# Uvod v pisanje vtičnikov za RKWard

**Thomas Friedrichsmeier**
**Meik Michalke**
**Prevod: Matjaž Jeran**

Uvod v pisanje vtičnikov za RKWard

# Kazalo

# Tabele

**Povzetek**

To je vodnik za pisanje vtičnikov za RKWard.

# Poglavje 1

# Uvod

Ta dokument opisuje, kako napisati svoje lastne vtičnike. Dokumentacija se je sčasoma precej povečala. Naj vas to ne prestraši. Priporočamo, da preberete štiri osnovne korake (kot je opisano spodaj), da dobite osnovno predstavo o tem, kako stvari delujejo. Po tem boste morda želeli preleteti kazalo, da vidite, katere napredne teme bi lahko bile za vas pomembne.

Za vprašanja in komentarje pišite na RKWard razvojni poštni seznam.

*Tega vam ni treba brati, če želite uporabljati RKWard.* Ta dokument govori o razširitvi RKWard. Namenjen je naprednim uporabnikom ali ljudem, ki so pripravljeni pomagati izboljšati RKWard.

Pisanje standardnega vtičnika je v bistvu postopek v štirih korakih:

- Umestitev novega dejanja v hierarhijo menija
- Opis videza in delovanja vtičnika GUI
- Uporabnik določi, kako naj se generira koda R iz nastavitev GUI
- Dodajanje strani s pomočjo vašemu vtičniku

Ti se bodo obravnavali po vrsti.

Nekateri napredni koncepti se lahko uporabijo v teh štirih korakih, vendar so obravnavani v ločenih poglavjih, da bodo stvari preproste:

- GUI logika
- Vdelava vtičnikov v vtičnike
- Uporabni koncepti za ustvarjanje številnih serij podobnih vtičnikov

Prav tako nobeno od poglavij ne prikazuje vseh možnosti, temveč le osnovne pojme. Celoten sklic možnosti je na voljo posebej.

# Poglavje 2

# Uvod: kaj so vtičniki v RKWard? Kako delujejo?

Seveda je prvo vprašanje, ki ga morda imate: kateri deli RKWard funkcionalnosti je realizirana z vtičniki? Ali: kaj lahko storijo vtičniki?

Eden od načinov za odgovor na to je: prekličite izbiro vseh datotek `.pluginmap` pod **Nastavitve** → **Konfiguriraj RKWard** → **Vtičnike** in poglejte, kaj manjka. Nekoliko bolj koristen odgovor: večina dejanskih statističnih funkcij, dostopnih prek GUI so realizirani z uporabo vtičnikov. Z uporabo vtičnikov lahko ustvarite tudi dokaj prilagodljive GUI-je za vse vrste operacij.

Osnovna paradigma RKWard vtičnikov je tisti, skozi katerega vas bomo vodili v tem dokumentu: datoteka XML opisuje, kako naj bo videti GUI. Dodatna datoteka JavaScript se uporablja za ustvarjanje R sintakse iz GUI nastavitev. To pomeni, da vtičnikom v resnici ni treba izvajati nobenih statističnih izračunov. Namesto tega vtičniki ustvarijo R sintakso, potrebno za izvajanje teh izračunov. R sintaksa se nato pošlje v R zaledje za ovrednotenje in običajno je rezultat prikazan v izhodnem oknu.

V naslednjih poglavjih preberite, kako se to naredi.

# Poglavje 3

# Ustvarjanje menijskih vnosov

Ko ustvarite nov vtičnik, morate povedati RKWard o tem. Torej, prva stvar, ki jo morate narediti, je napisati datoteko `.pluginmap` (ali spremenite obstoječo). Oblika zapisa `.pluginmap` je XML. Vodil vas bom skozi primer (seveda se prepričajte, da imate RKWard konfiguriran za nalaganje vašega `.pluginmap` -- **Nastavitve** → **Konfiguriraj RKWard** → **Vtičniki**):

> NAMIG
> Ko preberete to poglavje, si oglejte tudi paket rkwarddev. Zagotavlja nekaj R funkcij za ustvarjanje večine RKWard-jevih XML oznak za vas.

```
<!DOCTYPE rkpluginmap>
```

Tip dokumenta se v resnici ne interpretira, vendar ga vseeno nastavite na *"rkpluginmap"*.

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

Atribut *base_prefix* je mogoče uporabiti, če so vsi vaši vtičniki v skupnem imeniku. V bistvu lahko ta imenik izpustite iz spodaj navedenih imen datotek. To je varno pustiti pri *""*.

Kot boste videli spodaj, vsi vtičniki dobijo edinstven identifikator, *id*. *imenski prostor* je način za organiziranje teh ID-jev in zmanjšanje možnosti, da bi po nesreči ustvarili podvojeni identifikator. Interno se v bistvu imenski prostor in nato '::' doda pred vse identifikatorje, ki jih podate v tem `.pluginmap`. Na splošno, če nameravate distribuirati svoje vtičnike v paketu R, je dobro uporabiti ime paketa kot parameter *namespace*. Vtičniki, dobavljeni z uradno RKWard distribucijo imajo *namespace="rkward"*.

Atribut *id* ni obvezen, vendar navedba ID-ja za vaš `.pluginmap` drugim ljudem omogoča, da njihovi `.pluginmap`-ji samodejno naložijo vaš `.pluginmap` (glejte razdelek o odvisnostih).

```
<komponente>
```

Komponente? Ali ne govorimo o vtičnikih? Da, vendar v prihodnosti vtičniki ne bodo nič več kot poseben razred komponent. Tukaj torej registriramo vse komponente/vtičnike pri RKWard. Poglejmo primer vnosa:

```
<component type="standard" id="t_test_two_vars" file="t_test_two_vars.xml" ↩
    label="t-test dveh spremenljivk" />
```

Najprej atribut *type*: to za zdaj pustite na *»standard«*. Nadaljnje vrste še niso implementirane. *id*, ki smo ga že nakazali. Vsaka komponenta mora dobiti edinstven (v svojem imenskem

prostoru) identifikator. Izberite tisto, ki je zlahka prepoznavna. Izogibajte se presledkom in kakršnim koli posebnim znakom. Ti zaenkrat niso prepovedani, vendar imajo lahko poseben pomen. Z atributom *file* določite, kje se nahaja opis dejanskega vtičnika. To je glede na imenik datoteka .pluginmap in *base_prefix* zgoraj. Na koncu dajte komponenti oznako. Ta oznaka bo prikazana povsod, kjer je vtičnik nameščen v meniju (ali v prihodnosti morda tudi na drugih mestih).

Običajno bo datoteka .pluginmap vsebovala več komponent, zato je tukaj še nekaj:

```
<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, to je bil prvi korak. RKWard zdaj ve, da ti vtičniki obstajajo. Toda kako jih priklicati? Umestiti jih je treba v hierarhijo menijev:

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

Takoj pod oznako **<hierarchy>** začnete opisovati, v kateri **<meni>** naj gredo vaši vtičniki. Z zgornjo vrstico v bistvu rečete, da mora biti vaš vtičnik v meniju **Analysis** (ne nujno neposredno tam, ampak v podmeniju). Meni **Analysis** je standarden v RKWard, zato ga dejansko ni treba ustvariti od začetka. Če pa še ne bi obstajal, bi mu z uporabo atributa *label* dali njegovo ime. Končno, *id* ponovno identificira ta **<meni>**. To je potrebno, zato je več .pluginmap datoteke lahko postavijo svoje vtičnike v iste menije. To storijo tako, da poiščejo **<meni>** z danim *id*. Če ID še ne obstaja, bo ustvarjen nov meni. V nasprotnem primeru bodo vnosi dodani v obstoječi meni.

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

Tu je v bistvu ista stvar: zdaj definiramo podmeni v meniju **Analysis**. Imenoval se bo **Means**.

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

In zadnja raven v hierarhiji menijev: podmeni podmenija **Means**.

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

Zdaj, končno, to je meni, v katerega želimo postaviti vtičnik. Oznaka **<entry>** signalizira, da je to pravzaprav prava stvar, namesto drugega podmenija. Atribut *component* se nanaša na *id*, ki ste ga dali zgornjemu vtičniku/komponenti.

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

Če ste izgubili sled: To je še en podmeni v meniju **Analysis**. Oglejte si spodnji posnetek zaslona. Nekaj nevidnega, označenega z [...], bomo preskočili.

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

To so zadnji vnosi, vidni na spodnjih posnetkih zaslona.

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

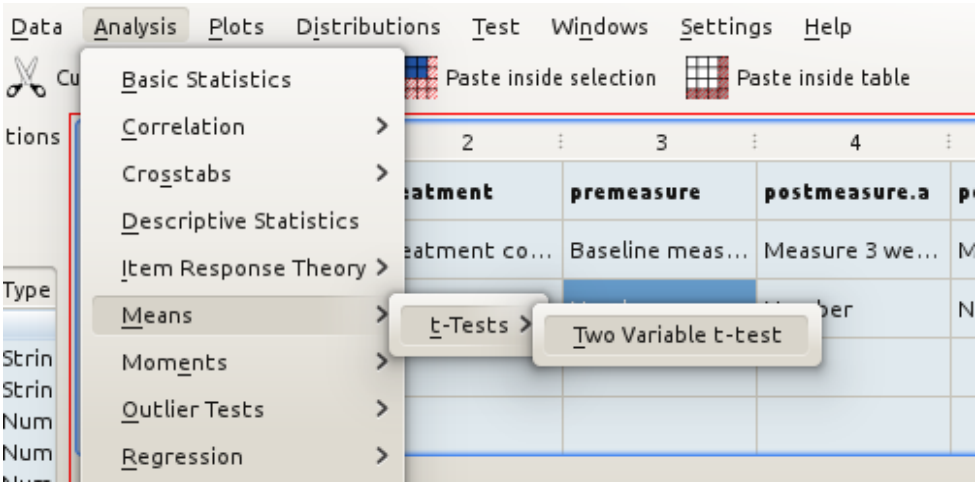
Seveda lahko svoje vtičnike postavite tudi v menije, ki niso **Analysis**.

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

Tudi v standardnih menijih, kot je **Datoteka**. Vse kar potrebujete je pravilen `id`.

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

Tako se to naredi. In ta posnetek zaslona prikazuje rezultat:




Zmedeni? Najlažji način za začetek je verjetno uporaba nekaterih obstoječih datotek `.pluginmap`, ki so priložene distribuciji, in jih prilagodite svojim potrebam. Če potrebujete pomoč, ne oklevajte in pišite na poštni seznam za razvoj.

## 3.1 Nadzor vrstnega reda vnosov v meniju

Privzeto bodo vsi elementi (vnosi/podmeniji) znotraj menija samodejno razvrščeni po abecedi. V *nekaterih* primerih boste morda želeli več nadzora. V tem primeru lahko elemente združite na naslednji način:

- Skupine lahko določite znotraj katerega koli menija, kot je ta. Vsi elementi, ki pripadajo isti skupini, bodo združeni:

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

- Če želite, da je skupina vizualno ločena od drugih vnosov, uporabite:

```
<group id="somegroup" separated="true"/>
```

- Vnose, menije in skupine je mogoče dodati določeni skupini z uporabo:

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

- Pravzaprav je možno tudi implicitno definirati skupine (brez ločilnih črt):

```
<entry component="first" group="a"/>
                <entry component="third"/>
                <entry component="second" group="a"/>
```

- Imena skupin so specifična za vsak meni. Skupina ″a″ v meniju ″Data″ na primer ni v nasprotju s skupino ″a″ v meniju ″Analysis″.
- Najpogostejši primer uporabe je definiranje skupin na vrhu ali na dnu menija. Za to sta v vsakem meniju vnaprej določeni skupini ″top″ in ″bottom″.
- Vnosi znotraj vsake skupine so razvrščeni po abecedi. Skupine se prikažejo po vrstnem redu deklaracije (razen če so dodane drugi skupini, seveda).
- Meniji in vnosi brez specifikacije skupine prav tako logično tvorijo skupino (″″).

# Poglavje 4

# Definiranje GUI

## 4.1 Definiranje pogovornega okna

V prejšnjem poglavju ste videli, kako registrirati vtičnik v RKWard. Najpomembnejša sestavina je bila določitev poti do XML datoteko z opisom, kako je vtičnik dejansko videti. V tem poglavju se boste naučili, kako ustvariti to XML datoteko.

> NAMIG
> Ko preberete to poglavje, si oglejte tudi paket rkwarddev. Zagotavlja nekaj R funkcij za ustvarjanje večine RKWard-jevih XML oznak za vas.

Še enkrat vas bomo popeljali skozi primer. Primer je (nekoliko poenostavljena) različica t-testa z dvema spremenljivkama.

```
<!DOCTYPE rkplugin>
```

Tip dokumenta še ni zares razložen. Vseeno ga nastavite na *rkplugin*.

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

Vsi vtičniki ustvarijo neko kodo. Trenutno je edini način za to uporaba JS, kot je podrobno opisano v naslednjem poglavju. To določa, kje iskati kodo JS. Ime datoteke je relativno glede na imenik vtičnika XML.

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

Običajno je dobro, da za svoj vtičnik zagotovite tudi stran s pomočjo. Ime datoteke te strani s pomočjo je podano tukaj glede na imenik, vtičnik XML Pisanje strani s pomočjo je dokumentirano tukaj. Če ne zagotovite datoteke pomoči, izpustite to vrstico.

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

Kot veste, imajo lahko vtičniki pogovorno okno ali vmesnik čarovnika ali oboje. Tukaj začnemo definirati pogovorni vmesnik. Atribut *label* določa napis pogovornega okna.

```
<tabbook>
                        <tab label="Basic settings">
```

GUI elemente je mogoče organizirati s knjigo zavihkov (tabbook). Tukaj definiramo tabbook kot prvi element v pogovornem oknu. Uporabite **<tabbook>**[...]**</tabbook>**, da definirate zavihek, nato pa za vsako stran v zavihku uporabite **<tab>**[...]**</tab>**. Atribut *label* v elementu **<tab>** vam omogoča, da določite napis za to stran knjige zavihkov.

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

Oznaki **<row>** in **<column>** določata postavitev GUI elementov. Tukaj pravite, da bi radi postavili nekaj elementov drug ob drugem (od leve proti desni). Atribut *id* ni nujno potreben, vendar ga bomo uporabili pozneje, ko bomo našemu vtičniku dodajali vmesnik čarovnika. Prvi element, ki ga postavite v vrstico, je:

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

S to preprosto oznako ustvarite seznam, s katerega lahko uporabnik izbere spremenljivke. Za ta element morate določiti *id*, zato RKWard ve, kako ga najti.

> POZOR
> V nizu *id* NE smete uporabiti pike (.).

```
<column>
```

Nato v vrstico ugnezdimo **<column>**. To pomeni, da bodo naslednji elementi postavljeni drug nad drugim (od zgoraj navzdol) in vsi bodo desno od **<varselector>**.

