net als voor min_rows, maar wordt alleen getest al er minstens een rij is (optioneel, integer)

max_rows
Indien opgegeven, wordt de set gemarkeerd als ongeldig, tenzij die hoogstens dit aantal rijen heeft (optioneel, integer).

keycolumn (sleutelkolom)
Id (naam) van de kolom die als sleutelkolom wordt gebruikt. Een optionset met een (geldige) keycolumn acteert als een ˝driven˝ (aangedreven) optionset. Een optionset zonder keycolumn staat het zelf invoegen/verwijderen toe van items. De keycolumn moet als extern worden gemarkeerd. (optioneel, standaard is geen keycolumn).

Child-elements: (kind-element: van een parent afgeleid element, child en parent zijn onvertaalbare termen uit de programmeertaal)

<optioncolumn> (optiekolom)

Declareert één option column van de set. Voor elke waarde die u uit de optionset wilt ophalen, moet u een aparte <optioncolumn> declareren. Attributen:

id
De id (naam) van de optioncolumn (vereist, string).

extern
Ingesteld op true, indien de optioncolumn van buiten de optionset wordt bestuurd (optioneel, boolean, standaard is false).

label
Indien gegeven, wordt de optionset getoond in een kolom met die naam (optioneel, string, standaard is niet getoond).

verbinden
De eigenschap waarmee de optionset moet worden verbonden, gegeven als id (naam) in het <content>-gebied. Voor externe <optioncolumn>s wordt de overeenkomende waarde ingesteld op de extern ingestelde waarde. Voor reguliere (niet-externe) <optioncolumn>s wordt de overeenkomende rij van de <optioncolumn>-eigenschap ingesteld, wanneer de eigenschap verandert binnen het content-gebied. (optioneel, string, standaard is niet verbonden).

standaard
Alleen voor externe kolommen: de aan te nemen waarde voor deze kolom, indien geen waarde bekend is voor een ingang, Zelden nuttig. (optioneel, standaard is lege string)

<content> (inhoud)

Declareer de inhoud / UI van de set. Geen attributen. Alle gebruikelijke actieve, passieve, en layout-elementen zijn toegestaan als childname-elementen. Bovendien, in eerdere versies van RKWard (tot aan 0.6.3) was het speciale child-element **<optiondisplay>** toegestaan. Dit is verouderd in RKWard 0.6.4, en moet domweg worden verwijderd uit bestaande plugins.

<logic> (logica)

Optionele specificatie van de toe te passen UI-logica *in* het contents-gebied van de optionset. Zie de naslag over <logic>

<browser> (bladerprogramma)

Een element dat is ontworpen voor het selecteren van een enkele bestandsnaam (of naam van een directory). Merk op dat dit veld elke string accepteert, zelfs als die alleen in bestanden zou moeten worden gebruikt:

label
Naam van de browser (optioneel, standaard is ˝Enter filename˝ (voer bestandsnaam in))

aanvankelijk
Aanvangstekst van de browser (optioneel, standaard is ˝˝, bijv. een lege string)

type
Een van ˝file˝ (bestand), ˝dir˝, of ˝savefile˝ (opslaan bestand). Voor het selecteren van een bestaand bestand, bestaande directory, of niet-bestaand bestand. (optioneel, standaard is ˝file˝)

allow_urls
Of (niet-lokale) urls kunnen worden geselecteerd (optioneel, standaard is ˝false˝)

filter
Bestandstype filter, bijv.: (˝*.txt *.csv˝ voor .txt en .csv -bestanden) Een aparte ingang voor ˝Alle bestanden˝ wordt automatisch toegevoegd Optioneel, standaard is ˝˝, bijv. Alle bestanden.)

vereist
Of het - bij het indienen van de code- vereist is dat het veld niet leeg is. Merk op dat dit niet noodzakelijk betekent dat de geselecteerde bestandsnaam geldig is. Zie vereist-eigenschap (optioneel, standaard is true)

<saveobject> (object opslaan)

Een element ontworpen voor het selecteren van de naam van een R-object waarnaar het moet worden opgeslagen (bijv. in het algemeen niet reeds bestaand, in tegenstelling tot een varslot():

label
Naam voor de invoer (optioneel, standaard is ˝Save to:˝ (opslaan op:))

aanvankelijk
Begintekst voor de invoer (optioneel, standaard is ˝mijn.data˝ (mijn gegevens))

vereist
Of - bij het indienen van de code - het vereist is dat het veld een toegestane objectnaam bevat. Zie vereist-eigenschap (optioneel, standaard is true)

activeerbaar
In veel gevallen is het opslaan naar een R-object optioneel. In deze gevallen kan met dit attribuut een keuzevakje worden ingebouwd in het saveobject-element. Indien ingesteld op true, wordt de saveobject aan/uit gezet door het keuzevakje. Zie de active-eigenschap van saveobject (optioneel, standaard is false)

geactiveerd
Alleen voor kiesbare saveobject-elementen: of dit standaard gekozen/aan is (optioneel, standaard is false)

<spinbox> (spinveld)

Een spinveld waarin de gebruiker een numerieke waarde kan kiezen, of met directe toetsenbord-invoer, of met op/neer pijltjes. Attribuut:

label
Naam voor het spinveld (aanbevolen, standaard is ˝Enter value:˝ (waarde invoeren))

min
De laagste waarde die in het spinveld mag worden ingevoerd (optioneel, standaard is de laagste technisch mogelijke waarde in het spinveld)

max
De hoogste waarde die in het spinveld mag worden ingevoerd (optioneel, standaard is de hoogste technisch mogelijke waarde in het spinveld)

aanvankelijk
De beginwaarde getoond in het spinveld (optioneel, standaard is ˝0˝)

type
Een van ˝real˝ (kommagetal) of ˝integer˝ (geheel getal). Of in het spinveld kommagetallen of gehele getallen moeten worden gekozen. (optioneel, standaard is ˝real˝)

default_precision (standaard nauwkeurigheid)
Alleen zinvol als het type van het spinveld ˝real˝ is. Geeft het standaard aantal decimalen op in het spinveld (alleen dit aantal laatste nullen wordt getoond). Wanneer de gebruiker de op/neer pijltjes gebruikt, wordt deze decimale plaats veranderd. Maar de gebruiker kan toch waarden invoeren met een grotere nauwkeurigheid (zie onder) (optioneel, standaard is ˝2˝)

max_precision (maximale nauwkeurigheid)
Het grootste aantal cijfers dat zinvol kan worden weergegeven (optioneel, standaard is ˝8˝)

<formula> (formule)