```
<varslot types="number" id="x" source="vars" required="true" label="compare ↩
    "/>
                                                <varslot types="number" id ↩
                                                    ="y" source="vars"  ↩
                                                    required="true" label=" ↩
                                                    against" i18n_context=" ↩
                                                    compare against"/>
```

Ti elementi so protipostavka **<varselector>**. Predstavljajo 'reže (slots)', v katere lahko uporabnik vstavi spremenljivke. Opazili boste, da je *source* nastavljen na isto vrednost kot *id* **<varselector>**. To pomeni, da bo vsak **<varslot>** prevzel svojo spremenljivko iz izbirnika spremenljivk. **<varslot>** morajo prav tako dobiti *id*. Lahko imajo *label* in so lahko nastavljeni na *required*. To pomeni, da gumb **Pošlji** ne bo omogočen, dokler **<varslot>** ne vsebuje veljavne vrednosti. Končno atribut *type* še ni interpretiran, vendar bo uporabljen za zagotavljanje, da bodo v **<varslot>** dovoljeni samo pravilni tipi spremenljivk.

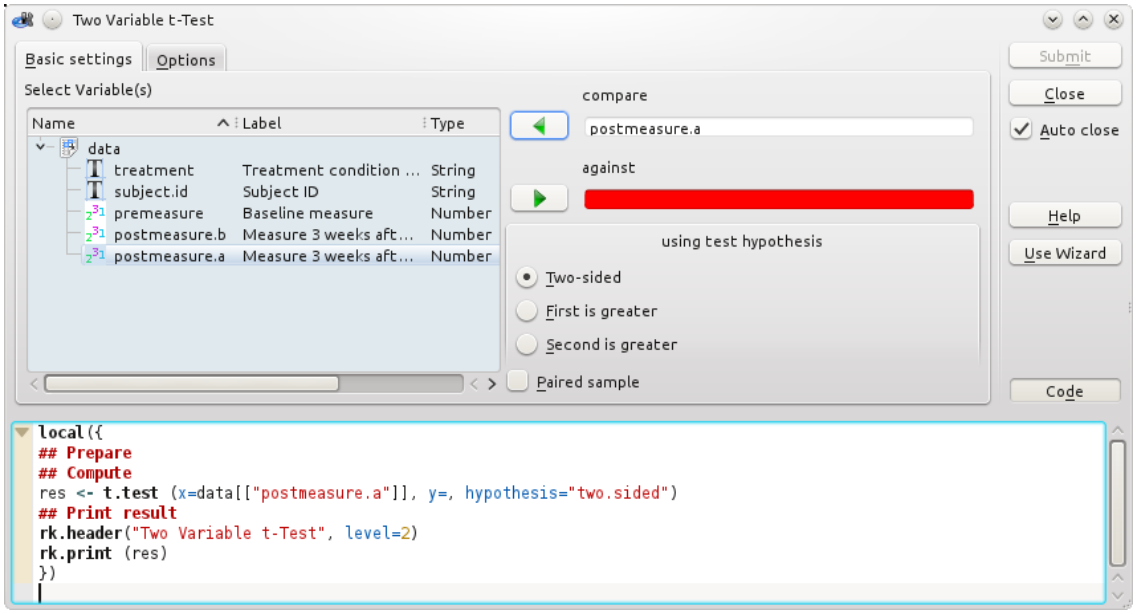
Če se sprašujete o atributu *i18n_context*: To zagotavlja kontekst za pomoč pri pravilnem prevodu besede "against", ki se uporablja kot **<varslot>**' s oznako, vendar ne vpliva neposredno na funkcionalnost vtičnika. Več o tem v ločenem poglavju.

```
<radio id="hypothesis" label="using test hypothesis">
                                                        <option value="two. ↩
                                                            sided" label=" ↩
                                                            Two-sided"/>
                                                        <option value=" ↩
                                                            greater" label=" ↩
                                                            First is greater ↩
                                                            "/>
                                                        <option value="less ↩
                                                            " label="Second  ↩
                                                            is greater"/>
                                                <</radio>
```

Tukaj določite skupino ekskluzivnih gumbov **<radio>**. Skupina ima *label* in *id*. Vsak **<option>** (gumb) ima *oznako* in mu je dodeljena *vrednost*. To je vrednost, ki jo element **<radio>** vrne, ko je izbrana možnost.

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

Vsako oznako je treba zapreti. Vse elemente, ki smo jih želeli (dva **<varslots>** in **<radio>**) smo postavili v **<column>**. Vse želene elemente (**<varselector>** in **<column>** s temi elementi) postavimo v **<row>**. Vse elemente, ki smo jih želeli, smo postavili na prvo stran v **<tabbook>**. Nismo še končali z definiranjem **<tabbook>** (prihaja več strani), seveda pa je še več v **<pogovornem oknu>**. Toda ta posnetek zaslona je v bistvu to, kar smo naredili do zdaj:




Upoštevajte, da nismo določili **Pošlji**,**Zapri** itd. gumbi ali pogled kode. Ti elementi seustvarijo samodejno. Seveda pa moramo še definiratidrugo stran **<tabbook>**:

```
<tab label="Options">
                                <checkbox id="varequal" label="assume equal ↩
                                    variances" value=", var.equal=TRUE"/>
```

Elementi bodo privzeto postavljeni od zgoraj navzdol kot v **<column>**. Ker je to tisto, kar želimo tukaj, nam ni treba izrecno navesti postavitve **<row>** ali **<column>**. Prvi element, ki ga definiramo, je potrditveno polje. Tako kot **<radio><option>** ima potrditveno polje *oznako* in *vrednost*. *vrednost* je tisto, kar se vrne, če je potrditveno polje potrjeno. Seveda potrditveno polje potrebuje tudi *id*.

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

Tukaj je še en element postavitve: Da bi signalizirali, da spodnja elementa spadata skupaj, narišemo **<frame>** (polje). Ta okvir ima lahko *oznako* (napis). Ker je okvir samo pasivni element postavitve, ne potrebuje *id*, vseeno pa ga definiramo tukaj, saj se bomo nanj sklicevali kasneje, ko bomo definirali dodatni vmesnik čarovnika.

```
<checkbox id="confint" label="print confidence interval" value="1" checked ↩
    ="true"/>
                                        <spinbox type="real" id="conflevel" ↩
                                            label="confidence level" min ↩
                                           ="0" max="1" initial="0.95"/>
```

```
                                    </frame>
```

Znotraj **<frame>** postavimo še en **<checkbox>** (z uporabo *checked="true"*, znak, da mora biti potrditveno polje privzeto potrjeno), in **<spinbox>**. Spinbox uporabniku omogoča izbiro vrednosti med *"min"* in *"max"* s privzeto/začetno vrednostjo *"0, 95"*. Nastavitev *type* na *"real"* signalizira, da so realna števila sprejeta v nasprotju s *type="integer"*, ki bi sprejel samo cela števila.

> OPOMBA
> Prav tako je mogoče in pogosto bolje, da se omogoči preverjanje samega **<frame>**, namesto dodajanja **<checkbox>** znotraj. Za podrobnosti si oglejte referenco. To tukaj ni storjeno zaradi ponazoritve.

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

To je vse za drugo stran **<tabbook>**, vse strani v **<tabbook>** in vse elemente v **<pogovornem oknu>**. Končali smo z definiranjem, kako je videti pogovorno okno.

```
</document>
```

Na koncu zapremo oznako **<document>** in to je to. GUI je definiran. Zdaj lahko shranite datoteko. Kako pa se R sintaksa ustvari iz GUI nastavitve? S tem se bomo ukvarjali v naslednjem poglavju. Najprej pa si bomo ogledali dodajanje vmesnika čarovnika in nekaj splošnih premislekov.

## 4.2 Dodajanje vmesnika čarovnika

Pravzaprav nam ni treba definirati dodatnega vmesnika **<wizard>**, toda tukaj je in tako bi to naredili. Če želite dodati vmesnik čarovnika, boste dodali oznako **<wizard>** na isti ravni kot oznako **<dialog>**:

```
<wizard label="Two Variable t-Test">
                        <page id="firstpage">
                                <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.</text ↩
                                            >
                                <copy id="main_settings_row"/>
                        </page>
```

Nekaj od tega je precej razumljivo: dodamo oznako **<wizard>** z *label* za čarovnika. Ker lahko čarovnik vsebuje več strani, ki so prikazane ena za drugo, nato definiramo prvo **<page>** in vanjo dodamo razlagalno opombo **<text>**. Nato uporabimo oznako **<copy>**. Kaj to stori, je, da nam res prihrani ponovno definiranje, kar smo že napisali za **<dialog>**: Oznaka za kopiranje išče drugo oznako z enakim *id* prej v XML. To je definirano v razdelku **<dialog>** in je **<row>**, v kateri je **<varselector>** , **<varslots>** in nadzor 'hypothesis' **<radio>**. Vse to je kopirano 1:1 in vstavljeno tik ob element **<copy>**.

Zdaj pa na drugo stran:

```
<page id="secondpage">
                                <text>Below are some advanced options. It  ↩
                                    is 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.</text>
                                <copy id="varequal"/>
                                <text>Sometimes it is 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).</text>
                                <copy id="frame_conf_int"/>
                        </page>
                </wizard>
```

Tukaj je skoraj isto. Dodamo nekaj besedil in vmes ta **<copy>** nadaljnje razdelke iz pogovornega vmesnika.

Seveda lahko naredite, da je vmesnik čarovnika zelo drugačen od navadnega pogovornega okna in sploh ne uporabite oznake **<copy>**. Vendar se prepričajte, da ustreznim elementom dodelite isti *id* v obeh vmesnikih. To se ne uporablja le za prenos nastavitev iz pogovornega vmesnika v vmesnik čarovnika in nazaj, ko uporabnik zamenja vmesnik (kar se v trenutni različici RKWard še ne zgodi), ampak tudi poenostavi pisanje predloge kode (glejte spodaj) .

## 4.3 Nekaj pomislekov o GUI oblikovanju

Ta razdelek vsebuje nekaj splošnih premislekov, na podlagi katerih GUI elementov za uporabo kje. Če je to vaš prvi poskus ustvarjanja vtičnika, lahko ta razdelek preskočite, saj ni pomemben za pridobitev osnovnega GUI delovanja. Pozneje se vrnite sem, da vidite, ali lahko izboljšate svoj GUI tako ali drugače.

### 4.3.1 <radio> v primerjavi s <checkbox> v primerjavi s <dropdown>

Vsi trije elementi **<radio>**, **<checkbox>**, **<dropdown>** imajo podobno funkcijo: za izbiro enega izmed več možnosti. Očitno potrditveno polje omogoča izbiro le med dvema možnostma: potrjeno ali nepotrjeno, zato ga ne morete uporabiti, če lahko izbirate med več kot dvema možnostma. Toda kdaj uporabiti katerega od elementov? Nekaj osnovnih pravil:

Če ugotovite, da ustvarjate **<radio>** ali **<dropdown>** s samo dvema možnostma, se vprašajte, ali je vprašanje v bistvu tip vprašanja da/ne. Npr. izbira med 'prilagodi rezultate' in 'ne prilagodi rezultatov' ali med 'odstrani manjkajoče vrednosti' in 'ohrani manjkajoče vrednosti'. V tem primeru je **<checkbox>** najboljša izbira: porabi malo prostora, ima najmanj besed oznak in ga uporabnik najlažje prebere. Obstaja zelo malo situacij, ko bi morali izbrati **<radio>** namesto **<potrditveno polje>**, ko sta na voljo samo dve možnosti. Primer tega je lahko: 'Metoda izračuna: 'pearson'/'spearman''. Tu si lahko predstavljamo več metod, ki v resnici ne tvorijo para nasprotij.

Izbira med **<radio>** in **<dropdown>** je večinoma vprašanje prostora. **<dropdown>** ima prednost, ker zavzame malo prostora, tudi če lahko izbirate med številnimi možnostmi. Po drugi strani

pa ima **<radio>** to prednost, da so vse možne izbire vidne uporabniku hkrati, brez klikanja na spustno puščico. Na splošno, če lahko izbirate med šestimi ali več možnostmi, je bolje uporabiti **<dropdown>**. Če obstaja pet ali manj možnosti, je **<radio>** boljša izbira.

# Poglavje 5

# Ustvarjanje R kode iz nastavitve GUI

## 5.1 Uporaba JavaScript v RKWard vtičnikih

Zdaj imamo GUI definiran, vendar moramo še vedno ustvariti nekaj R kode iz tega. Za to potrebujemo drugo besedilno datoteko, `code.js`, ki se nahaja v istem imeniku kot `description.xml`. Morda poznate JavaScript (ali če smo tehnično natančni: ECMA-script). Dokumentacijo o JS je mogoče najti v izobilju, tako v tiskani obliki kot na internetu (npr.: https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide). Toda za večino namenov vam sploh ne bo treba vedeti veliko o JS, saj bomo uporabljali le nekatere zelo osnovne funkcije.

> NAMIG
> Ko preberete to poglavje, si oglejte tudi rkwarddev paket. Zagotavlja nekaj R funkcij za ustvarjanje JavaScript kode, ki se pogosto uporablja v RKWard. Prav tako lahko samodejno zazna spremenljivke, uporabljene v vtičniku XML datoteko in ustvarite osnovni JavaScript kodo iz tega, s katero lahko začnete.

> OPOMBA
> Predpostavlja se, da so datoteke vtičnika `.js` kodirane z UTF-8. Ne pozabite preveriti kodiranja urejevalnika, če uporabljate znake, ki niso ascii.

Za t-test z dvema spremenljivkama je datoteka `code.js` videti takole (z vmesnimi komentarji):

### 5.1.1 preprocess()

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

Datoteka JS je organizirana v tri ločene funkcije: `preprocess()`, `calculate()` in `printout()`. To je zato, ker ni potrebna vsa koda na vseh stopnjah. Trenutno se funkcija predprocesa na mnogih mestih res ne uporablja (običajno jo boste v celoti izpustili).

### 5.1.2 calculate()

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

Ta funkcija ustvari dejansko R sintakso, ki se izvaja iz GUI nastavitev. Oglejmo si to podrobno: Koda, ki jo je treba uporabiti, je ustvarjena s stavkom `echo()`. Če pogledamo stavek `echo()` korak za korakom, je prvi del tega

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

kot golo besedilo. Nato moramo izpolniti vrednost, uporabnika, izbranega kot prvo spremenljivko. To pridobimo s pomočjo `getString (~x~)` in ga dodamo nizu, ki ga odmevamo 'echoed'. To natisne vrednost GUI-elementa z *`id=~x~`*: naš prvi **<checkbox>**. Nato dodamo ', ' in naredimo isto, da pridobimo vrednost elementa *`~y~`* - drugi **<checkbox>**. Za hipotezo (skupina **<radio>**) in enake variance **<checkbox>** je postopek zelo podoben.

Upoštevajte, da lahko namesto združevanja izhodnih izrezkov s '+' uporabite tudi več stavkov `echo()`. Vse je natisnjeno v eni vrstici. Če želite ustvariti prelom vrstice v ustvarjeni kodi, vstavite *`~\n~`* v odmevani niz. Teoretično lahko ustvarite celo veliko vrstic z enim stavkom echo, vendar naj bo le ena vrstica (ali manj) generirane kode na `echo()`.

> **OPOMBA**
> Poleg `getString()` obstajajo tudi funkcije `getBoolean()`, ki bodo poskušale vrniti vrednost kot logično (primerno za uporabo v `if()`-statement) in `getList()`, ki bo poskušal vrniti seznamom podobne podatke v JS `Array()`. Kasneje bomo pokazali primere teh.
> Ko si ogledujete obstoječe vtičnike, boste našli tudi veliko vtičnikov, ki uporabljajo `getValue()` namesto `getString()` in dejansko sta *skoraj* enaka. Vendar je uporaba `getString()`, `getBoolean()` in `getList()` priporočena praksa od različice 0.6.1.

Za raven zaupanja postane malo bolj težavno. Zaradi estetike ne želimo izrecno določiti stopnje zaupanja, ki naj se uporabi, če ustreza privzeti vrednosti. Zato namesto brezpogojnega tiskanja vrednosti najprej pridobimo v spremenljivki. Nato preverimo, ali se ta spremenljivka razlikuje od *`~0.95~`*, in če se, natisnemo dodatni argument. Nazadnje ponovimo zaključni oklepaj in prelom vrstice: *`~)\n~`*. To je vse za funkcijo izračuna.

### 5.1.3 printout()

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

In to je bilo v večini primerov vse, kar je pri funkciji tiskanja. `rk.header()` natisne standardni naslov za rezultate. Upoštevajte, da morate v datotekah `.js` ročno označiti vse nize, ki jih je mogoče prevesti, z uporabo **i18n()** ali nekaterih drugih ukazov. Več o tem v poglavju o internacionalizaciji. Temu lahko dodate še nekaj informacij, če želite, npr.:

```
function printout () {
        new Header (i18n ("Two Variable t-Test"))
                  .addFromUI ("varequal")
                  .add (i18n ("Confidence level"), getString ("conflevel")) ↩
                        // Note: written like this for illustration purposes ↩
                      . More automatic:
        //        .addFromUI ("conflevel")
                  .print ();
echo ('rk.print (res)\n');
}
```

`rk.print()` uporablja paket R2HTML za zagotavljanje HTML formatiranega izhoda. Druga koristna funkcija je `rk.results()`, ki lahko izpiše tudi različne vrste tabel rezultatov. Če ste v dvomih, preprosto uporabite `rk.print()` in zaključite. Razred JS `Header` je pomočnik na ravni JS za generiranje klica `rk.header()` (samo poglejte ustvarjeno kodo R). V nekaterih primerih boste morda želeli neposredno poklicati `echo ('rk.header (...)')`, da natisnete glavo za vaš izhod.

Upoštevajte, da je interno izhod le navaden HTML dokument v tem trenutku. Zato vas bo morda zamikalo dodati HTML z uporabo `rk.cat.output()`. Čeprav bo to delovalo, tega ne počnite. Izhodni format se lahko v prihodnosti spremeni (npr. v ODF), zato je najbolje, da HTML določeno kodo. Raje poenostavite stvari s `rk.header()`, `rk.print()`, `rk.results()` in - če je potrebno - - `rk.print.literal()`. Če se zdi, da ne izpolnjujejo vaših potreb po oblikovanju, se za pomoč obrnite na nas na poštnem seznamu.

Čestitke! Ustvarili ste svoj prvi vtičnik. Preberite v naslednjih poglavjih za naprednejše koncepte.

## 5.2 Konvencije, politike in ozadje

Obstaja veliko načinov za pisanje R kode za določeno nalogo in obstaja še več načinov za ustvarjanje te R kode iz JS. Kako točno boste to storili, je prepuščeno vam. Kljub temu morate upoštevati številne vidike in osnovne informacije, ki jih morate razumeti.

### 5.2.1 Razumevanje okolja `local()`

Pogosteje boste morali ustvariti enega ali več začasnih R predmetov v kodi, ki jo ustvari vaš vtičnik. Običajno ne želite, da so postavljeni v delovni prostor uporabnika, saj lahko celo prepišejo uporabniške spremenljivke. Zato se vsa koda, ustvarjena z vtičnikom, izvaja v okolju `local()` (glejte R stran s pomočjo za funkcijo `local()`). To pomeni, da so vse spremenljivke, ki jih ustvarite, začasne in ne bodo trajno shranjene.

Če uporabnik izrecno zahteva shranjevanje spremenljivke, boste morali temu objektu dodeliti z uporabo `.GlobalEnv$objectname <- value`. Na splošno ne uporabljajte operatorja `<<-`. Ni nujno, da bo dodelil v .GlobalEnv.

Ena pomembna past je uporaba `eval()`. Tukaj morate upoštevati, da bo eval privzeto uporabil trenutno okolje za vrednotenje, tj. lokalnega. To bo večinoma dobro delovalo, vendar ne vedno. Torej, če morate uporabiti `eval()`, boste verjetno želeli podati parameter *`envir`*: `eval(..., envir=globalenv())`.

### 5.2.2 Oblikovanje kode

Najpomembnejša stvar je za vaše ustvarjene R kode za delo, vendar mora biti tudi enostavna za branje. Zato bodite pozorni tudi na oblikovanje. Nekaj premislekov:

Normalna najvišja raven R stavki morajo biti levo poravnani.

Stavki v spodnjem bloku morajo biti zamaknjeni z enim tabulatorjem (glejte primer spodaj).

Če delate zelo zapletene izračune, tu in tam dodajte komentar, npr. za označevanje logičnih odsekov. Upoštevajte, da obstaja posebna funkcija **comment()** za vstavljanje prevedljivih komentarjev v ustvarjeno kodo.

Na primer, ustvarjena koda je lahko videti takole. Enako kodo brez zamikov ali komentarjev bi bilo precej težko brati, kljub skromni zapletenosti:

```
# first determine the wobble and rotation
my.wobble <- wobble (x, y)
my.rotation <- wobble.rotation (my.wobble, z)

# boggling method needs to be chosen according to rotation
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 Ukvarjanje s kompleksnimi možnostmi

Številni vtičniki lahko naredijo več kot eno stvar. Na primer, vtičnik 'Descriptive Statistics' lahko izračuna povprečje, obseg, vsoto, produkt, mediano, dolžino itd.. Vendar pa se bo uporabnik običajno odločil, da izvede le nekatere od teh izračunov. V tem primeru poskusite ohraniti ustvarjeno kodo čim bolj preprosto. Vsebovati mora samo dele, ki so pomembni za možnosti, ki so dejansko izbrane. Da bi to dosegli, je tukaj primer običajnih oblikovalskih vzorcev, kot bi jih uporabili (v JS; tukaj bi bili »domean«, »domedian« in »dosd« elementi <checkbox>):

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

        if (getBoolean ("domean.state")) echo ("results$" + i18n ("Mean  ↩
            value") + " <- mean (x)\n");
        if (getBoolean ("domedian.state")) echo ("results$" + i18n ("Median ↩
            ") + " <- median (x)\n");
        if (getBoolean ("dosd.state")) echo ("results$" + i18n ("Standard  ↩
            deviation") + " <- sd (x)\n");
        //...
}
```

## 5.3 Namigi in triki

Tukaj je nekaj izbranih trikov, s katerimi bo pisanje vtičnikov manj dolgočasno:

Če potrebujete vrednost GUI nastavitev na več mestih v vašem vtičnikukodo, razmislite o dodelitvi spremenljivki v JS in uporabi tega namestopridobivanje vedno znova z `getString()/getBoolean()/getList()`. To je hitreje,bolj berljivo in hkrati manj tipkanja:

```
function calculate () {
        var narm = "";        // na.rm=FALSE is the default in all  ↩
            functions below
        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");
        // ...
}
```

Preprosta pomožna funkcija `makeOption()` lahko v mnogih primerih olajša izpuščanje parametrov, ki imajo privzeto vrednost:

```
function calculate () {
        var options
        //...
        // This will do nothing, if VALUE is 0.95 (the default). Otherwise  ↩
            it will append ', conf.int=VALUE' to options.
        options += makeOption ("conf.int", getString ("confint"), "0.95");
        //...
}
```

# Poglavje 6

# Pisanje strani s pomočjo

Ko vaš vtičnik v bistvu deluje, je prišel čas, da zagotovite stran s pomočjo.Čeprav običajno ne boste želeli razložiti vseh temeljnih konceptov vglobino, boste morda želeli dodati še kakšno razlago za nekatere možnosti,in povezavo do povezanih vtičnikov in R funkcije.

> NAMIG
> Ko preberete to poglavje, si oglejte rkwarddev paket tudi. Zagotavlja nekaj R funkcij za ustvarjanje večine RKWard-jevih XML oznake za vas. Prav tako lahko ustvari osnovno okostje datoteke pomoči iz obstoječega vtičnika XML datoteke, s katerimi lahko začnete.

Morda se spomnite, da ste to vstavili v svoj vtičnik XML (če niste dali to notri, naredite to zdaj):

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

Kjer bi očitno zamenjali `ime datoteke` z bolj primernim imenom. Zdaj je čas, da ustvarite to `.rkh`. Tukaj je samoopisni primer:

```
<!DOCTYPE rkhelp>
<document>
        <summary>
In this section, you will put some short, very basic information about what ↩
    the plugin does.
This section will always show up on the very top of the help page.
        </summary>

        <usage>
The usage section may contain a little more practical information. It does  ↩
   not explain all
the settings in detail (that is done in the "settings" section), however.

To start a new paragraph, insert an empty line, as shown above.
This line, in contrast, will be in the same paragraph.

In all sections you can insert some simple HTML code, such as <b>bold</b>  ↩
   or
<i>italic</i> text. Please keep formatting to the minimum needed, however.
```

```
The usage section is always the second section shown in a help page.
        </usage>

        <section id="sectionid" title="Generic section" short_title=" ↩
            Generic">
If you need to, you can add additional sections between the usage and  ↩
    settings sections.
However, usually, you will not need this while documenting plugins. The "id ↩
    "-attribute
provides an anchor point to jump to this section from the navigation menu.  ↩
    The "short_title"
attribute provides a short title to use in the navigation bar. This is  ↩
    optional, by default
the main "title" will be used both as a heading to the section, and as the  ↩
    link name in the
navigation bar.

In any section you may want to insert links to further information. You do  ↩
    this by adding

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

Where URL could be an external link such as https://rkward.kde.org .
Several special URLs are supported in the help pages:

<link href="rkward://page/path/page_id"/>

This links to a top level rkward help page (not for a plugin).

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

This links to the help page of another plugin. The [namespace/] part may be ↩
     omitted
(in this case, rkward is assumed as the standard namespace, e.g.:
<link href="rkward://component/import_spss"/> or
<link href="rkward://component/rkward/import_spss"/> are equivalent).
The component_id is the same that you specified in the .pluginmap.

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

Links to the R help page on "rfunction".

Note that the link names will be generated automatically for these types of ↩
     links.
        </section>

        <settings>
                <caption id="id_of_tab_or_frame"/>
                <setting id="id_of_element">
Description of the GUI element identified by the given id
                </setting>
                <setting id="id_of_elementb" title="description">
Usually the title of the GUI element will be extracted from the
XML definition of the plugin, automatically. However,
for some GUI elements, this description may not be enough to identify them, ↩
     reliably.
In this case, you can add an explicit title using the "title" attribute.
```

```
                </setting>
                <setting id="id_of_elementc">
Description of the GUI element identified by "id_of_elementc"
                </setting>
                [...]
        </settings>

        <related>
The related section typically just contains some links, such as:

<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>
The technical section (optional, always last) may contain some technical details of the plugin
implementation, which are of interest only to RKWard developers. This is particularly relevant
for plugins that are designed to be embedded in many other plugins, and could detail, which
options are available to customize the embedded plugin, and which code sections contain which
R code.
        </technical>
</document>
```

# Poglavje 7

# Logične interakcije med GUI elementi

## 7.1 GUI logic

Vsi osnovni koncepti ustvarjanja vtičnika za RKWard so bili opisani v prejšnjih poglavjih. Ti osnovni koncepti bi morali zadoščati za številne - če ne večino - primerov. Vendar včasih želite več nadzora nad tem, kako se obnaša GUI vašega vtičnika.

Na primer, recimo, da želite razširiti primer t-testa, uporabljen v tej dokumentaciji, da omogočite oboje: primerjavo spremenljivke z drugo spremenljivko (kot je prikazano) in primerjavo spremenljivke s konstantno vrednostjo. Eden od načinov za to bi bil dodati radijsko krmiljenje, ki preklaplja med obema načinoma, in dodajanje vrtilnega polja za vnos konstantne vrednosti za primerjavo. Razmislite o tem poenostavljenem primeru:

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

        <dialog label="T-Test">
                <row>
                        <varselector id="vars"/>
                        <column>
                                <varslot id="x" types="number" source="vars ↩
                                    " required="true" label="compare"/>
                                <radio id="mode" label="Compare against">
                                        <option value="variable" checked=" ↩
                                            true" label="another variable ( ↩
                                            select below)"/>
                                        <option value="constant" label="a  ↩
                                            constant value (set below)"/>
                                </radio>
                                <varslot id="y" types="number" source="vars ↩
                                    " required="true" label="variable"  ↩
                                    i18n_context="Noun; a variable"/>
                                <spinbox id="constant" initial="0" label=" ↩
                                    constant" i18n_context="Noun; a constant ↩
                                    "/>
                        </column>
                </row>
        </dialog>
```

```
</document>
```

Zaenkrat je vse v redu, vendar obstajajo številne težave s tem GUI. Prvič, tako varslot kot spinbox sta vedno prikazana, medtem ko je samo eden od obeh resnično uporabljen. Še huje, varslot vedno zahteva veljavno izbiro, tudi če primerjate s konstanto. Očitno, če ustvarimo večnamenski GUI kot je ta, želimo več prilagodljivosti. Vnesite: razdelek **<logic>** (vstavljen na isti ravni kot **<code>**, **<dialog>** ali **<wizard>**).

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

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

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

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

Prva vrstica znotraj logičnega razdelka je oznaka **<convert>**. V bistvu to zagotavlja novo logično (vklopljeno ali izklopljeno, resnično ali napačno) lastnost, ki jo je mogoče uporabiti pozneje. Ta lastnost (*˜varmode˜*) je resnična, kadar je izbran zgornji izbirni gumb, in napačna, ko je izbran spodnji izbirni gumb. Kako se to naredi?

Najprej so pod *sources* navedene lastnosti vira, na katerih je treba delati (v tem primeru vsaka samo po ena; navedete jih lahko več kot *sources=˜mode.string; somethingelse˜*, potem bi bil *˜varmode˜* resničen le, če sta oba *˜mode.string˜* in *˜somethingelse˜* enaka na niz *˜variable˜*). Upoštevajte, da v tem primeru ne pišemo le *˜mode˜* (kot bi v `getString(˜mode˜)`), ampak *˜mode.string˜*. To je pravzaprav interni način delovanja radijskega krmilnika: ima lastnost 'string', ki hrani svojo vrednost niza. `getString(˜mode˜)` je samo okrajšava in enakovredno `getString(˜mode.string˜)`. Oglejte si referenco za vse lastnosti različnih GUI elementi.

Drugič, način pretvorbe nastavimo na *mode=˜equals˜*. To pomeni, da želimo preveriti, ali je vir(i) enak(i) določeni vrednosti. Končno je standard vrednost za primerjavo, zato s *standard=˜variable˜* preverimo, ali je lastnost *˜mode.string˜* enaka na niz *˜variable˜* (vrednost zgornje radijske možnosti). Če je enako, potem je lastnost varmode resnična, drugače je napačna.