In dit geavanceerde element kan de gebruiker een formule/aantal interacties kiezen uit geselecteerde variabelen. Bijvoorbeeld voor een GLM (Vert.: ik vermoed dat bedoeld wordt: een Generalized Linear Model: een gegeneraliseerd lineair model, maar verder doet dit er niet toe) kan dit element worden gebruikt voor het opgeven van de interactie-termen in het model. Attributen:

fixed_factors
De ID van de varslot die de geselecteerde fixed factors (vaste factoren) bevat (vereist)

afhankelijk
De ID van de varslot dat de afhankelijke variabele bevat (vereist)

<embed> (inbedden)

Bed een andere plugin in in deze plugin (zie hoofdstuk over inbedden). Attributen:

component
De geregistreerde naam van de in te bedden component (zie hoofdstuk over inbedden). Attributen:

as_button (als knop)
Indien ingesteld op ˝true˝, wordt alleen een drukknop geplaatst in de inbeddende GUI, de ingebedde GUI wordt alleen getoond (in een afzonderlijk venster) wanneer op de drukknop wordt gedrukt (optioneel, standaard is ˝false˝)

label
Alleen zinvol als as_button = ˝true˝: De naam van de knop (aanbevolen, standaard is ˝Opties˝)

<preview> (voorbeeld, voorweergave)

Keuzevakje voor het aan/uitzetten van de voorbeeldfunctionaliteit. Merk op dat vanaf versie 0.6.5 van RKWard **<preview>** preview-elementen speciaal zijn in plugin-dialogen (niet assistenten): Zij worden geplaatst in de kolom voor knoppen, ongeacht van waar ze in de UI zijn gedefinieerd. Maar het is nog steeds een goed idee ze op een zinvolle plaats in de layout te definiëren, voor achterwaartse compatibiliteit (backwards compatibility). Attributen:

label
Naam van het vakje (optioneel, standaard is ˝Preview˝)

modus
Type van preview. Ondersteund worden ˝plot˝ (zie hoofdstuk over grafiek previews), ˝output˝ (zie hoofdstuk over (HTML) uitvoer previews), ˝data˝ (zie gegevens previews), en ˝custom˝ (zie aangepaste previews). (optioneel, standaard is ˝plot˝)

plaatsing
Plaatsing van de preview: ˝attached˝ (vastgemaakt aan de hoofdwerkplaats), ˝detached˝ (los venster), ˝docked˝ (vastgemaakt aan plugin-dialoog) en ˝default˝ (dit is nu nog hetzelfde als ˝docked˝, maar kan in de toekomst mogelijk worden ingesteld door de gebruiker). In het algemeen wordt aanbevolen dit de standaard instelling te laten voor de beste UI-consistentie (optioneel, standaard is ˝default˝)

actief
Of de preview standaard actief is. In het algemeen worden alleen docked previews standaard actief gemaakt, en zelfs dan is er een reden voor dat dit standaard inactief is (optioneel, standaard is ˝false˝)

A.3.5 Logische sectie

<logic> (logica)

Het element dat de logische sectie bevat. Alle onderstaande elementen kunnen alleen in het <logic> element voorkomen. Het <logic> element kan alleen een directe child van het <document>-element zijn (hoogstens een per document), of van <optionset>-elementen (hoogstens een per optionset). De logische sectie van het document geldt op dezelfde manier voor zowel <dialog> en <wizard>-GUIs.

<external> (extern)

Maakt een nieuwe (string) eigenschap aan die bedoeld wordt als aanknopingspunt voor een externe eigenschap als de plugin ingebed wordt. Zie hoofdstuk over ˝incomplete˝ plugins. Attributen:

id
De ID (naam) van de nieuwe eigenschap (vereist)

standaard
De standaard string-waarde van de nieuwe eigenschap, bijv. de gebruikte waarde, indien de eigenschap niet verbonden is met een externe eigenschap (optioneel, standaard is een lege string)

<i18n> (internationalisatie)

Maakt een nieuwe (string)eigenschap aan bedoeld voor een ge-internationaliseerde naam. Attributen:

id
De ID (naam) van de nieuwe eigenschap (vereist)

label
De naam. Deze wordt straks vertaald. (vereist)

<set> (verzameling)

Stelt een vaste waarde in voor een eigenschap (natuurlijk blijft die niet vast als u de eigenschap ook verbindt met een andere eigenschap). Bijvoorbeeld, als u een plugin inbedt, maar enkele van zijn elementen wilt verbergen, kunt u de zichtbaarheid- eigenschap van die elementen op ˝false˝ instellen. Vooral nuttig voor inbeddende/ingebedde plugins. Merk op: als er meerdere <set>-elementen zijn met dezelfde `id`, heeft de laatst gedefinieerde voorrang. Dit kan soms nuttig zijn om op te vertrouwen bij het gebruik van <include>d onderdelen (parts). Attributen:

id
De ID (naam) van de in te stellen eigenschap (vereist)

aan
De in te stellen string-waarde van de eigenschap (vereist). Merk op: voor booleaanse eigenschappen zoals zichtbaarheid, aan staan, stelt u het attribuut typisch in op =˝true˝ of op =˝false˝.

<convert> (converteren, omzetten)

Maakt een nieuwe booleaanse eigenschap aan die afhangt van de toestand van een of meer verschillende eigenschappen. Attributen:

id
De ID (naam) van de nieuwe eigenschap (vereist)

bronnen
De ids (namen) van de eigenschappen waar deze eigenschap vanaf hangt. Een of meer eigenschappen kunnen worden genoemd, gescheiden door een ˝;˝. (vereist)

modus
De modus van de conversie/bewerking. Een van ˝equals˝ (gelijk aan), ˝notequals˝ (ongelijk aan), ˝range˝ (bereik), ˝and˝ (en), ˝or˝ (of). Indien in ˝equals˝-modus, is de eigenschap alleen ˝true˝, als de waarde van al zijn sources gelijk zijn aan de attribuut-standaard (zie onder). Indien in de ˝notequals˝-modus, is de eigenschap alleen dan ˝true˝ indien de waarde van al zijn sources verschillen met de attribuut-standaard (zie onder). Indien in ˝range˝-modus moeten de sources numeriek zijn (integer of real). De eigenschap is alleen dan ˝true˝, indien alle bronnen binnen het bereik liggen die gegeven wordt door de attributen min en max (zie onder). In ˝and˝-modus moeten de bronnen booleaanse eigenschappen zijn. De eigenschap is alleen dan ˝true˝, als alle bronnen tegelijk ˝true˝ zijn. In ˝or˝-modus, moeten de sources booleaanse eigenschappen zijn. De eigenschap is alleen dan ˝true˝, als minstens een van de bronnen ˝true˝ is. (vereist)

standaard
: Alleen zinvol in de modi ˝equals˝ of ˝notequals˝: de string-waarde waarmee wordt vergeleken (vereist indien in een van deze modi)

min
: Alleen zinvol in de modus ˝range˝: de kleinste waarde waarmee wordt vergeleken (optioneel, standaard is de kleinste real die de machine aan kan)

max
: Alleen zinvol in de modus ˝range˝: de grootste waarde waarmee wordt vergeleken (optioneel, standaard is de grootste real die de machine aan kan)

require_true (vereist_waar)
: Indien ingesteld op ˝true˝, wordt de eigenschap vereist, en is die alleen geldig als de toestand true/on is. Dus, als de eigenschap ˝false˝ is, blokkeert die de knop **Indienen** (optioneel, standaard is ˝false˝).