Zdaj pa k pravim stvarem: **<connect>** lastnost *˜varmode˜* povežemo z y.visible, ki nadzira, ali varslot *˜y˜* je prikazan ali ne. Upoštevajte, da kateri koli element, ki je neviden, implicitno ni potreben. Torej, če je izbrana zgornja radijska možnost, je varslot *˜y˜* zahtevan in viden. Sicer ni zahtevana in skrita.
Za spinbox želimo ravno obratno. Na srečo za to ne potrebujemo drugega **<convert>**: logične lastnosti je mogoče zelo enostavno zanikati z dodajanjem modifikatorja *˜not˜*, zato **<connect>** *˜varmode.not˜* z lastnostjo vidnosti spinboxa. V bistvu je bodisi varslot prikazan in zahtevan, *ali* spinbox je prikazan in zahtevan - odvisno od tega, katera možnost je izbrana v radijskem nadzoru. GUI se spreminja glede na možnost radia. Poskusite primer, če želite.

Za celoten seznam lastnosti glejte referenco. Še ena lastnost pa je posebna v tem, da vsi GUI elementi imajo to: 'omogočeno'. To je nekoliko manj drastično kot 'vidno'. Ne prikaže/skrije GUI element, ampak ga samo omogoči/onemogoči. Onemogočeni elementi so običajno prikazani sivo in se ne odzivajo na uporabniški vnos.

> OPOMBA
> Poleg **<convert>** in **<connect>** obstaja več dodatnih elementov za uporabo v razdelku **<logic>**. Npr. pogojne konstrukcije je mogoče implementirati tudi z uporabo elementa **<switch>**. Za podrobnosti glejte referenco o logičnih elementih.

## 7.2 Skriptirana GUI logika

Medtem ko je povezovanje lastnosti, kot je opisano zgoraj, pogosto dovolj, je včasih bolj prilagodljivo ali bolj priročno uporabiti JS za skript GUI logiko. Na ta način bi lahko zgornji primer prepisali kot:

```
[...]
        <code file="code.js"/>
'
        <logic>
                <script><![CDATA[
                        // ECMAScript code in this block
                        // the top-level statement is only called once
                        gui.addChangeCommand ("mode.string", "modeChanged ↩
                            ()");

                        // this function is called whenever the "mode" was ↩
                            changed
                        modeChanged = function () {
                                var varmode = (gui.getString ("mode.string ↩
                                    ") == "variable");
                                gui.setValue ("y.enabled", varmode);
                                gui.setValue ("constant.enabled", !varmode) ↩
                                    ;
                        }
                ]]></script>
        </logic>

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

Prva vrstica kode pove RKWard za klic funkcije `modeChanged()` vsakič, ko se spremeni vrednost izbirnega polja *`id=˜mode˜`*. Znotraj te funkcije definiramo pomožno spremenljivko *`˜varmode˜`*, ki je true, ko je način *`˜variable˜`*, false, ker je *`˜constant˜`*. Nato uporabimo `gui.setValue()` za nastavitev 'omogočenih' lastnosti *`˜y˜`* in *`˜constant˜`*, na povsem enak način, kot smo to storili prej z uporabo stavkov **<connect>**.

Skriptni pristop k GUI logika postane še posebej uporabna, ko želite spremeniti razpoložljivo možnost glede na vrsto predmeta, ki ga je izbral uporabnik. Oglejte si referenco za razpoložljive funkcije.

Upoštevajte, da je skriptni pristop k GUI logiko lahko zmešate s stavki **<connect>** in **<convert>**, če želite. Upoštevajte tudi, da oznaka **<script>** omogoča podajanje imena datoteke skripta poleg ali kot alternativo vstavljanju kode skripta. Običajno pa je najbolj priročna vdelava kode skripta, kot je prikazano zgoraj.

# Poglavje 8

# Vdelava vtičnikov v vtičnike

## 8.1 Primeri uporabe za vdelavo

Ko pišete vtičnike, boste pogosto ugotovili, da ustvarjate več vtičnikov, ki se razlikujejo le v nekaterih pogledih, vendar imajo veliko več skupnega. Na primer, za risanje obstaja več generičnih R možnosti, ki jih je mogoče uporabiti pri skoraj vseh vrstah grafikonov. Če ustvarite GUI in predloga JS za tiste vedno znova?

Očitno bi bilo to precej težav. Na srečo vam tega ni treba storiti. Namesto tega ustvarite skupno funkcionalnost enkrat, kasneje pa jo lahko vdelate v več vtičnikov. Pravzaprav je mogoče kateri koli vtičnik vdelati v kateri koli drug vtičnik, tudi če izvirni avtor vdelanega vtičnika nikoli ni pomislil, da bi nekdo želel svoj vtičnik vdelati v drugega.

## 8.2 Vdelava znotraj pogovornega okna

OK, dovolj povedano. Kako deluje? Preprosto: preprosto uporabite oznako **<embed>**. Tukaj je skrajšan primer:

```
<dialog>
        <tabbook>
                <tab [...]>
                        [...]
                </tab>
                <tab label="Plot Options" i18n_context="Options concerning ↩
                    the plot">
                        <embed id="plotoptions" component="rkward:: ↩
                            plot_options"/>
                </tab>
                <tab [...]>
                        [...]
                </tab>
        </tabbook>
</dialog>
```

Tukaj se zgodi, da celoten GUI ali pa je vtičnik možnosti izrisa (seveda razen standardnih elementov, kot je gumb **Pošlji** itd.) vdelan neposredno v vaš vtičnik (poskusite!).

Kot lahko vidite, je sintaksa oznake **<embed>** dokaj preprosta. Kot večina elementov potrebuje *id*. Komponenta parametra določa, kateri vtičnik je treba vdelati, kot je definirano v mapi `.pluginmap` datoteka (*˜rkward::plot_options˜* je rezultat veriženja imenskega prostora 'rkward', ločila '::' in imena komponente 'plot_options').

## 8.3 Generiranje kode pri vdelavi

Zaenkrat je vse v redu, kaj pa ustvarjena koda? Kako sta koda za vdelavo in vdelani vtičnik združena? V kodo JS vtičnika za vdelavo preprosto napišite nekaj takega:

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

Torej v bistvu pridobivamo kodo, ki jo ustvari vdelani vtičnik, tako kot pridobivamo kateri koli drug GUI nastavitev. Tukaj lahko niz *˜plotoptions.code.printout˜* razčlenimo na: 'Oddelek za izpis ustvarjene kode elementa z *id* plotoptions' (plotoptions je ID, ki smo ga dali za oznako **<embed>** zgoraj). In ja, če želite napreden nadzor, lahko celo pridobite vrednosti posameznih GUI elementov znotraj vdelanega vtičnika (vendar ne obratno, saj vdelani vtičnik ne ve ničesar o svoji okolici).

## 8.4 Vdelava znotraj čarovnika

Če vaš vtičnik ponuja čarovnika GUI, vdelava v osnovi deluje na enak način. Na splošno boste uporabili:

```
<wizard [...]>
                [...]
                <page id="page12">
                        [...]
                </page>
                <embed id="plotoptions" component="rkward::plot_options"/>
                <page id="page13">
                        [...]
                </page>
                [...]
        </wizard>
```

Če vdelani vtičnik ponuja vmesnik čarovnika, bodo njegove strani vstavljene neposredno med *˜page12˜* in *˜page13˜* vašega vtičnika. Če vdelani vtičnik ponuja samo pogovorni vmesnik, bo med vašima stranema *˜page12˜* in *˜page13˜* dodana ena nova stran. Uporabnik ne bo nikoli opazil.

## 8.5 Manj vdelana vdelava: gumb za nadaljnje možnosti

Čeprav je vdelava kul, pazite, da ne pretiravate. Preveč funkcij znotraj GUI le težko je najti ustrezne možnosti. Seveda boste včasih morda želeli vdelati veliko možnosti (kot so vse možnosti za `plot()`), a ker so te res izbirne, jih ne želite izpostaviti v svojem GUI.

Druga možnost je vdelava teh možnosti 'kot gumb':

```
<dialog>
        <tabbook>
                [...]
                <tab label="Options">
```

```
                        [...]
                        <embed id="plotoptions" component="rkward:: ↩
                            plot_options" as_button="true" label="Specify  ↩
                            plotting options"/>
                </tab>
                [...]
        </tabbook>
</dialog>
```

V tem primeru bo vašemu vtičniku dodan en sam gumb z oznako **Določi možnosti izrisa**. Ko pritisnete ta gumb, se prikaže ločeno pogovorno okno z vsemi možnostmi vdelanega vtičnika. Čeprav ta vdelani GUI večino časa ni viden, lahko njegove nastavitve pridobite tako, kot je opisano zgoraj.

> OPOZORILO
> Verjetno bi se moral pristop 'gumba' uporabljati le za vtičnike, ki nikoli ne morejo biti neveljavni (za manjkajoče/slabe nastavitve). Sicer uporabnik ne bi mogel oddati kode, ampak bi jo težko ugotovil, saj se vzrok za to skriva za nekim gumbom.

## 8.6 Vdelava/definiranje nepopolnih vtičnikov

Nekateri vtičniki – in pravzaprav je plot_options, uporabljen kot zgornji primer, eden izmed njih – sami po sebi niso popolni. Preprosto nimajo GUI elementov za izbiro nekaterih pomembnih vrednosti. Namenjeni so samo uporabi kot vdelani v druge vtičnike.

V kolikšni meri je vtičnik plot_options nepopoln? No, za nekatere nastavitve možnosti mora poznati imena predmetov/izrazov za osi x in y (pravzaprav bo v redu, če ima samo eno od obeh, vendar za pravilno delovanje potrebuje vsaj enega). Vendar pa nima mehanizma za izbiro teh predmetov ali vnos na kakršen koli drug način. Kako torej ve zanje?

V logičnem razdelku vtičnika plot_options sta dve dodatni vrstici, ki še nista zajeti:

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

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

To definira dve dodatni lastnosti v vtičniku plot_options, katerih edini namen je povezava z nekaterimi (še neznanimi) lastnostmi vtičnika za vdelavo. V vtičniku plot_options se ti dve lastnosti preprosto uporabljata kot vse druge in na primer obstajajo klici `getString(˝xvar˝)` v predlogi plot_options JS.

Zdaj za nepopoln vtičnik ni mogoče vedeti, kam bo vdelan in kako se bodo imenovale ustrezne nastavitve v vtičniku za vdelavo. Zato moramo dodati tudi dve dodatni vrstici v logičnem razdelku vtičnika za vdelavo:

```
<logic>
                [...]

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

To načeloma ni nič novega, obravnavali smo stavke **<connect>** v poglavju GUI logika. Preprosto povežete vrednosti v dveh varlotih (imenovanih *˜xvarslot˜* in *˜yvarslot˜* v tem primeru) s sprejemnimi 'zunanjimi' lastnostmi vdelanega vključiti. To je to. Za vse ostalo je samodejno poskrbljeno.

# Poglavje 9

# Ukvarjanje s številnimi podobnimi vtičniki

## 9.1 Pregled različnih pristopov

Včasih boste morda želeli razviti vtičnike za vrsto podobnih funkcij. Kot primer upoštevajte distribucijske ploskve. Ti generirajo precej podobno kodo, seveda pa je zaželeno, da so grafični vmesniki med seboj podobni. Končno so lahko veliki deli datotek pomoči enaki. Le nekaj parametrov se razlikuje za vsak vtičnik.

Naiven pristop k temu je razviti en vtičnik, nato pa kopirati in prilepiti celotno vsebino `.js`, `.xml` in `.rkh`, nato spremenite nekaj delov, ki se razlikujejo. Kaj pa, če čez nekaj časa najdete napako v črkovanju, ki je bila kopirana in prilepljena v vse vtičnike? Kaj pa, če želite dodati podporo za novo funkcijo? Ponovno bi morali obiskati vse vtičnike in spremeniti vsakega posebej. Utrujajoč in dolgočasen proces.

Drugi pristop bi bil uporaba vdelave. Vendar v nekaterih primerih to ni primerno za obravnavano težavo, večinoma zato, ker so 'kosi', ki jih lahko vdelate, včasih preveliki, da bi bili uporabni, in postavlja nekatere omejitve na postavitev. Za te primere so koncepti vključno z datotekami `.js`, vključno z `.xml` in delčki so lahko zelo uporabni (vendar si oglejte razmišljanja o tem, kdaj je bolje uporabi vdelavo).

Ena beseda previdnosti, preden začnete z branjem: ti koncepti lahko pomagajo poenostaviti delo s številnimi podobnimi vtičniki in lahko izboljšajo vzdržljivost in berljivost teh vtičnikov. Vendar lahko pretiravanje zlahka privede do nasprotnega učinka. Uporabljajte z nekaj previdnosti.

## 9.2 Uporaba stavka JS include

Eno skriptno datoteko lahko preprosto vključite v druge RKWard vtičnike. Vrednost tega postane takoj očitna, če so nekateri deli vaše kode JS podobni med vtičniki. Te razdelke lahko preprosto definirate v ločeni datoteki `.js` in to vključite v vse datoteke `.js` vtičnika. Na primer kot v:

```
// this is a file called "common_functions.js"

function doCommonStuff () {
        // perhaps fetch some options, etc.
        // ...
        comment ("This is R code you want in several different plugins\n");
        // ...
}
```

```
// this is one of your regular plugin .js files

// include the common functions
include ("common_functions.js");

function calculate () {
        // do something
        // ...

        // insert the common code
        doCommonStuff ();
}
```

Upoštevajte, da je včasih še bolj uporabno to obrniti in definirati 'skeleton' `preprocess()`, `calculate()` in `funkcije printout()` je običajna datoteka in naredi te povratne klice za tiste dele, ki se med vtičniki razlikujejo. Npr.:

```
// this is a file called "common_functions.js"

function calculate () {
        // do some things which are the same in all plugins
        // ...

        // add in something that is different across plugins
        getSpecifics ();

        // ...
}
```

```
// this is one of your regular plugin .js files

// include the common functions
include ("common_functions.js");

// note: no calculate() function is defined in here.
// it in the common_functions.js, instead.

function getSpecifics () {
        // print some R code
}
```

Ena težava, ki se je morate zavedati pri uporabi te tehnike, je obseg spremenljivk. Oglejte si priročnik JS o obsegih spremenljivk.

Ta tehnika se pogosto uporablja v vtičnikih za distribucijo CLT plot, zato boste morda želeli tam pogledati primere.

## 9.3 Vključno z datotekami `.xml`

V bistvu je ista funkcija vključevanja datotek na voljo tudi za uporabo v `.xml`, `.pluginmap` in datoteke `.rkh`. Na katero koli mesto v teh datotekah lahko postavite oznako **<include>**, kot je prikazano spodaj. Učinek je, da celotna vsebina tega XML datoteka (če smo natančni: vse v oznaki **<document>** te datoteke) je vključeno dobesedno na tej točki v datoteki. Upoštevajte, da lahko vključite samo drugo datotekoXML.

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

Atribut *file* je ime datoteke glede na imenik, v katerem se nahaja trenutna datoteka.

## 9.4 Uporaba <snippets>

Medtem ko je vključevanje datotek, kot je prikazano v prejšnjem razdelku, precej zmogljivo, postane najbolj uporabno, če se uporablja v kombinaciji z **<snippets>**. Izrezki so res manjši odseki, ki jih lahko vstavite na drugo mesto v datoteki. To najbolje ponazarja primer:

```
<document>
        <snippets>
                <snippet id="note">
                        <frame>
                                <text>
                This will be inserted at two places in the GUI
                                </text>
                        </frame>
                </snippet>
        </snippets>
        <dialog label="test">
                <column>
                        <insert snippet="note"/>
                        [...]
                        <insert snippet="note"/>
                </column>
        </dialog>
</document>
```

Zato delček definirate na enem mestu na vrhu datoteke XML datoteko in jo nato **<insert>** na poljubnem mestu(-ih).

Čeprav ta primer sam po sebi ni preveč uporaben, razmislite o kombinaciji z datoteko **<include>**d `.xml`. Upoštevajte, da lahko v isto datoteko postavite tudi delčke za datoteko `.rkh`. Tja bi preprosto **<include>** tudi datoteko in **<insert>** ustrezni snippet (delček):

```
<!-- This is a file called "common_snippets.xml" -->
<document>
        <snippet id="common_options">
                <spinbox id="something" [...]/>
                [...]
        </snippet>
        <snippet id="common_note">
                <text>An important note for this type of plugin</text>
        </snippet>

        <snippet id="common_help">
                <setting id="something">This does something</setting>
                [...]
        </snippet>
</document>
```

```
<!-- This is the .xml file of the plugin -->
<document>
        <snippets>
                <!-- Import the common snippets -->
                <include file="common_snippets.xml"/>
        </snippets>

        <dialog label="test2">
                <insert snippet="common_note"/>
                <spinbox id="something_plugin_specific" [...] />
                <insert snippet="common_options"/>
        </dialog>
</document>
```

Podobno kot vključitev v JS je obratni pristop pogosto celo bolj uporaben:

```
<!-- This is a file called "common_layout.xml" -->
<document>
        <column>
                <insert snippet="note">
                [...]
                <insert snippet="plugin_parameters">
        </column>
        [...]
</document>
```

```
<!-- This is the .xml file of the plugin -->
<document>
        <snippets>
                <snippet id="note">
                        <text>The note used for this specific plugin</text>
                </snippet>

                <snippet id="plugin_parameters">
                        <frame label="Parameters specific to this plugin">
                                [...]
                        </frame>
                </snippet>
        </snippets>

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

Končno je možno tudi delčke **<insert>** v druge delčke, če: a) obstaja samo ena raven gnezdenja in b) ukaz **<snippets>** razdelek je postavljen na vrh datoteke (preden se vstavi ugnezdeni delček); to je zato, ker se stavki **<insert>** razrešijo od zgoraj navzdol.

## 9.5 <include> in <snippets> v primerjavi z <embed>

Na prvi pogled **<include>** in **<snippets>** nudita funkcionalnost, ki je precej podobna vdelavi: omogoča ponovno uporabo nekatere dele kode v vtičnikih. Kakšna je torej razlika med temi pristopi in kdaj bi morali uporabiti katerega?

Ključna razlika med tema pojmoma je, da so vgradljivi vtičniki bolj tesen sveženj. Združujejo celotno kodo GUI za ustvarjanje R kodo iz tega in stran s pomočjo. Nasprotno pa vključitev in vstavljanje omogočata veliko bolj natančen nadzor, vendar za ceno manjše modularnosti.

To pomeni, da vtičnik, ki vgrajuje drug vtičnik, običajno ne potrebuje veliko informacij o notranjih podrobnostih vdelanega vtičnika. Glavni primer je vtičnik plot_options. Vtičnikom, ki želijo to vdelati, ni nujno, da poznajo vse ponujene možnosti ali kako so na voljo. To je dobra stvar, saj bo sicer zaradi spremembe vtičnika plot_options morda treba prilagoditi vse vtičnike, ki to vdelujejo (veliko). Nasprotno pa vključi in vstavi resnično razkrije vse notranje podrobnosti in vtičniki, ki to uporabljajo, bodo na primer morali poznati natančne ID-je in morda celo vrsto uporabljenih elementov.

Zato velja pravilo naslednje: vključi in vstavi sta odlična, če sta ustrezni možnosti potrebni le za jasno omejeno skupino vtičnikov. Vgrajeni vtičniki so boljši, če skupina vtičnikov, za katere bi lahko bili koristni, ni jasno definirana in če je funkcionalnost mogoče enostavno modularizirati. Še eno pravilo: Če lahko skupne dele postavite v en sam 'kos', potem to storite in uporabite vdelavo. Če potrebujete veliko majhnih izrezkov za definiranje skupnih delov -- no, uporabite **<snippets>**. Končni pogled na to: če vsi vtičniki zagotavljajo *zelo* podobno funkcionalnost, sta vključitev in vstavljanje verjetno dobra ideja. Če si delita le enega ali dva skupna 'modula', je vdelava verjetno boljša.

# Poglavje 10

# Koncepti za uporabo v specializiranih vtičnikih

To poglavje vsebuje informacije o nekaterih temah, ki so uporabne samo za določene razrede vtičnikov.

## 10.1 Vtičniki, ki ustvarijo grafikon

Ustvarjanje grafikona iz vtičnika je preprosto. Vendar pa obstaja nekaj subtilnih težav, ki se jim morate izogniti, in tudi nekaj odličnih splošnih funkcij, ki se jih morate zavedati. Ta razdelek prikazuje osnovne koncepte in se zaključi s kanoničnim primerom, ki ga morate upoštevati pri ustvarjanju vtičnikov za grafikon.

### 10.1.1 Risanje grafikona v izhodno okno

Če želite narisati grafikon v izhodno okno, uporabite `rk.graph.on()` neposredno pred ustvarjanjem grafa in `rk.graph.off()` neposredno zatem . To je podobno kot npr. klic `postscript()` in `dev.off()` v navadni R seji.

Pomembno pa je, da morate *vedno* poklicati `rk.graph.off()` po klicu `rk.graph.on()`. V nasprotnem primeru bo izhodna datoteka ostala v pokvarjenem stanju. Če želite zagotoviti, da bo `rk.graph.off()` res klican, bi morali zaviti *vse* R ukaze med obema klicema v stavku `try()`. Še nikoli slišali za to? Brez skrbi, enostavno je. Vse kar morate storiti je, da sledite vzorcu, prikazanemu v primeru spodaj.

### 10.1.2 Dodajanje funkcije predogleda

> OPOMBA
> Ta razdelek obravnava dodajanje funkcije predogleda vtičnikom, ki ustvarjajo grafikone. Obstajajo ločeni razdelki o predogledih (HTML) izpisa, predogledih (uvoženih) podatkov in predogledi po meri. Vendar je priporočljivo, da najprej preberete ta razdelek, saj je pristop v vsakem primeru podoben.

Zelo uporabna funkcija za vse vtičnike, ki ustvarjajo risbo/graf, je zagotoviti samodejno posodabljanje predogleda. Če želite to narediti, boste potrebovali dve stvari: dodati potrditveno polje **<preview>** v GUI definicija in prilagajanje generirane kode za predogled.

Dodajanje potrditvenega polja **<preview>** je preprosto. Preprosto postavite naslednje nekam v GUI. Poskrbel bo za vso čarovnijo v zakulisju ustvarjanja naprave za predogled, posodabljanja predogleda vsakič, ko se spremenijo nastavitve itd. Primer:

> OPOMBA
> Začenši z različico 0.6.5 RKWard so elementi predogleda **<preview>** v pogovornih oknih vtičnikov (ne čarovnikov) označeni s posebnimi črkami: postavljeni bodo v stolpec z gumbi, ne glede na to, kje točno so definirani v uporabniškem vmesniku. Še vedno je dobra ideja, da jih definirate na smiselnem mestu v postavitvi zaradi združljivosti za nazaj.

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

In to je to za GUI definicija.

Prilagajanje predloge JS je le malo več dela, tukaj se boste morali prepričati, da je ustvarjen samo sam grafikon in prikazan v napravi na zaslonu, namesto da bi bil usmerjen na izhod. tj. brez tiskanja glav, `rk.graphics.on()` ali podobnih klicev. Da bi vam pri tem pomagal, RKWard bo poklical funkcije `preprocess()`, `calculate()` in `printout()` z dodatnim parametrom, ki je nastavljen na *`true`* pri generiranju kode za predogled. (Pri generiranju končne kode je parameter izpuščen. V javascriptu bo to ovrednoteno kot *`false`*, če bo uporabljeno znotraj stavka `if`.) Glejte primer spodaj za tipičen vzorec, ki ga boste uporabili.

Če pa potrebujete več nadzora, lahko namesto tega dodate novo funkcijo, imenovano `preview()`, v svojo predlogo JS in tam ustvarite kodo, potrebno za predogled (verjetno vsaj delno, spet s klicem `calculate()` itd.).

### 10.1.3 Splošne možnosti grafikona

Opazili ste, da večina vtičnikov za risanje v RKWard ponuja širok nabor generičnih možnosti itd. za prilagajanje naslovov osi ali robov figur. Dodajanje teh možnosti v vaš vtičnik je enostavno. Zagotavlja jih vtičnik embeddable, imenovan **rkward::plot_options**. To vdelajte v uporabniški vmesnik vtičnika tako:

```
<document>
                [...]
                <logic [...]>
                        <connect client="plotoptions.xvar" governor="x. ↩
                            available"/>
                        <set id="plotoptions.allow_type" to="true"/>
                        <set id="plotoptions.allow_ylim" to="true"/>
                        <set id="plotoptions.allow_xlim" to="false"/>
                        <set id="plotoptions.allow_log" to="false"/>
                        <set id="plotoptions.allow_grid" to="true"/>
                </logic>
                <dialog [...]>
```

```
                        [...]
                        <embed id="plotoptions" component="rkward:: ↩
                            plot_options" as_button="true" label="Plot  ↩
                            Options"/>
                        [...]
                </dialog>
                [...]
        </document>
```

S tem boste v uporabniški vmesnik dodali gumb za prikaz okna z možnostmi grafikona.Logični del je samo primer. Omogoča vam določen nadzor nad vtičnik možnosti risanja. Preberite več na strani s pomočjo vtičnika plot_options (povezava na strani s pomočjo katerega koli vtičnika, ki ponuja splošne možnosti).

Nato se morate prepričati, da koda, ki ustreza vašim možnostim grafikona, doda ustvarjeno kodo za vaš grafikon. Če želite to narediti, pridobite lastnosti**code.preprocess**, **code.printout** in**code.calculate** iz vdelanega vtičnika možnosti izrisa invstavite jih v svojo kodo, kot je prikazano v primer spodaj.

### 10.1.4 Kanonični primer

Tukaj je primer datoteke .JS, ki bi jo morali uporabiti kot predlogo, kadar koli ustvarite vtičnik za risanje:

```
function preprocess () {
    // the "somepackage" is needed to create the plot
    echo ("require (somepackage)\n");
  }

  function printout (is_preview) {
    // If "is_preview" is set to false/undefined, it generates the full  ↩
        code, including headers.
    // If "is_preview" is set to true, only the essentials will be  ↩
        generated.

    if (!is_preview) {
      echo ('rk.header (' + i18n ("An example plot") + ')\n\n');
      echo ('rk.graph.on ()\n');
    }
    // only the following section will be generated for is_preview==true

    // remember: everything between rk.graph.on() and rk.graph.off() should ↩
         be wrapped inside a try() statement:
    echo ('try ({\n');
    // insert any option-setting code that should be run before the actual  ↩
        plotting commands.
    // The code itself is provided by the embedded plot options plugin.  ↩
        printIndentedUnlessEmpty() takes care of pretty formatting.
    printIndentedUnlessEmpty ('\t', getString ("plotoptions.code.preprocess ↩
        "), '', '\n');

    // create the actual plot. plotoptions.code.printout provides the part  ↩
        of the generic plot options
    // that have to be added to the plotting call, itself.
    echo ('plot (5, 5' + getString ("plotoptions.code.printout") + ')\n');

    // insert any option-setting code that should be run after the actual  ↩
        plot.
```

```
    printIndentedUnlessEmpty ('\t', getString ("plotoptions.code.calculate ↩
        "), '\n');
    echo ('})'\n);  // the closure of the try() statement

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

## 10.2 Predogled podatkov, izpisov in drugih rezultatov

### 10.2.1 Predogled (HTML) izhoda

> OPOMBA
> Ta razdelek obravnava dodajanje funkcije predogleda vtičnikom, ki ustvarjajo izpise/izpise HTML. Priporočamo, da pred tem razdelkom preberete ločen razdelek o predogledih ploskev.

Ustvarjanje predogleda izpisa HTML je skoraj enak postopek kot ustvarjanje predogleda grafikona. V tem primeru se preprosto prepričajte, da **preview()** ustvari ustrezne ukaze **rk.print()/rk.results()**. Vendar pa je na splošno dobro izpustiti izjave glave v predogledu. Tukaj je skrajšan primer:

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

Upoštevajte specifikacijo *mode="output"* v elementu **<preview>**.

```
// In the plugin's JS file
        function preview () {
                // generates the code used for preview
                printout (true);
        }

        function printout (is_preview) {
                // only generates a header if is_preview==false
                if (!is_preview) {
                        new Header ("This is a caption").print ();
                }
                echo ('rk.print (result)');
        }
```

### 10.2.2 Predogledi (uvoženih) podatkov

> OPOMBA
> Ta razdelek obravnava dodajanje funkcije predogleda vtičnikom, ki ustvarjajo (uvažajo) podatke. Priporočamo, da pred tem razdelkom preberete ločen razdelek o predogledih grafikonov.