> LET OP
> Als u dit gebruikt, moet u ervoor zorgen dat de gebruiker gemakkelijk kan vinden wat er mis is, zoals het tonen van een uitleggende <text>.

<switch> (schakelaar)

Maak een nieuwe eigenschap aan die naar verschillende doel-eigenschappen (target properties) (of vaste strings) doorgeeft (relay), op basis van de waarde van een conditie-eigenschap. Hierdoor kunt u logica opbouwen die vergelijkbaar is met de constructs `if()` of `switch()`. Attributen:

id
: De ID (naam) van de nieuwe eigenschap (vereist)

conditie
: De id (naam) van de conditie-eigenschap (vereist)

Child elementen: (een child is een voortzetting van een parent. Dit soort termen zijn algemeen in een moderne computertaal)

<true> (waar)
: Indien de conditie-eigenschap een boolean is, kunt u de twee child elementen <true> en <false> opgeven (en deze alleen). (vereist, als <false> ook gegeven wordt)

<false> (onwaar)
: Indien de conditie-eigenschap een boolean is, kunt u de twee child elementen <true> en <false> opgeven (en deze alleen). (vereist, als <true> ook gegeven wordt)

<case> (in geval dat)
: Indien de conditie-eigenschap een boolean is, kunt u een willekeurig aantal <case>-elementen opgeven, een voor elke waarde van de conditie-eigenschap die u wilt gebruiken (minstens een zo'n element is vereist, indien de conditie niet een boolean is)

<default> (standaard)
: Indien de conditie-eigenschap geen boolean is, maakt het optionele <default>-element het mogelijk het gedrag op te geven, indien geen <case>-element overeenkomt met de waarde van de conditie-eigenschap (optioneel, alleen een keer toegestaan, in combinatie met een of meer <case>-elementen).

Child-elementen <true>, <false>, <case>, en <default> hebben de volgende attributen:

standaard
: Alleen voor <case>-elementen: de waarde waaraan de conditie-eigenschap moet voldoen (vereist, string).

fixed_value (vaste_waarde)
: Een vaste string die moet worden opgegeven als de waarde van de <switch>-eigenschap, als de huidige conditie klopt (vereist, als de dynamic_value niet wordt gegeven).

dynamic_value (dynamische_waarde)
De `id` (naam) van de doeleigenschap die moet worden gegeven als de waarde van de <switch>-eigenschap, als de huidige conditie klopt (vereist, als fixed_value niet is gegeven).

<connect> (verbinden)
Verbindt twee eigenschappen. De client-eigenschap wordt gewijzigd zodra de governor-eigenschap verandert (maar niet andersom!). Attributen:

client
De ID van de client_eigenschap, bijv. de aan te passen eigenschap (vereist)

governor
De ID van de governor-eigenschap, bijv. de eigenschap die de klanteigenschap aanpast. Deze kan een modifier inhouden (vereist)

reconcile (in overeenstemming brengen)
Indien ˝true˝zal de client-eigenschap de governor-eigenschap bij verbinding zo aanpassen dat de governor-eigenschap alleen waarden accepteert die ook door de client worden geaccepteerd (bijv. stel dat de governor een numerieke eigenschap is met als kleinste waarde ˝0˝, en de client een numerieke eigenschap met als kleinste waarde ˝100˝. Als reconcile de waarde ˝true˝heeft wordt voor beide eigenschappen de kleinste waarde ˝100˝). Dit werkt in het algemeen alleen indien beide eigenschappen van hetzelfde basistype zijn (optioneel, standaard is dit ˝false˝)

<dependency_check> (controleren op afhankelijkheden)
Maakt een boolean eigenschap aan die true is, indien aan de opgegeven afhankelijkheden wordt voldaan, en anders false. De xml-syntaxis van het element is dezelfde als die van de **<dependencies>**-elementen, beschreven in de `.pluginmap`-naslag. Vanaf RKWard 0.6.1, worden alleen de RKWard en R versiespecificaties in beschouwing genomen, en niet de afhankelijkheden van pakketten en pluginmaps.

<script>
Definieer scriptcode voor de besturing van de UI-logica. Zie de sectie voor GUJI-logica in scripts voor de details. De te gebruiken scriptcode kan in het `´´file´´`(bestand) -attribuut worden gegeven, , of als een tekst met commentaar van het element. Het **<script>**-element is niet toegestaan in de **<logic>** sectie van een optionset. Attributen:

bestand
Bestandsnaam van het scriptbestand (vereist)

A.4 Eigenschappen van plugin-elementen

Alle layout-elementen, en alle actieve elementen hebben de volgende eigenschappen, toegankelijk via de ˝id_van_element.naam_van_eigenschap˝:

zichtbaar
Of het GUI-element zichtbaar is of niet (boolean)

ingeschakeld
Of het GUI-element is ingeschakeld of niet (boolean)

vereist
Of het GUI-element vereist is (om een geldige instelling te bevatten) of niet. Let op dat elk element dat is uitgeschakeld of verborgen, impliciet niet vereist is (boolean)

Daarbij kunnen sommige elementen extra eigenschappen hebben die u kunt verbinden. De meeste actieve elementen hebben ook een standaard eigenschap waarvan de waarde wordt teruggegeven na het aanroepen van `getBoolean/getString/getList (´´...´´)`, indien er geen specifieke eigenschap werd genoemd, zoals hieronder beschreven:

<text> (tekst)

Standaard eigenschap is tekst

tekst
De getoonde tekst (text)

<varselector> (variabelekiezer)

Geen standaard eigenschap

geselecteerd
De nu geselecteerde objecten. Dit heeft u waarschijnlijk niet nodig. Wordt intern gebruikt (Robject)

root
Het root/parent object van de objecten waaruit u kunt kiezen (RObject)

<varslot>

Standaard eigenschap is ˝available˝ (beschikbaar)

available
Alle objecten in de varslot (RObject)

geselecteerd
Van de objecten in de varslot, die objecten die nu zijn geselecteerd. Dit heeft u waarschijnlijk niet nodig. Wordt intern gebruikt (RObject)

bron
Een kopie van de geselecteerde objecten in de overeenkomende varselector. Dit heeft u waarschijnlijk niet nodig. Wordt intern gebruikt (RObject)

<valueselector> (waarde selector)

Standaard eigenschap is ˝geselecteerd˝

geselecteerd
De nu geselecteerde strings. Modifier ˝labeled˝ voor het ophalen van de overeenkomstige namen. In een <valueselector> heeft u dit waarschijnlijk niet direct nodig, (alleen in een <select>). (read/write StringList)

available
De lijst van string-waarden waaruit u kunt kiezen. (read/write StringList)

labels
Te tonen naamteksten voor de stringwaarden. (read/write StringList)

<valueslot>

Zelfde als <varslot>, maar de eigenschappen zijn lijsten met strings, in plaats van RObjecten.