Ustvarjanje predogleda uvoženih podatkov (katere koli vrste podatkov, ki jih lahko obdeluje **rk.edit()**) je zelo podobno ustvarjanju predogleda grafikona. Naslednji skrajšani primer bi moral pomagati ponazoriti, kako ustvariti predogled podatkov:

```
<!-- In the plugin's XML file -->>
        <dialog label="Import CSV data" >
                <browser id="file" type="file" label="File name"/>
                <!-- [...] -->>
                <preview id="preview" active="true" mode="data"/>
        </dialog>
>
```

Upoštevajte, da element **<preview>** tokrat določa *mode="data"*. *active="true"* preprosto naredi predogled privzeto aktiven.

```
// In the plugin's JS file
        function preview () {
                // generates the code used for preview
                calculate (true);
        }

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

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

Spet funkcija **preview()** ustvari skoraj enakoR kodo kot funkcijo **calculate()**, tako da ustvarimo apomožno funkcijo **doCalcuate()** za izločanje skupnih delov. Najpomembnejša stvar, ki jo morate upoštevati, je, da boste morali dodeliti uvožene podatke v objekt, imenovan *preview_data* (znotraj trenutnega – lokalnega – okolja). *Vse ostalo se bo zgodilo samodejno* (grobo rečeno, bo RKWard poklical **rk.edit(preview_data)**, zavit v klic **.rk.with.window.hints()**).

> OPOMBA
> Čeprav so predogledi odlična funkcija, porabljajo vire. V primeru predogledov podatkov, lahko pride do primerov, ko lahko predogledi povzročijo znatne težave z zmogljivostjo. To je lahko za uvoz ogromnih naborov podatkov (ki so preprosto preveliki, da bi ga odprli za urejanje v oknu urejevalnika RKWard), vendar tudi "običajni" nabori podatkov bi lahko bili napačno uvoženi, kar bi povzročilo ogromno število vrstic ali stolpcev. *Zelo priporočljivo je, da omejite preview_data* v dimenzijo, ki zagotavlja uporaben predogled, brez nevarnosti ustvarjanja opazne zmogljivostiteže (npr. 50 vrstic s 50 stolpci bi moralo biti v večini primerov več kot dovolj).

### 10.2.3 Predogledi po meri

Ukaz **<preview>** element lahko uporabite za ustvarjanjepredogledov za katero koli vrsto okna ″dokumenta″, ki ga je mogoče pripeti na RKWardnjegovo delovno mesto. Poleg grafovin podatkovnih oken, to vključuje HTMLdatoteke, R skripte in okna s povzetkom objektov. Za slednje tibo moral uporabiti **<preview mode=″custom″>**.

Če ste prebrali razdelke, ki opisujejo predogled izrisa in predogled podatkov, stebi morali imeti splošno predstavo o postopku, vendar so potrebni predogledi po merinekoliko več ročnega dela v zakulisju. Najpomembnejša R funkcija za ogled je **rk.assign.preview.data()**, tukaj. Naslednji kratek seznam prikazuje, kaj ste ustvarili (predogled) R koda bi lahko videti kot za vtičnik, ki ustvarja izhod besedilne datoteke:

```
## To be generated in the preview() code section of a plugin
        pdata <- rk.get.preview.data("SOMEID")
        if (is.null (pdata)) {
                outfile <- rk.get.tempfile.name(prefix="preview", extension ↩
                    =".txt")
                pdata <- list(filename=outfile, on.delete=function (id) {
                        unlink(rk.get.preview.data(id)$filename)
                })
                rk.assign.preview.data("SOMEID", pdata)
        }
        try ({
                cat ("This is a test", pdata$filename)
                rk.edit.files(file=pdata$filename)
        })
```

Tukaj bi morali pridobiti vrednost *SOMEID* iz lastnosti *id* elementa **<preview>**. tj. z uporabo **getString (″preview.id″)** v datoteki .js vtičnika.

## 10.3 Kontekstno odvisni vtičniki

Doslej smo domnevali, da so vsi vtičniki vedno smiselni in vsi nameščeni v glavnem meniju. Vendar so nekateri vtičniki smiselni samo (ali dodatno) v določenem kontekstu. Na primer vtičnik za izvoz vsebine datoteke R Grafična naprava X11 je očitno najbolj uporabna, če jo postavite v meni naprave X11, ne v glavno menijsko vrstico. Prav tako mora tak vtičnik vedeti za številko naprave, na kateri naj deluje, ne da bi o tem vprašal uporabnika.

Takim vtičnikom pravimo, da so odvisni od konteksta. Skladno s tem v `.pluginmap` datoteko, niso (ali ne samo) postavljeni v glavno **<hierarchy>**, temveč v element **<context>**. Zaenkrat sta podprta samo dva različna konteksta (več jih bo na voljo pozneje): x11 in uvoz datotek. Te bomo obravnavali po vrsti. Tudi če vas zanima le uvozni kontekst, preberite tudi razdelek o kontekstu x11, saj je nekoliko bolj dodelan.

### 10.3.1 Kontekst naprave X11

Če želite uporabiti vtičnik v kontekstu naprave x11 – to pomeni, da ga postavite v menijsko vrstico okna, ki se prikaže, ko pokličete `x11()` v konzoli, ga najprej deklarirajte kot običajno v `.pluginmap` datoteki:

```
<document [...]>
        <components>
                [...]
                <component id="my_x11_plugin" file="my_x11_plugin.xml"  ↩
                    label="An X11 context plugin"/>
```

```
                [...]
        </components>
```

Vendar vam ga ni treba definirati v hierarhiji (lahko, če jesmiseln tudi kot vtičnik najvišje ravni):

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

Namesto tega dodajte definicijo konteksta ″x11″ in jo dodajte v menije tja:

```
<context id="x11">
                [...]
                <menu id="edit">
                        [...]
                        <entry id="my_x11_plugin"/>
                </menu>
        </context>
</document>
```

V logičnem razdelku vtičnika xml lahko zdaj deklarirate dve lastnosti **<external>**: *devnum* in *context*. *context* (če je deklariran) bo nastavljen na *»x11«*, ko bo vtičnik priklican v tem kontekstu. *devnum* bo nastavljen na številko grafične naprave za delovanje. In to je vse.

### 10.3.2 Uvoz konteksta podatkov

Preden preberete ta razdelek, obvezno preberite razdelek o kontekstu naprave X11, saj pojasnjuje osnovne pojme.

Kontekst *»import«* se uporablja za deklaracijo vtičnikov filtrov za uvoz datotek. Preprosto jih postavite v kontekst z *id=″import″* v datoteko .pluginmap. Vendar pa je pri deklaraciji teh vtičnikov še en dodaten zasuk: če želite ponuditi poenoteno pogovorno okno za izbiro datotek za vse podprte vrste datotek, morate deklarirati en dodaten bit informacij o svoji komponenti:

```
<document [...]>
        <components>
                [...]
                <component id="my_xyz_import_plugin" file=" ↩
                    my_xyz_import_plugin.xml" label="Import XYZ files">
                        <attribute id="format" value="*.xyz *.zyx" label=" ↩
                            XYZ data files"/>
                </component>
                [...]
        </components>
        <hierarchy>
                [...]
        </hierarchy>
        <context id="import">
                [...]
                <menu id="import">
                        [...]
                        <entry id="my_xyz_import_plugin"/>
                </menu>
        </context>
        [...]
</document>
```

Vrstica atributa preprosto pravi, da so povezane pripone imena datotek za datoteke XYZ `*.xyz` ali `*.zyx` in da filter naj bo v pogovornem oknu za izbiro datoteke označen z 'Podatkovne datoteke XYZ'.

V svojem vtičniku lahko deklarirate dve lastnosti **<external>**. *`filename`* bo nastavljen na izbrano ime datoteke, *`context`* pa bo nastavljen na *`~import~`*.

## 10.4 Poizvedovanje R za informacijo

V nekaterih primerih boste morda želeli pridobiti dodatne informacije iz R, ki bodo predstavljene v uporabniškem vmesniku vašega vtičnika. Na primer, morda boste želeli ponuditi izbor ravni faktorja, ki ga je uporabnik izbral za analizo. Od različice 0.6.2 RKWard to je mogoče storiti. Preden začnemo, je pomembno, da se zavedate nekaterih opozoril:

Koda R, ki se izvaja znotraj logike uporabniškega vmesnika vtičnika, se ovrednoti v zanki dogodkov R, kar pomeni, da se lahko izvaja *medtem, ko* se izvajajo drugi izračuni. S tem zagotovimo, da bo uporabniški vmesnik vašega vtičnika uporaben, tudi ko R je zaposlen z drugimi stvarmi. Vendar je zaradi tega zelo pomembno, da vaša koda nima stranskih učinkov. Še posebej:

- *Ne* izvaja nobenih dodelitev v .GlobalEnv ali katerem koli drugem nelokalnem okolju.
- *Ne* natisne ničesar v izhodno datoteko.
- *Ne* riše ničesar na zaslon.
- Na splošno *ne* počne ničesar, kar ima stranske učinke. Vaša koda lahko *bere informacije*, ne pa *"naredi"* česar koli.

S tem v mislih je tukaj splošni vzorec. To boste uporabili v razdelku skriptirane logike uporabniškega vmesnika:

```
<script><![CDATA[
                                last_command_id = -1;
                                gui.addChangeCommand ("variable", "update ↩
                                    ()");
                                update = function () {
                                        gui.setValue ("selector.enabled", ↩
                                            0);
                                        variable = gui.getValue ("variable ↩
                                            ");
                                        if (variable == "") return;

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

                                commandFinished = function (result, id) {
                                        if (id != last_command_id) return; ↩
                                             // another result is about to ↩
                                            arrive
                                        if (typeof (result) == "undefined") ↩
                                             {
                                                gui.setListValue ("selector ↩
                                                    .available", Array (" ↩
                                                    ERROR"));
                                                return;
                                        }
```

```
                                    gui.setValue ("selector.enabled",  ↩
                                        1);
                                    gui.setListValue ("selector. ↩
                                        available", result);
                                }
        ]]></script>
```

Tu je *variable* lastnost, ki vsebuje ime predmeta (npr. znotraj **<varslot>**). Kadarkoli se to spremeni, boste želeli posodobiti prikaz ravni znotraj **<valueselector>**, imenovanega *selector*. Ključna funkcija tukaj je **doRCommand()**, ki kot prvi parameter vzame ukazni niz, ki ga je treba zagnati, in kot drugi parameter ime funkcije, ki jo je treba poklicati, ko je ukaz končan. Upoštevajte, da se ukaz izvaja asinhrono, zaradi česar so stvari nekoliko bolj zapletene. Kot eno stvar se želite prepričati, da vaš **<valueselector>** ostane onemogočen, medtem ko ne vsebuje posodobljenih informacij. Druga stvar je, da ste morda v čakalno vrsto postavili več kot en ukaz, preden dobite prve rezultate. Zato je vsakemu ukazu dodeljen "id", ki ga shranimo v *last_command_id* za kasnejšo uporabo.

Ko je ukaz končan, se prikliče navedeni povratni klic (*commandFinished*, v tem primeru) z dvema parametroma: samim rezultatom in ID-jem ustreznega ukaza. Rezultat bo vrste, ki spominja na predstavitev v R, tj. številski niz, če je rezultat številski, itd. Lahko je celo R **list()**, vendar bo v tem primeru predstavljen kot JS **Array()** brez imen.

Upoštevajte, da je tudi ta primer nekoliko poenostavljen. V resnici bi morali sprejeti dodatne previdnostne ukrepe, itd., da se izognete vstavljanju prevelikega števila stopenj v izbirnik. Dobra novica je, da vam verjetno ni treba storiti vsega tega sami. Zgornji primer je na primer vzet iz vtičnika **rkward::level_select**, ki ga lahko preprosto vdelate v svoj vtičnik. To vam celo omogoča, da določite drugačen izraz za izvajanje namesto **levels()**.

## 10.5 Sklicevanje na trenutni predmet ali trenutno datoteko

Za mnoge vtičnike je zaželeno, da delajo na 'trenutnem' objektu. Na primer, vtičnik 'sort' lahko vnaprej izbere data.frame, ki se trenutno ureja za razvrščanje. Ime trenutnega predmeta je na voljo vtičnikom kot vnaprej določena lastnost, imenovana *current_object*. S to nepremičnino se lahko povežete na običajen način. Če noben objekt ni trenuten, je lastnost enaka praznemu nizu. Podobno URL trenutne datoteke skripta je dostopen kot vnaprej določena lastnost, imenovana *current_filename*. Ta lastnost je prazna, če se trenutno ne ureja nobena skriptna datoteka ali če skriptna datoteka še ni bila shranjena.

Trenutno je lahko *current_object* samo razreda `data.frame`, vendar se ne zanašajte na to, saj bo to v prihodnosti razširjeno na druge vrste podatkov. Če vas zanimajo samo objekti `data.frame`, se raje povežite z lastnostjo *current_dataframe*. Zahteve glede vrste lahko uveljavite tudi z uporabo ustreznih omejitev na vaših **<varslot>** ali z uporabo GUI logično skriptiranje.

## 10.6 Ponavljanje (niz) možnosti

Včasih želite ponoviti nabor možnosti za poljubno število elementov. Npr. Recimo, da želite implementirati vtičnik za razvrščanje data.frame. Morda boste želeli dovoliti razvrščanje po poljubnem številu stolpcev (v primeru izenačenja med prvimi stolpci). To bi lahko preprosto realizirali tako, da bi uporabniku omogočili izbiro več spremenljivk v **<varslot>** s *multi="true"*. Če pa želite to razširiti, npr. omogoča uporabniku, da za vsako spremenljivko določi, ali naj se pretvori v znakovno/številsko ali naj bo razvrščanje naraščajoče ali padajoče, potrebujete več prilagodljivosti. Drugi primeri bi bili risanje več črt v enem izrisu (ki omogoča izbiro predmeta, sloga črte, barve črte itd. za vsako črto) ali določanje preslikave za ponovno kodiranje iz niza starih vrednosti v nove vrednosti.

Vnesite **<optionset>**. Najprej si poglejmo preprost primer:

```
<dialog [...]>
        [...]
        <optionset id="set" min_rows="1">
                <content>
                        <row>
                                <input id="firstname" label="Given name(s)" ↩
                                     size="small">
                                <input id="lastname" label="Family name"  ↩
                                    size="small">
                                <radio id="gender" label="Gender">
                                        <optioncolumn label="Male" value="m ↩
                                            "/>
                                        <optioncolumn label="Female" value ↩
                                            ="f"/>
                                </radio>
                        </row>
                </content>

                <optioncolumn id="firstnames" label="Given name(s)" connect ↩
                    ="firstname.text">
                <optioncolumn id="lastnames" label="Family name" connect=" ↩
                    lastname.text">
                <optioncolumn id="gender" connect="gender.string">
        </optionset>
        [...]
</dialog>
```

Tukaj smo ustvarili uporabniški vmesnik za določanje števila oseb (npr. avtorjev). Uporabniški vmesnik zahteva vsaj en vnos (*min_rows="1"*). Znotraj elementa **<optionset>** začnemo z določitvijo **<content>**, tj. tiste elemente, ki pripadajo naboru možnosti. Seznanjeni boste z večino elementov znotraj **<content>**.

Nato določimo zanimive spremenljivke, ki jih bomo želeli prebrati iz nabora možnosti v naši datoteki JS. Ker bomo imeli opravka s poljubnim številom postavk, ne moremo samo brati `getString ("ime")` v JS. Namesto tega za vsako vrednost, ki nas zanima, določimo **<optioncolumn>**. Za prvi stolpec možnosti v primeru **<connect="firstname.text">** pomeni, da se vsebina elementa **<input>** bere za vsak predmet. **<optioncolumn>**, za katere je podana *label*, bodo prikazani na zaslonu v stolpcu s to oznako. V JS lahko zdaj pridobimo imena za vse avtorje z uporabo `getList("set.firstname"),getList("set.lastnames")` za družinska imena, in `getList("set.gender")` za matriko nizov "m"/"f".

Upoštevajte, da ni nobenih omejitev glede tega, kaj lahko postavite znotraj **<optionset>**. Uporabite lahko celo vdelane komponente. Tako kot pri katerem koli drugem elementu je vse, kar morate storiti, zbrati izhodne spremenljivke, ki vas zanimajo, v specifikaciji **<optioncolumn>**. V primeru vdelanih vtičnikov je to pogosto del lastnosti "code". Npr.:

```
<dialog [...]>
        [...]
        <optionset id="set" min_rows="1">
                <content>
                        [...]
                        <embed id="color" component="rkward::color_chooser" ↩
                             label="Color"/>
                </content>