<radio> (radioknoppen, selectierondjes)

Standaard eigenschap is ˝string˝

string, tekenreeks
De waarde van de nu geselecteerde optie (string)

aantal
Het nummer van de nu geselecteerde optie (opties zijn van boven naar beneden genummerd, te beginnen met 0) (integer (geheel getal))

<dropdown> (neerklapmenu)

Zelfde als <radio>

<select> (kies, selecteer)

Zelfde als <valueselector>

<option> (optie)

Geen standaard eigenschap. ˝enabled˝ (ingeschakeld) is de *enige* eigenschap, en is op dit moment niet beschikbaar voor opties in een <select> of <valueselector>. <option> heeft niet de eigenschappen ˝visible˝ (zichtbaar) of ˝required˝ (vereist).

ingeschakeld

Of deze ene eigenschap moet worden aan- of uitgezet. In de meeste gevallen wilt u de gehele <radio< of <dropdown< (keuzelijst) aan/uitzetten. Maar u kunt dit gebruiken om het aangezet zijn dynamisch in te stellen voor een enkele optie in een <radio< of <dropdown< (bool)

<checkbox> (keuzevakje)

De standaard eigenschap is ˝state.labeled˝, wat betekent dat de waarden opgegeven in de attributen `value` (waarde), en `value_unchecked` (waarde_niet_geactiveerd) worden teruggegeven, en *niet* de tekst getoond bij het keuzevakje.

status

Status van het keuzevakje (aan of uit). Merk op dat de nuttige modifiers van deze eigenschap (zoals van alle boolean eigenschappen) zijn ˝not˝ en ˝labeled˝ (van naam voorzien) (zie eigenschap-typen). Maar het is vaak beter geen modifier te koppelen aan een eigenschap, bijv. ˝*checkbox_id*.state˝, die de status teruggeeft van een keuzevakje in een formaat waardoor het gebruikt kan worden in een if statement (0 of 1). (boolean)

<frame> (onvertaalbaar, zoiets als raamwerk)

Standaard eigenschap is ˝checked˝ (geactiveerd), als - en alleen als - de frame kan worden geactiveerd. Voor frames die niet kunnen worden geactiveerd is er geen standaard eigenschap.

geactiveerd

Alleen beschikbaar voor activeerbare frames: status van het keuzevakje (aan of uit). Let erop dat nuttige modifiers van deze eigenschap (zoals van alle boolean eigenschappen) zijn ˝not˝ en ˝numeric˝ (zie (see eigenschap-typen). (boolean)

<input> (invoer)

Standaard eigenschap is ˝text˝

tekst

Huidige tekst in het invoerveld (string)

<matrix>

Standaard eigenschap is ˝cbind˝,

rijen

Aantal rijen in de matrix (integer, geheel getal). Indien de gebruiker het aantal rijen in de matrix kan wijzigen, moet deze eigenschap worden beschouwd als read-only (alleen lezen). U kunt anders door dit te wijzigen, de grootte van de matrix veranderen.

kolommen

Aantal kolommen in de matrix (integer, geheel getal). Indien de gebruiker het aantal kolommen in de matrix kan wijzigen, moet deze eigenschap worden beschouwd als read-only (alleen lezen). U kunt anders door dit te wijzigen, de grootte van de matrix veranderen.

tsv
Gegevens in de matrix in het tsv-formaat (string: read-write (lezen-schrijven)). Merk op dat, vergeleken met de gebruikelijke tsv-indeling, de *kolommen*, en niet de rijen, van elkaar worden gescheiden door newline (nieuwe regel) karakters, en de cellen in een kolom door tab-karakters .

0,1,2...
De gegevens uit een kolom (de meest linkse kolom is 0). `getValue()`/`getString()` geven dit terug als een enkele string, gescheiden door ˝\n˝. Maar aangeraden wordt hiervoor de functie `getList()` te gebruiken, die deze kolom teruggeeft als een array van strings.

rij.0,rij.1,rij.2...
De gegevens uit een rij (de bovenste kolom is 0). `getValue()`/`getString()` geven dit terug als een enkele string, gescheiden door ˝\n˝. Maar aangeraden wordt hiervoor de functie `getList()` te gebruiken, die deze kolom teruggeeft als een array van strings.

cbind
Gegevens in een formaat dat geschikt is voor plakken in R, wrapped (een computerterm, die zoiets betekent als: ingevouwen, opgeborgen in) een cbind statement (string: read only (alleen lezen))

<optionset> (verzameling van opties)

Geen standaard eigenschap.

row_count
Aantal elementen in de optionset (integer). Read-only (alleen-lezen).

current_row (huidige rij)
Huidig actieve element in de optionset (integer). -1 als er geen actief element is. Read-write (Lezen-schrijven).

optioncolumn_ids (namen optionkolommen)
Voor elke <optioncolumn> die u definieert, wordt een string list eigenschap aangemaakt met de opgegeven id (naam).

<browser> (bladerprogramma)

Standaard eigenschap is ˝selection˝

selection (selectie)
Huidige tekst (geselecteerde bestandsnaam) in de browser (string)

<saveobject> (object opslaan)

Standaard eigenschap is ˝selection˝

selection (selectie)
Volledige naam van het geselecteerde object (string: read-only - gebruik ˝parent˝ en ˝objectnaam˝ voor instellen in een programma)

parent
Het parent object (het object dat de basis is van het huidige object) van het geselecteerde object. Dit is altijd een bestaand object in R van een type dat andere objecten kan bevatten (bijv. een list of data.frame). Indien ingesteld op een lege string of een ongeldig object, wordt ˝.GlobalEnv˝ aangenomen. (Robject)

objectnaam
De basisnaam van het geselecteerde object, bijv. de door de gebruiker ingevoerde string (gewijzigd naar een geldige naam in R indien nodig) (string)

actief
Alleen voor activeerbare saveobjects: of controle actief/aangezet is. Altijd true voor niet-activeerbare saveobjects (bool)

<spinbox> (spinveld)

Standaard eigenschap is altijd ˝int˝ of ˝real.formatted˝, afhankelijk van de modus van het spinveld

int
Gehele waarde in het spinveld, of dichtstbijzijnd geheel getal (integer)

real
Kommagetal in het spinveld (en eventueel geheel) (real)

<formula> (formule)

Standaard eigenschap is ˝model˝

model
De huidige model-string (string)

table (tabel)
De data.frame die de nodige variabelen bevat. Indien de variabelen van slechts een data.frame worden gebruikt, wordt de naam van de data.frame teruggegeven. Anders wordt een nodige nieuwe data.frame aangemaakt (string)

labels
Indien variabelen uit meerdere data.frames komen, kunnen hun namen botsen (bijvoorbeeld, indien in beide data.frames een variabele met de naam ˝x˝ voorkomt). Dit geeft een lijst terug met deze gebotste namen als indices (indexen) en de beschrijvende naam als waarde (string)

fixed_factors
De fixed factors. Dit heeft u waarschijnlijk niet nodig. Wordt intern gebruikt (RObject)

dependent
De afhankelijke variabele(n). Dit heeft u waarschijnlijk niet nodig. Wordt intern gebruikt (RObject)

<embed> (inbedden)

Geen standaard eigenschap

code
De (programma)code die door de ingebedde plugin wordt gegenereerd (code)

<preview> (voorbeeld, voorweergave)

Standaard eigenschap is ˝state˝ (status)

status
Of het keuzevakje voor het voorbeeld actief is (niet nodig als het voorbeeld al eerder werd getoond) (boolean)

<convert> (converteren, omzetten)

Dit element (gebruikt in de sectie <logic>) is bijzonder, omdat hettechnisch gesproken een eigenschap *is*, in plaats van dat het een of meer eigenschappen omvat. Het is booleanachtig. Merk op dat nuttige modifiers van deze eigenschap (zoals van alle boolean eigenschappen) ˝not˝ en ˝numeric˝ zijn (zie eigenschap-typen)

<switch> (schakelaar)

Dit element (gebruikt in de sectie <logic>) is bijzonder, omdat hettechnisch gesproken een eigenschap *is*, in plaats van dat het een of meer eigenschappen bevat. Hiermee kunt u tussen diverse doeleigenschappen schakelen, afhankelijk van de waarde van een conditie-eigenschap, of waarden van de conditie-eigenschap herindelen. Alle modifiers die u invoert worden doorgegeven naar de doeleigenschappen, dus, bijv. indien alle doeleigenschappen RObject-eigenschappen zijn, kunt u bij het schakelen ook de ˝korte naam˝-modifier gebruiken. Maar als de doeleigenschappen niet allemaal van hetzelfde type zijn kan het gebruiken van modifiers tot fouten leiden. Voor `fixed_value`s, (vaste waaarden) wordt elke modifier stilzwijgend genegeerd. Merk op dat doeleigenschappen, met een schakelaar beschikbaar, altijd read-only zijn.

A.5 Ingebedde plugins meegeleverd met de officiële uitgave van RKWard

Met RKWard worden een aantal ingebedde plugins meegeleverd, die in uw eigen plugins kunnen worden gebruikt. Uitvoerige documentatie is op dit ogenblik alleen beschikbaar in de bron- en helpbestanden van deze plugins. Maar hier volgt een kort overzicht van wat er beschikbaar is:

ID	**Pluginmap**	**Beschrijving**	**Voorbeeld van gebruik**
rkward::plot_options	embedded.pluginmap (ingebed.pluginmap)	Geeft een groot aantal opties voor plots. Wordt in de meeste plugins voor plotten gebruikt.	Plots->Barplot, meeste andere plugins voor plotten
rkward::color_chooser (kleurkiezer)	embedded.pluginmap (ingebed.pluginmap)	Zeer eenvoudige plugin voor kleuren. Huidige implementatie geeft een lijst met namen van kleuren. Toekomstige implementaties kunnen meer mogelijkheden geven voor het kiezen van een kleur.	Plots->Histogram
rkward::plot_stepfun_options	embedded.pluginmap (ingebed.pluginmap)	Plotopties voor stapfunctie	Plots->ECDF-plot
rkward::histogram_options	embedded.pluginmap (ingebed.pluginmap)	Histogram (plot) opties	Plots->Histogram
rkward::barplot_embed (staafdiagram inbedden)	embedded.pluginmap (ingebed.pluginmap)	Opties voor staafdiagrammen	Plots->Barplot
rkward::one_var_tabulation	embedded.pluginmap (ingebed.pluginmap)	Voor tabel maken van een enkele variabele.	Plots->Barplot
rkward::limit_vector_length	embedded.pluginmap (ingebed.pluginmap)	Beperk de lengte van een vector (tot de n grootste of kleinste elementen).	Plots->Barplot
rkward::level_select (level of niveau selecteren)	embedded.pluginmap (ingebed.pluginmap)	Geeft een <valueselector> gevuld met de levels (of unieke waarden) van een vector.	Data->Recode Categorical data

rkward::multi_input (multi invoer)	embedded.pluginmap (ingebed.pluginmap)	Combineert spinveld, (spinbox), input (invoer) en radioknoppen voor invoer van karakter, numerieke, logische gegevens.	Data->Recode Categorical data

Tabel A.1: Standaard plugins die kunnen worden ingebed

A.6 Elementen voor gebruik in `.pluginmap`-bestanden

<document>

Moet in elk `.pluginmap`-bestand aanwezig zijn als de root-node (precies een keer). Attributen:

base_prefix (basis voorvoegsel)

Bestandsnamen opgegeven in het `.pluginmap`-bestand worden relatief aangenomen ten opzichte van de directory van het `.pluginmap`-bestand + het voorvoegsel dat u hier opgeeft. Nuttig, vooral indien al uw componenten onder een enkele subdirectory zijn geplaatst.

namespace (onvertaalbaar, naamruimte)

Een namespace voor de component-ids (componentnamen). Wanneer bij inbedding de componenten worden gezocht, worden die vindbaar via een string ˝namespace::component_id˝. Stel dit voorlopig in op ˝rkward˝.

id

Een optionele identificatiestring voor deze `.pluginmap`. Dit opgeven maakt het mogelijk dat derde auteurs naar deze `.pluginmap` kunnen verwijzen en die kunnen inlezen vanuit die van hun (zie hoofdstuk voor omgaan met dependencies).

prioriteit (voorrang)

Een van *´´hidden´´* (verborgen, niet zichtbaar), *´´low´´* (laag), *´´medium´´* (midden), of *´´high´´* (hoog). `.pluginmap`s met ˝medium˝ of ˝high˝ priority worden automatisch actief gemaakt, op de plek waar RKWard ze de eerste keer vindt. Gebruik *priority=´´hidden´´* voor `.pluginmap`s die niet actief mogen worden, directory (?) (alleen bedoeld voor insluiten). Maar in de huidige implementatie wordt hierdoor de `.pluginmap` niet werkelijk verborgen. (Optioneel, standaard is ˝medium˝).

<dependencies> (afhankelijkheden)

Dit element, dat afhankelijkheden opgeeft, kan een directe child zijn van het <document>-element (een keer), en als een child van <component> -elementen (een keer voor elk <component>-element). Bevat de afhankelijkheden van de plugin waaraan moet worden voldaan. Zie het hoofdstuk over dependencies voor een overzicht. Attributen:

rkward_min_version, rkward_max_version

Toegestane maximum- en minimumversies van RKWard. Versie-specificaties kunnen niet-numerieke toevoegingen hebben, zoals ˝0.5.7z-devel1˝. Indien er aan een bepaalde afhankelijkheid niet wordt voldaan, worden de ontbrekende plugin(s) waar die op betrekking hebben, *genegeerd*. Meer informatie. Optioneel; indien niet opgegeven, zijn er geen minimum / maximum versies vereist van RKWard.

R_min_version, R_max_version

Toegestane maximum- en minimumversies van R. Versie-specificaties kunnen *niet* - numerieke toevoegingen hebben, zoals ˝0.5.7z-devel1˝. De afhankelijkheid van de R-versie wordt getoond in de help-pagina's van de plugin, maar heeft vanaf RKWard 0.6.1 geen direct effect.Meer informatie. Optioneel; indien niet opgegeven, zijn er geen minimum / maximum versies vereist van R.

Child elementen: (een child is een voortzetting van een parent. Dit soort termen zijn algemeen in een moderne computertaal)

<package> (pakket)

Voegt een afhankelijkheid (dependency) toe van een opgegeven R-pakket. Attributen:

naam
Pakketnaam (vereist)

min_version, max_version
Minimum / maximum toegestane versie (optioneel).

repository
Repository waarin het programma kan worden gevonden. Optioneel, maar hoogst gewenst, indien het pakket niet op CRAN beschikbaar is.

<pluginmap>

Voegt een afhankelijkheid toe van een specifieke RKWard `.pluginmap`. Attributen:

naam
Id-string (naamstring) van de vereiste `.pluginmap` (vereist).

min_version, max_version
Minimum / maximum toegestane versie (optioneel).

url
URL waar de `.pluginmap` kan worden gevonden. Vereist.

<about> (over)

Kan precies een keer aanwezig zijn als een directe child van het <document>- element. Bevat meta-informatie over de `.pluginmap` (of plugin). Zie het hoofdstuk ' over' informatie voor een overzicht. Attributen:

naam
Naam zichtbaar voor de gebruiker. Optioneel. Hoeft niet gelijk te zijn aan de ˝id˝.

versie
Versienummer. Optioneel. Er zijn geen beperkingen voor het formaat, maar voor de zekerheid kunt u de algemene versieschema's gebruiken, zoals ˝x.y.z˝.

releasedate (datum van uitgifte)

pecificatie van de uitgiftedatum. Optionele formaat is ˝YYYY-MM-DD˝ (JJJJ-MM-DD)

shortinfo (korte informatie)
Een *korte* beschrijving van de plugin / `.pluginmap`. Optioneel.

url
URL waar meer informatie kan worden gevonden. Optioneel, maar aanbevolen.

copyright
Copyright-specificatie, bijv. ˝2012-2013 door Pietje Puk˝. Optioneel, maar aanbevolen.

licentie
Licentiegegevens, bijv. ˝GPL˝ of ˝BSD˝. Zorg ervoor dat u bij uw bestanden een complete kopie toevoegt van de erbij horende licentie! Optioneel, maar aanbevolen.

categorie
Categorie van de plugin(s), bijv. ˝Item response theorie˝. Vanaf RKWard 0.6.1. worden geen categorieën vooraf gedefinieerd. Optioneel.

Child elementen: (een child is een voortzetting van een parent. Dit soort termen zijn algemeen in een moderne computertaal)

<author> (auteur)

Voegt informatie toe over een auteur. Attributen:

naam, voornaam, familienaam
Geef de volledige naam op voor `naam`, of geef de `voornaam` en de `familienaam`, apart op.

rol
Beschrijving rol van de auteur (optioneel)

e-mail
E-mail adres van de auteur. Vereist. Kan een rkward-devel mailing list adres zijn, als u er op bent ingeschreven, en uw plugin in de officiële RKWard uitgave moet worden opgenomen.

url
URL met nadere informatie over de auteur, bijv. homepage (optioneel).

<components>

Moet precies één keer aanwezig zijn als een directe child van het <document>-element. Bevat de afzonderlijke <component>-elementen zoals hieronder beschreven.Geen attributen.

<component>

Een of meer <component>-elementen moeten worden gegeven als directe children van het <componenten>-element (en alleen daar). Registreert een component/plugin met rkward. Attributen:

type
Voor toekomstige uitbreiding: Het type van component/plugin. Zet deze voorlopig steeds op ˝standaard˝ (het enige ondersteunde type).

id
De ID (naam) waarmee deze component kan worden opgehaald (om die in het menu te plaatsen (zie onder), of voor inbedding). Zie <document>-namespace boven.

bestand
Tenminste vereist voor componenten van het type=˝standard˝: de bestandsnaam van het XML-bestand dat de GUI beschrijft.

label
De naam voor deze component, wanneer die in de menu-hiërarchie wordt geplaatst.

<attribute> (attribuut)

Definieert een attribuut van een component. Tot nu toe allen van betekenis voor importeren van plugins. Alleen toegestaan als een directe child van <component>. Attributen:

id
Id (naam) van het attribuut

waarde
Waarde van het attribuut

labels
Naam behorende bij het attribuut

<hierarchy> (hiërarchie)

Moet precies een keer aanwezig zijn als een directe child van het <document>-element. Beschrijft waar de hierboven gedeclareerde componenten moeten worden geplaatst in de menu-hiërarchie. Accepteert alleen <menu>-elementen als directe children. Geen attributen.

<menu>

Een of meer <menu>-elementen moeten worden gegeven als directe children van het <hierarchy> element. Declareert een nieuw (sub)menu. Indien er al een menu bestaat met de opgegeven ID (zie onder), worden de twee menu's samengevoegd (merged). Het <menu>-element is toegestaan zowel als een directe child van het <hierarchy>-element (topniveau menu), of als de directe child van elk ander <menu>-element (submenu). Omgekeerd, accepteert het <menu>-element andere <menu>-elementen of <ingangs> (entry)-elementen als children. Attributen:

id

Een identificerende string van het menu. Nuttig wanneer menudefinities worden gelezen uit verschillende `.pluginmap`-bestanden, zodat zeker is dat plugins in het zelfde menu kunnen worden opgenomen. Sommige menu-namen, zoals ˝bestand˝ refereren aan bestaande menu's (in dit geval het ˝Bestand˝-menu). Zorg ervoor dat bestaande `.pluginmap`-bestanden hiermee consistente namen hebben.

label

Een naam voor het menu.

groep

Maakt het ordenen mogelijk van menu-ingangen. Zie Ordenen menu-ingangen. Optioneel.

<entry> (ingang)

Een menu-ingang, bijv. een menu-optie voor het aanroepen van een plugin. Kan alleen worden gebruikt als een directe child van een <menu>-element, en accepteert geen child-elementen (onderliggende elementen). Attributen:

component

De ID (naam) van de component die moet worden aangeroepen, wanneer deze menu-ingang wordt geactiveerd.

groep

Maakt het ordenen mogelijk van menu-ingangen. Zie Ordenen menu-ingangen. Optioneel.

<group> (groep)

Declareert een groep van ingangen in het menu. Zie Ordenen menu-ingangen. Attributes:

id

De naam van deze groep.

gescheiden

Optioneel. Indien ingesteld op ˝true˝ wordt de item van deze groep visueel gescheiden van de omringende items.

groep

De naam van de groep die aan deze groep moet worden gegeven (optioneel).

<context>

Declareert de ingangen in een context. Alleen toegestaan als een directe child van de <document>-tag. Accepteert alleen <menu>-tags als directe children. Attributen:

id

De ID (naam) van de context. Tot dusver zijn slechts twee contexts geïmplementeerd: ˝x11˝ en ˝import˝.

<require> (vereisen)

Een andere `.pluginmap`-bestand insluiten. Dit `.pluginmap`-bestand wordt slechts een keer ingelezen, zelfs als het voor verschillende andere bestanden <require>d is. Het belangrijkste voorbeeld hiervan is het opnemen van een pluginmap-bestand waarin componenten worden gedeclareerd, die ingebed zijn in componenten gedeclareerd in deze `.pluginmap`. <require>d elementen zijn alleen toegestaan als directe children van de <document>-node. Attributen:

bestand
De bestandsnaam van de op te nemen `.pluginmap`. Deze is relatief ten opzichte van de directory van het huidige `.pluginmap`-bestand + de base_prefix (zie boven, <document>-element).Indien u het relatieve pad naar de op te nemen `.pluginmap` niet kent, verwijst u in plaats daarvan naar de naam in het *`map`*-attribuut.

map
U kunt een `.pluginmap`-bestand uit een ander pakket opnemen (of een RKWard `.pluginmap` uit uw externe `.pluginmap`), door er naar te verwijzen met zijn *`namespacename::id`*, zoals opgegeven in het vereiste `.pluginmaps` <document>-element. Dit zal niet lukken indien geen`.pluginmap` met die naam bekend is (bijv. niet op uw systeem is geïnstalleerd). u mag dit alleen doen voor `.pluginmaps` die niet in uw pakket aanwezig zijn. Voor maps in uw pakket,is het opgeven van een relatief pad (*`file`* attribuut) sneller en betrouwbaarder.

A.7 Te gebruiken elementen in .rkh (help) bestanden

<document>
Moet aanwezig zijn in elk `.xml`-bestand als de root-node (precies een keer). Geen attributen.

<title>
Titel van de help-pagina. Dit geldt *niet* voor help-pagina's van een plugin (die krijgen de titel van de plugin zelf), alleen voor afzonderlijke pagina's. Geen attributen. De tekst opgenomen in de <title>-tag wordt de koptekst van de help-pagina. Mag alleen één keer worden gedefinieerd, als een directe child van de <document>-node.

<summary> (samenvatting)
Een korte samenvatting van de help-pagina (of waarvoor deze plugin wordt gebruikt). Dit is altijd zichtbaar bovenin de help-pagina. Geen attributen. De tekst in de <summary>-tag wordt getoond. Aanbevolen, maar niet vereist. Mag alleen maar een keer worden gedefinieerd, als een directe child van de <document>-node.

<usage> (gebruik)
Een ietwat meer uitvoerige samenvatting van het gebruik. Dit wordt altijd getoond direct na de <summary>. Geen attributen. De tekst in de <usage>-tag wordt getoond. Aanbevolen voor de help-pagina's van plugins, maar niet vereist. Mag alleen maar één keer worden gedefinieerd, als een directe child van de <document>-node.

<section> (sectie)
Een sectie voor alles. Kan elk aantal keren worden gebruikt als een directe child van de <document>-node. Deze secties worden getoond in de volgorde waarin ze zijn gedefinieerd, maar alle *na* de sectie <usage> en *voor* de sectie <settings>. De tekst in de <section>-tag wordt getoond.

id
Er is een identifier (naam) nodig om naar deze sectie te gaan vanuit de navigatiebalk (of een link). Moet binnen het bestand uniek zijn. Vereist, geen standaard.

titel
De titel (kop) van deze sectie. Vereist, geen standaard.

short_title
Een korte titel is handig voor het tonen in de navigatiebalk. Optioneel, standaard is de volledige titel.

<settings> instellingen)

Definieert de sectie die de verwijzing (referentie) bevat naar de diverse GUI-instellingen. Alleen van nut en alleen gebruikt voor help-pagina's bij plugins. Gebruik als een directe child van het <document>. Kan alleen <setting> en <caption>-elementen bevatten als directe children. Geen attributen.

<setting> (instellingen)

Verklaart een enkele instelling in de GUI. Alleen toegestaan als een directe child van het <settings>-element. De tekst in het element wordt getoond.

id
De ID (naam) van de instelling in de .xml van de plugin. Vereist, geen standaard.

titel
Een optionele titel voor de instelling. Indien weggelaten (dit wordt in de meeste gevallen aangeraden), wordt de titel gelezen in de .xml van de plugin.

<caption> (koptekst)

Een koptekst waarmee een aantal instellingen zichtbaar in groepen wordt verdeeld. Mag alleen worden gebruikt als een directe child van het <settings>-element.

id
De ID (naam) van het overeenkomende element (typisch een <frame>, <page> of <tab>) in de .xml van de plugin.

titel
Een optionele titel voor de koptekst. Indien weggelaten (dit wordt in de meeste gevallen aangeraden), wordt de titel gelezen in de .xml van de plugin.

<related> (gerelateerd)

Definieert een sectie die links bevat naar verdere informatie. Wordt altijd getoond na de sectie <settings>. Geen attributen. De tekst in de <related>-tag wordt getoond. Gewoonlijk bevat die een lijst in HTML-stijl. Aanbevolen voor help-pagina's van plugins, maar niet vereist. Mag alleen één keer worden gedefinieerd, als een directe child van de <document>-node.

<technical>

Definieert een sectie met technische informatie die niet bestemd is voor eindgebruikers (zoals de interne structuur van een plugin). Wordt altijd als laatste getoond in een help-pagina. Geen attributen. De tekst in de <related>-tag wordt getoond. Niet vereist, en niet aanbevolen voor de meeste help-pagina's van plugins. Mag alleen één keer worden gedefinieerd, als een directe child van de <document>-node.

<link> (koppeling)

Een koppeling. Kan in elk van de hierboven beschreven secties worden gebruikt.

href
De doel-url. Merk op dat diverse rkward-specifieke URLs beschikbaar zijn. Zie verder de sectie voor schrijven help-pagina's

<label> (naamtekst)

Voegt de waarde in van een UI-naam. Kan in elk van de hierboven beschreven secties worden gebruikt.

id
De id (naam) van het element in de plugin, waarvan het`label`-attribuut wordt gekopieerd.

<various html tags> (diverse html-tags)

In de secties worden de meeste basis html-tags toegestaan. Maar gebruik zelf formatteren zo weinig mogelijk.

A.8 Functies die in logische scripts voor de GUI kunnen worden gebruikt

Class ˝Component˝ (Class: onvertaalbaar, is een woord in de programmeertaal)

Class voor een enkele component of component-eigenschap. De belangrijkste instance (niet goed onvertaalbaar, variabele van deze class) hiervan is de variabele ˝gui˝ die al is gedefinieerd als de root-eigenschap van de huidige component. Voor instances van de class ˝Component˝ zijn de meeste volgende methods (functies) beschikbaar:

absoluteId(base_id)
Geeft de absolute ID terug van *base_id*, of - als base_id is weggelaten - de identificerende naam van de component.

getValue(id)
Afgeraden. Gebruik in plaats hiervan de functies `getString()`, `getBoolean()`, of `getList()`. Geeft de waarde terug van de gegeven child-eigenschap. Geeft de waarde terug van deze eigenschap, als ID wordt weggelaten.

getString(id)
Geeft de waarde terug als een string van de gegeven child-eigenschap. Geeft de waarde terug van deze eigenschap, als ID wordt weggelaten.

getBoolean(id)
Geeft de waarde terug als een boolean van de gegeven child-eigenschap (indien mogelijk). Geeft de waarde terug van deze eigenschap, als ID wordt weggelaten.

getList(id)
Geeft de waarde terug (indien mogelijk) als een array van strings van de gegeven child-eigenschap. Geeft de waarde terug van deze eigenschap, als ID wordt weggelaten.

setValue(id, waarde)
Stelt de waarde van de gegeven child-eigenschap in op *waarde*.

getChild(id)
Geeft een instance (variabele) terug van de child-eigenschap met de gegeven *id* (naam).

addChangeCommand(id, opdracht)
Doe de *opdracht*, steeds wanneer de child-eigenschap gegeven door *id* wordt gewijzigd.

Class ˝RObject˝

Class voor een enkel R object. Een instance (variabele) van deze class kunt u krijgen met **makeRObject(objectnaam)**. De volgende methods (functies) in een class) zijn beschikbaar voor instances van de class ˝RObject˝:

> WAARSCHUWING
> Als er in de backend nog opdrachten wachten, kan de door deze methods geleverde informatie verouderd zijn tegen de tijd dat de plugin-code wordt gestart. Vertrouw er *niet* op bij kritieke bewerkingen (kans op verlies van gegevens).

getName()
Geeft de absolute naam terug van het object.

exists()
Geeft terug of het object bestaat. In het algemeen moet u dit controleren voordat u een van de onderstaande methoden gebruikt.

dimensions()
Geeft een array van dimensies terug (net zoals **dim()** in R).

classes()
Geeft een array van classes terug (net zoals **class()** in R).

isClass(class)
Geeft true terug, als het object een class *class*-object is.

isDataFrame()
Geeft true terug als het object een data.frame is.

isMatrix()
Geeft true terug als het object een matrix is.

isList()
Geeft true terug als het object een list is.

isFunction()
Geeft true terug als het object een functie is.

isEnvironment()
Geeft true terug als het object een environment (omgeving) is.

isDataNumeric()
Geeft true terug als het object een vector is met numerieke gegevens (getallen).

isDataFactor()
Geeft true terug als het object een vector is met factorgegevens.

isDataCharacter()
Geeft true terug als het object een vector is met karaktergegevens.

isDataLogical()
Geeft true terug als het object een vector is met logische gegevens.

parent()
Geeft een instance (variabele) terug van ˝RObject˝ die de parent is van dit object.

child(childnaam)
Geeft een instance terug van ˝RObject˝ die de child *childnaam* is van dit object.

Class ˝RObjectArray˝

Een array van instances (variabelen) van RObject. Een instance van deze class kunt u verkrijgen met **makeRObjectArray(objectnamen)**. Dit is in het bijzonder bruikbaar bij varslots die het mogelijk maken meerdere objecten te selecteren.

include()-functie

include(bestandsnaam) kan worden gebruikt voor het opnemen van een afzonderlijk JS-bestand.

doRCommand()-functie

doRCommand(opdracht, callback) kan worden gebruikt om aan R informatie op te vragen. Lees de sectie informatie van R opvragen vanuit een plugin voor details, en waarschuwingen.

Bijlage B

Problemen oplossen bij het ontwikkelen van een plugin

U heeft dus alle documentatie gelezen, alles goed gedaan, en u kunt het nog steeds niet aan de gang krijgen? Wanhoop niet, we gaan er wat aan doen. Het eerste dat moet gebeuren is: activeer het venster van **RKWard Debug berichten** (in het **Vensters** - menu, of rechtsklik op een van de taakbalken), en start uw plugin nog een keer op. In het algemeen gesproken zult u nu geen uitvoer zien in het berichtenvenster, wanneer de plugin wordt gestart, of op elk ander moment. Als die er toch is, is het waarschijnlijk dat uw plugin daar de oorzaak van is. Zie of die uitvoer u verder kan helpen.

Indien alles er in de console goed uitziet, probeer dan de debug-level te verhogen (op de opdrachtregel doet u dit met **rkward --debug-level 3**, of door de debug level op 3 in te stellen in het menu **Instellingen** → **RKWard instellen** → **Debug**). Niet alle berichten in de hogere debug levels duiden noodzakelijk op een probleem, maar mogelijk duikt uw probleem ergens in de berichten op.

Indien u nog steeds niet kunt vinden wat er mis is, wanhoop dan niet. We weten dat dit lastig is, en - wie weet - mogelijk heeft u een bug in RKWard gevonden, en is het RKWard zelf dat moet worden gerepareerd. Schrijf ons op de development mailing list (in het Engels), en vertel ons wat uw probleem is. We zullen u graag helpen.

Tenslotte, zelfs als u zelf de oplossing heeft gevonden, maar de documentatie onvoldoende vindt, of zelfs op sommige punten onjuist, wilt u dit dan alstublieft melden op deze mailing list, zodat we de documentatie kunnen verbeteren.

Bijlage C

Licentie

Op- of aanmerkingen over de vertalingen van de toepassing en haar documentatie kunt u melden op http://www.kde.nl/bugs.

Dit document is vertaald in het Nederlands door Freek de Kruijf freekdekruijf@kde.nl.

Dit document is vertaald in het Nederlands door Jaap Woldringh jjhwoldringh op kde punt nl.