                [...]
                <optioncolumn id="color_params" connect="color.code. ↩
                    printout">
        </optionset>
```

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

Seveda lahko uporabite tudi logiko uporabniškega vmesnika znotrajnabora možnosti. Obstajata dve možnosti za to: To lahko storite tako, da nareditepovezavo (ali skript) v glavnem **<logic>**razdelku vtičnika, kot običajno. Vendar boste dostopali do elementov uporabniškega vmesnikav območju vsebine kot (npr.) ″set.contents.firstname.XYZ″. Upoštevajtepredpono ″set″ (*id*, ki ste ga dodelili nizuin ″vsebina″). Lahko pa dodate ločeno **<logic>** kot podrejeni element vašega nabora **<optionset>**. V tem primeru bodo *id* bodo naslovljeni glede na področje vsebine, npr. ″firstname.XYZ″. Samo element **<script>** ni dovoljeno v logičnem delu nabora možnosti. Če želite uporabiti skripte, boste morali uporabiti razdelek glavne **<logike>** vtičnika.

> OPOMBA
> Pri skriptni logiki v naboru možnosti je vse, kar lahko storite, dostop do trenutne *current* regije vsebine. Tako je običajno samosmiselno povezati elemente znotraj območja vsebine med seboj.Povezovanje lastnosti zunaj **<optionset>** aznotraj področja vsebine, je lahko uporabna za inicializacijo.Vendar bo spreminjanje območja vsebine po inicializaciji*ne* velja za postavke, ki jih je uporabnik že definiral.Samo na trenutno izbrani artikel v zbirki.

### 10.6.1 Poganjane zbirke možnosti

Doslej smo obravnavali **<optionset>**, kiponuja gumbe za dodajanje/odstranjevanje elementov. Vendar v nekaterih primerih jeveliko bolj naravno izbiranje elementov zunaj **<optionset>** in nudijo le možnosti za prilagajanje nekaterih vidikov vsakega elementav **<optionset>**. Npr. recimo, da želite dovolitiuporabnik izriše več objektov znotraj ene ploskve. Za vsak predmet uporabnikmora imeti možnost določiti barvo črte. To bi *lahko* rešili tako, da postavite **<varselector>** in **<varslot>** znotraj **<content>** območja, ki uporabniku omogoča izbiro enega elementa naenkrat. Vendar pa bi pomenilo veliko manj klikov za uporabnika, če uporabite **<varslotmulti=″true″>** *zunaj*Namesto tega **<optionset>**. Potem boste to povezaliizbiranje predmetov v tako imenovanem ″pognanem″ naboru možnosti. Takole:

```
<dialog [...]>
        <logic>
                <connect client="set.vars" governor="vars.available"/>
                <connect client="set.varnames" governor="vars.available. ↩
                    shortname"/>
        </logic>
        [...]
        <varselector id="varsel"/>
        <varslot id="vars" label="Objects to plot"/>
        <optionset id="set" keycolumn="var">
                <content>
                        [...]
                        <embed id="color" component="rkward::color_chooser" ↩
                            label="Line color"/>
                </content>

                [...]
                <optioncolumn id="vars" external="true">
                <optioncolumn id="varnames" external="true" label="Variable ↩
                    ">
                <optioncolumn id="color_params" connect="color.code. ↩
                    printout">
        </optionset>
        [...]
</dialog>
```

Začeli bomo gledati primer na dnu. Opazili boste, da dvaSpecifikacije **<optioncolumn>** imajo*external="true"*. To pove RKWard da so te možnosti **<optionset>** nadzorovane od zunaj. Tukaj je edini namen stolpca z možnostmi "varnames" zagotoviti enostavno branje oznake na zaslonu nabora možnosti (povezan je s "kratkim imenom" modifikatorjem lastnosti, ki vsebuje izbrane predmete). Namen"vars"-optioncolumn naj služi kot "ključni" stolpec, kot določa**<optionset keycolumn="vars"...>**. To pomeni da bo za vsak vnos na tem seznamu nabor ponudil en nabor možnosti in možnosti so logično povezane s temi vnosi. Ta stolpec je povezan z lastnostjo, ki vsebuje izbrane predmete v **<varslot>**. To je za vsak predmet, ki je tam izbran,**<optionset>** omogoča določitev barve črte.

OPOMBA

Zunanji stolpec je mogoče tudi *povezati*z lastnostmiznotraj območja **<content>**. Vendar pa jepomembno je omeniti, da se stolpce možnosti deklarirane *external="true"* nikoli ne sme spreminjati znotraj **<optionset>** in deklarirani stolpci možnosti *external="false"* (privzeto) nikoli ne smejo bitispremenjeni zunaj **<optionset>**.

### 10.6.2 Alternative: Kdaj ne uporabljati naborov možnosti

Nabori možnosti (optionsets) so močno orodje, vendar lahko včasih naredijo več škode kotkoristi, saj dodajajo precejšnjo kompleksnost, tako z vidika razvijalca vtičnika kot z vidika uporabnika. Torej dvakrat premislite pri njihovi uporabi. Tukaj je nekaj nasvetov:

- Za nekatere preproste primere lahko element **<matrix>** nudi uporabno lahkotno alternativo.
- Naj vaš vtičnik ne naredi preveč. Navedli smo primer uporabe za nabor možnosti za vtičnik za risanje več črt znotraj ene risbe. Ampak na splošno ni dobra ideja ustvariti vtičnika, ki bo ustvaril posamezne ploskveza vsak element v naboru možnosti. Raje naj vtičnik ustvari eno ploskev in uporabnik ga lahko pokliče večkrat.
- Če ne pričakujete več kot dva ali tri predmete v kompletu, razmislite o ponavljanju možnosti, namesto tega kar ročno.

# Poglavje 11

# Ravnanje z odvisnostmi in težavami z združljivostjo

## 11.1 RKWard združljivost različic

Po najboljših močeh se trudimo zagotoviti, da bodo vtičniki razviti za staro različico RKWard ostali funkcionalni v kasnejših različicah RKWard. Vendar pa obratno ni vedno res, saj so dodane nove funkcije. Ker vsi uporabniki ne uporabljajo najnovejše različice RKWard, to pomeni, da lahko vaš vtičnikne deluje za vsakogar.

Kadar se zavedate takšnih težav z združljivostjo, se morate dokumentirati to dejstvo v svoji datoteki `.pluginmap` z uporabo elementa **<dependencies>**. Ukaz **<dependencies>** je mogoče določiti kot neposrednega naslednika <dokumenta> `.pluginmap` ali kot podrejenega elementa posameznih definicij **<component>**. V prvi odvisnosti veljajo za *vse* vtičnike v mapi. V slednjem primeru samo posamezni**<komponenti>**(-e). Lahko tudi mešate top ″global″ in″posebne″ odvisnosti. V tem primeru so ″globalne″ odvisnostidodani tistim iz posamezne komponente.

Poglejmo majhen primer:

```
<document ...>
        <dependencies rkward_min_version="0.5.0c" />
        <components ...>
                <component id="myplugin" file="reduced_version_of_myplugin. ↩
                    xml" ...>
                        <dependencies rkward_max_version="0.6.0z" />
                </component>
                <component id="myplugin" file="fancy_version_of_myplugin. ↩
                    xml" ...>
                        <dependencies rkward_min_version="0.6.1" />
                </component>
                ...
x        </components ...>
</document>
```

V tem primeru je znano, da vsi vtičniki zahtevajo vsaj različico 0.5.0c RKWard. En vtičnik z *id=″myplugin″* je na voljo v dveh alternativnih različicah. Prva, okleščena različica se da uporabiti za RKWard različice pred 0.6.1. Slednja uporablja funkcije ki so nove v RKWard 0.6.1 in se bo uporabljal samo od RKWard 0.6.1 naprej.

Zagotavljanje alternativnih različic, kot je ta, je zelo uporabniku prijazen načinuporabe novih funkcij, medtem ko še vedno ohranja podporo za prejšnje različiceRKWard. Alternativne različice morajo imeti isti *id* (v nasprotnem primeru bodo prikazana opozorila) in se jih lahko samo definira*znotraj iste* `.pluginmap` mape.

Vtičnik, ki ni združljiv z delujočo različico RKWard, inki ne prihaja z alternativno različico, bo prezrt z opozorilom.

> OPOMBA
> Pravzaprav RKWard 0.6.1 je prva različica, ki razlaga odvisnosti - inporoča o napakah odvisnosti – sploh. Torej, v nasprotju s tem, kar lahko primerpredlaga, navedba kakršnih koli prejšnjih različic v odvisnostih ne bo imelakakršnega koli neposrednega učinka (vendar je morda še vedno dobra ideja za namene dokumentacije).

*Včasih* bo mogoče težave z nezdružljivostjo različic celo obravnavati *znotraj* ene same datoteke `.pluginmap` z uporabo elementa **<dependency_check>** opisane v naslednjem razdelku.

## 11.2 R združljivost različice

Podobno kot *rkward_min_version* in *rkward_max_version* element **<dependencies>** omogoča specifikacijo atributov *R_min_version* in *R_max_version*. Vendar pa obstajajo naslednje razlike:

- Vtičniki, ki ne izpolnjujejo R zahteve za različico *niso* trenutno preskočeni pri branju datoteke `.pluginmap`. Uporabnik lahkoše vedno kliče vtičnik in ne bo videl nobenega takojšnjega opozorila (v prihodnjih različicah, bo verjetno prikazano opozorilno sporočilo)
- Posledično tudi *ni* mogoče definirati alternativnih različic vtičnika glede na različico R.
- Vendar pa je pogosto enostavno doseči združljivost za nazaj, kot je prikazano spodaj.Če poznate R težave z združljivostjo, razmislite o uporabi teganamesto definiranja odvisnosti od določene različice R.

V mnogih primerih je enostavno zagotoviti zmanjšano funkcionalnost, če določena funkcija ni na voljo v delujoči različici R. Upoštevajte naslednji kratek primer vtičnika `.xml` datoteke:

```
<dialog [...]>
        <logic>
                <dependency_check id="ris210" R_min_version="2.10.0"/>
                <connect client="compression.xz.enabled" governor="ris210 ↩
                    "/>
        </logic>
        [...]
        <radio id="compression" label="Compression method">
                <option label="None" value="">
                <option label="gzip" value="gzip">
                <option id="xz" label="xz" value="xz">
        </radio>
        [...]
</dialog>
```

V tem primeru bo možnost stiskanja "xz" preprosto onemogočena, koR različica izvajalnega okolja je starejša od 2.10.0 (ki ni podpirala xzstiskanja). Element **<dependency_check>**podpira iste atribute kot **<dependencies>**element v datoteki `.pluginmap`. Ustvari logično lastnost, kar je true, če so podane odvisnosti izpolnjene, sicer pa false.

## 11.3 Odvisnosti od R paketov

Odvisnosti od posebnih R pakete je mogoče definirati, vendar od RKWard 0.6.1 te odvisnosti niso niti preverjene niti nameščene/naložene samodejno. So prikazani v datotekah pomoči vtičnika. Tukaj jeprimer definicije:

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

OPOMBA
Vedno se prepričajte, da dodate ustrezne klice `require()`, čevaš vtičnik potrebuje za nalaganje določenih paketov.

OPOMBA
Če distribuirate svoj `.pluginmap` kotR paket in so vsi vtičniki odvisni od določenega paketa, potem morate definirati to odvisnost od R na ravni paketa. Definiranjeodvisnosti od R paketi na ravni RKWard `.pluginmap` jenajbolj uporabno, če le nekateri vaši vtičniki potrebujejo odvisnost, jeodvisnost ni na voljo pri CRAN ali vaš `.pluginmap` nidistribuiran kot R paket.

## 11.4 Odvisnosti od drugih datotek RKWard `.pluginmap`

Če so vaši vtičniki odvisni od vtičnikov, definiranih v drugem `.pluginmap` (to je *ni* del vašega paketa) lahko definirate to odvisnost takole:

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

Trenutno ne bo niti naložil, niti namestil, niti ne bo opozoril na manjkajoče datoteke `.pluginmap`, ampak vsaj informacije o odvisnostih (in kje jih dobiti) bodo prikazani na strani s pomočjo vtičnika. Ni vam treba (in ne smete) navesti odvisnosti od datotek `.pluginmap`, ki so dobavljene z uradne RKWard distribucije ali na datotekah `.pluginmap`, ki so znotraj vašega paketa. Nadalje, če je zahtevana datoteka `.pluginmap` distribuirana kot R paket, deklarirajte odvisnosti od paketa (kot je prikazano v prejšnjem razdelku), namesto od mape.

Če želite zagotoviti, da so zahtevani datoteke `.pluginmap` dejansko naložene, uporabite**<require>**-oznako (glejte referenca za podrobnosti).

## 11.5 Primer

Za pojasnitev, kako se lahko mešajo definicije odvisnosti, je tukaj kombinirani primer:

```
<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="myplugin" file="reduced_version_of_myplugin. ↩
                    xml" ...>
                        <dependencies rkward_max_version="0.6.0z" />
                </component>
                <component id="myplugin" file="fancy_version_of_myplugin. ↩
                    xml" ...>
                        <dependencies rkward_min_version="0.6.1" />
                </component>
                ...
x        </components ...>
</document>
```

# Poglavje 12

# Prevodi vtičnikov

Doslej smo uporabili nekaj konceptov v zvezi s prevodi ali ″i18n″ (kar je na kratko za ″internacionalizacijo″, ki ima 18 znakov med i in n). V tem poglavju podajamo bolj poglobljeno poročilo o tem, kaj je i18nfunkcionalno za RKWard vtičnike. Večinoma *ne* potrebujete vse to v svojih vtičnikih. Vendar je morda dobra idejaprebrati to poglavje v celoti, saj bi vam razumevanje teh konceptov moralo pomagati pri ustvarjanju vtičnikov, ki so popolnoma prevedljivi in omogočajo visoko kakovost prevodov.

## 12.1 Splošni premisleki

V nasprotju s tem je treba razumeti eno pomembno točko o prevodih programske opremev nasprotju s prevodi drugih besedilnih gradiv, je, da prevajalci pogosto precej težko dobijo popolno sliko *kar* prevajajo. Prevodi programske opreme nujno temeljijo na precejkratkih fragmentih besedila: vsaka oznaka, ki jo daste **<option>** v **<radio>**, vsak niz, ki ga označiteza prevajanje v klicu funkcije **i18n()**, bo oblikoval ločeno ″prevajalska enota″. V bistvu bo vsak tak fragment predstavljen prevajalcu ločeno. No, ne popolna izolacija, kot poskušamo prevajalcu zagotoviti čim več smiselnega konteksta ekstrahirano, samodejno. Toda na nekaterih točkah bodo potrebni prevajalci dodaten kontekst za razumevanje niza, zlasti tam, kjer so nizi kratki.

## 12.2 i18n v xml datotekah RKWard

Pri RKWard XML datotekah, bo i18n večinoma kar deloval. Če pišetevaš lasten **.pluginmap** (npr. za zunanji vtičnik), boste morali podati `po_id` poleg `id-ja datoteke pluginmap vtičnika` . To definira »katalog sporočil« za uporabo. Na splošno to mora biti nastavljen tako, kot je `id` vašega**.pluginmap**, vendar če podate več povezanih **.pluginmap** v enem paketu, boste verjetno želeli določitiskupni `po_id` v vaših mapah. `po_id` datoteke **.pluginmap** podedujejo vsi vtičniki, ki so deklarirani v njem, razen če to izjavi drugače `po_id`.

Za vtičnike in strani s pomočjo vam ni treba povedati RKWard katere nize je treba prevesti, ker je to na splošno razvidno iz njihove rabe. Vendar, kot je razloženo zgoraj, morate biti pozorni na nize, ki bi lahkobiti dvoumni ali potrebujejo razlago, da bi bili pravilno prevedeni. Za nize, ki bi lahko imeli različne pomene, navedite `i18n_context` takole:

```
<checkbox id="scale" label="Scale" i18n_context="Prikaži merilo"/> < ↩
    checkbox id="scale" label="Scale" i18n_context="Umeri risbo"/>
```

Zagotavljanje `i18n_context` bo povzročilo dva nizaprevedena ločeno. Sicer bi si delila en sam prevod. Poleg tega je kontekst prikazan prevajalcu. `i18n_context`-atribut je podprt na vsehelementih, ki imajo lahko prevedljive nize, nekje, vključno z elementi, ki vsebujejo besedilo (npr.**<text>**-elementi).

V drugih primerih ima niz za prevod en sam nedvoumen pomen, vendar bo morda še vedno potrebno nekaj razlage. V tem primeru lahko dodate komentar, ki bo prikazan prevajalcem. Primeri lahko vključujejo:

```
<!-- i18n: No, this is not a typo for screen plot! -->
<component id="scree_plot" label="Scree plot"/>

<!-- i18n: If you can, please make this string short. Having more than some ↩
     15 chars
looks really ugly at this point, and the meaning should be mostly self- ↩
    evident to the
user (selection from a list of values shown next to this element) -->
<valueslot id="selected" label="Pick one"/>
```

Upoštevajte, da morajo biti taki komentarji pred elementom, na katerega se nanašajo, in se morajozačeti z ″i18n:″ ali ″TRANSLATORS:″.

Nazadnje, v redkih primerih boste morda želeli izključiti določene nize iz prevoda. To je morda smiselno, če na primer ponudite izbiro med več R imeni funkcij v kontrolniku **<radio>**.Potem ne želite, da se prevajajo (vendar odvisno od konteksta, raje razmislite o opisni oznaki):

```
<radio id="transformation" label="R function to apply">
  <option id="as.list" noi18n_label="as.list()"/>
  <option id="as.vector" noi18n_label="as.vector()"/>
  [...]
</radio>
```

Upoštevajte, da boste izpustili atribut *label*,nato in namesto tega podajte *noi18n_label*. tudiupoštevajte, da v nasprotju z *i18n_context* inkomentarjev, bo uporaba *noi18n_label* vašvtičnik ni združljiv z različicami RKWard pred 0.6.3.

## 12.3 i18n v RKWard datotekah in razdelkih js

V nasprotju z datotekami `.xml` ustvarjanjedatoteke `.js` vtičnika, ki ga je mogoče prevestizahteva več dela po meri. Ključna razlika je v tem, da ni spodobnega samodejnega načina ugotavljanja, ali naj bo niz prikazan kot človeku berljiv niz ali kot del kode. Torej morate to označiti vi. Ves čas smo že kazali primere tega. Tukaj je bolj popoln opis funkcij i18n, ki so na voljo v kodi js, in nekaj namigov za bolj zapletene primere:

**i18n (msgid, [...])**

Najpomembnejša funkcija. Označi niz za prevod. Niz (ne glede na to, ali je preveden ali ne) je vrnjen označen z dvojnimi narekovaji (′″′). V nizu lahko uporabite poljubno število ograd, kot je prikazano spodaj.Uporaba takšnih nadomestnih mest namesto veriženja majhnih podnizov je veliko lažja za prevajalce:

```
i18n ("Compare objects %1 and %2", getString ('x'), getString ('y'));
```

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

Enako kot **i18n()**, a dodatno nudi kontekst sporočila:

```
i18nc ("proper name, not state of mind", "Mood test");
```

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

Enako kot **i18n()**, vendar za sporočila, ki se lahko razlikujejo v edninski ali množinski obliki (in nekateri jeziki razlikujejo še več številskih oblik). Upoštevajte, da tako kot pri **i18n()**, lahko uporabite poljubno število zamenjav, vendar prva ('%1') jeobvezna in mora biti celo število.

```
i18np ("Comparing a single pair", "Comparing %1 distinct pairs",  ↩
    n_pairs);
```

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

**i18np()** z dodanim kontekstom sporočila.

**comment (comment, [indentation])**

Odmeva komentar kode, označen za prevod. V nasprotju z drugimi i18n() funkcijami, to ni v narekovajih, vendar je v vsako vrstico komentarja dodano '#'.

```
comment ("Transpose the matrix");
                        echo ('x <- t (x)\n');
```

Za dodajanje komentarjev prevajalcem (glejte zgoraj za razpravo o razlikah med komentarjem in kontekstom), dodajte komentar, ki se začne z "i18n:" ali "translators:" neposredno nad**i18n()**-pokliči za komentiranje. Npr.:

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

### 12.3.1 i18n in citati

Večinoma vam ne bo treba skrbeti za vedenje i18n() glede spoštovanja do citatov. Ker so običajno prevedljivi nizi nizovni literali, njihovo citiranje je prava stvar in vam prihrani nekaj tipkanja. V funkcijah, kot je **makeHeaderCode()/Header()**, ki običajno citirate njihove argumente, so nizi v i18n() zaščiteni pred podvajanjem citiranja. V bistvu to deluje tako, da se najprej pošlje prevedeni niz skozi **quote()** (da bo citiran), nato skozi**noquote()** (za zaščito pred dodatnim citiranjem). Če potrebujete prevedljiv niz, ki ni v narekovajih, uporabite **i18n(noquote ("Moje sporočilo"))**. Če potrebujete prevodniza, ki ga želite navesti, ga drugič pošljite prek **quote()**, *dvakrat*.

Kljub temu na splošno ni dobra ideja, da bi imena funkcij ali spremenljivk poimenovali tako, di bi bili prevedljivi. Prvič, R, je programski jezik, po naravi v angleščini in ni internacionalizacije samega jezika. Komentarji kode so drugačna stvar, vendar bi morali za to uporabitifunkcijo **comment()**. Drugič, izdelavasintaktično relevantnega dela ustvarjene kode prevedljivo pomeni, daprevodi lahko dejansko pokvarijo vaš vtičnik. Npr. če nič hudega slutečprevajalnik prevede niz, ki je mišljen kot ime spremenljivke, v dva različni besedi s presledkom vmes.

## 12.4 Vzdrževanje prevoda

Zdaj, ko ste svoj vtičnik naredili prevedljivega, kako ga dejansko dobite prevedenega? Na splošno vas mora to skrbeti le, ko razvijate zunanji vtičnik. Za vtičnike vglavnem skladišču RKWard, je vsa čarovnija narejena namesto vas. Tukaj je osnovapotek dela za zunanje vtičnike. Upoštevajte, da potrebujete nameščena orodja "gettext":

- Označite vse nize, po potrebi zagotovite kontekst in komentarje

- Zaženite **python3 scripts/update_plugin_messages.py --extract-only /path/to/my.pluginmap**. scripts/update_plugin_messages.py trenutno ni del izvornih izdaj, a se ga da najti v izvleku repozitorija.
- Distribuiraj dobljeni **rkward__*POID*.pot** vašim prevajalcem. Za zunanje vtičnike priporočamo, da ga postavite v podmapo ″po″ v inst/rkward.
- Prevajalec odpre datoteko v prevajalskem orodju, kot je **lokalize**. Pravzaprav, tudi če ne boste pripravili nobenega prevoda,sami, poskusite ta korak sami. Brskajte po izvlečenih nizi, da poiščete morebitne težave / dvoumnosti.
- Prevajalec shrani prevod kot **rkward__*POID*.*xx*.po** (kjer je *xx* koda jezika) in pošlje nazaj k vam.
- Kopirajte **rkward__*POID*.*xx*.po** v svoje vire poleg**rkward__*POID*.pot**. Poženite **python3 scripts/update_plugin_messages.py /path/to/my.pluginmap** (Opomba: tokrat brez *--extract-only*). To bo združilo prevod z vsemi vmesnimi spremembami niza, preveditеprevod in ga namestite v**DIR_OF_PLUGINMAP**/po/*xx*/LC_MESSAGES/rkward__*POID*.mo (kjer *xx* je spet koda jezika).
- Prav tako morate vključiti neprevedeni prevod (tj. **rkward__*POID*.*xx*.po**) v svojo distribucijo v podimeniku »po«. .
- Za vsako posodobitev vtičnika zaženite **skripte python3/update_plugin_messages.py /path/to/my.pluginmap** za posodobitev .potdatoteko, temveč tudi obstoječe datoteke .po in prevedene kataloge sporočil.

## 12.5 Pisanje prevodov vtičnikov

Predvidevamo, da poznate svoj prevajalski poklic ali pa ste pripravljeni prebrati to, drugje. Nekaj besed posebej o prevodih RKWard dodatkov pa:

- RKWard vtičnikov ni bilo mogoče prevajati do različice 0.6.3 in večinomaprej niso bili napisani z i18n v mislih. Tako se boste srečaliprecej več dvoumnih nizov in drugih i18n težav kot v drugih zrelih projekti. Prosimo, da tiho delate okoli teh težav, ampak nam (alivzdrževalcem vtičnikov) dajte vedeti za to, da lahko odpravimo te težave.
- Veliko RKWard vtičniki se nanašajo na visoko specializirane izraze, od obdelave podatkovin statistike, pa tudi z drugih področij znanosti. V mnogih primerih bo dober prevod zahteval vsaj osnovno poznavanje teh področij. V nekaterih primerih *ni* dobrega prevoda za tehnični izraz, najboljša možnost pa je lahko, da izraz pustite nepreveden ali da vključite angleški izraz v oklepaju. Ne osredotočajte se preveč na 100 % oznake prevedenih nizov, se osredotočite na zagotavljanje dobrega prevoda, tudi če to pomeni preskok nekaterih nizov (ali celo preskok nekaterih katalogov sporočil kotcelota). Drugi uporabniki bodo morda lahko zapolnili morebitne tehnične vrzeli.

# Poglavje 13

# Informacije o avtorju, licenci in različici

Torej ste napisali niz vtičnikov in se pripravljate na delite svoje delo. Da bi zagotovili, da bodo uporabniki vedeli, kaj je vaše delo, pod kakšnimi pogoji lahko uporabljajo in ga distribuirajo in na koga naj se obrnejo glede težav ali idej, dodate nekaj informacij *o* vaših vtičnikih. To lahko storite z elementom **<about>**. Lahko se uporablja bodisi v `.pluginmap` ali v posameznem vtičniku `.xml` (v obeh primerih kot neposredni podrejeni datotekioznaka dokumenta). Ko je podano v `.pluginmap` veljalo bo za vsevtičniki. Če je **<about>** podan na obeh mestih,informacije **<about>** v vtičniku `.xml` bo preglasila tisto v datoteki `.pluginmap`. Dodate lahko tudi element **<about>** v .rkh-strani, ki niso povezane z vtičnikom, če je to potrebno.

Tukaj je primer `.pluginmap` datoteko z le nekaj pojasnili spodaj. V primeru dvoma lahko več informacij dobite pri referenci.

```
<document
        namespace="rkward"
        id="SquaretheCircle_rkward"
>
        <about
                name="Square the Circle"
                shortinfo="Squares the circle using Heisenberg compensation ↩
                    ."
                version="0.1-3"
                releasedate="2011-09-19"
                url="http://eternalwondermaths.example.org/23/stc.html"
                license="GPL"
                category="Geometry"
        >
                <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>
```

Večino tega je treba razložiti samo po sebi, zato ne bomo razpravljali o vsaki oznaki elementa posebej. Toda poglejmo nekaj podrobnosti, ki verjetno potrebujejo nekaj komentarja za lažje razumevanje.

Element *category* v **<about>** je mogoče definirati precej prosto, vendar bi moral biti smiseln, kot jenaj bi se uporabljal za razvrščanje vtičnikov v skupine. Vsi drugi atributi v tej začetni oznaka so obvezni in morajo biti napolnjeni z razumno vsebino.

Vsaj en **<author>** z veljavnim e-poštnim naslovom mora biti podan tudi vloga 'aut' ('avtor') mora biti podana. V primeru, da vaš vtičnik povzroča težave ali bi ga kdo želel deliti z drugimi hvaležnost z vami, mora biti enostavno vzpostaviti stik z nekom, ki je vpleten. Za dodatne informacije o drugih veljavnih vlogah, kot je 'ctb' za kodoprispevke ali 'cre' za vzdrževanje paketa, glejteR dokumentacija o person().

> OPOMBA
> Ne pozabite, da lahko uporabite **<include>** in / ali**<insert>** za ponavljanje informacij v več `.xml` datoteke (npr. informacije oavtorju, ki je sodeloval pri več vtičnikih). Več informacij.

> NAMIG
> Ni vam treba napisati tega XML kodo ročno. Če uporabljate funkcijo `rk.plugin.skeleton()` iz paketa rkwarddev in zagotovite vse potrebne informacije prek *about* bo samodejno ustvaril `.pluginmap` datoteka z delujočim razdelkom <o programu> za vas.

# Poglavje 14

# Delite svoje delo z drugimi

## 14.1 Zunanji vtičniki

Od različice 0.5.5 je RKWard zagotavlja udoben način namestitve dodatnih vtičnikov tretjih oseb, ki ne sodijo v sam osnovni paket. Tem pravimo 'zunanji vtičniki'. Pridejo v obliki R paketa in ga je mogoče upravljati neposredno prek običajnega upravljanja paketov značilnosti R in/ali RKWard.

Ta razdelek dokumentacije opisuje, kako naj bodo zunanji vtičniki pakirani, tako da jih RKWard lahko uporablja. Sama izdelava vtičnika je seveda enaka prejšnjim razdelkom. Se pravi, verjetno bi moralinajprej napišite delujoč vtičnik in se nato vrniti sem, če želite izvedeti, kako ga distribuirati.

Ker so zunanji vtičniki razmeroma mlada funkcija, bi se lahko podrobnosti o tem verjetno spremenile v prihodnjih izdajah. Vabimo vas, da prispevate svoje ideje, da bi izboljšati postopek.

> NAMIG
> Ti dokumenti pojasnjujejo podrobnosti zunanjih vtičnikov, da se lahko naučite, kako jih delujejo. Poleg tega si oglejte tudi rkwarddev paket, ki je bil zasnovan za avtomatizacijo velikega dela procesa pisanja.

## 14.2 Zakaj zunanji vtičniki?

Število paketov za razširitev funkcionalnosti R je že zdaj neizmerno in raste. Po eni strani vas želimo spodbuditi k pisanju vtičnikov tudi za najbolj specializirane naloge, ki jih potrebujete. Na drugi strani se povprečen uporabnik ne bi smel izgubiti v ogromnih drevesih menijev, polnih neznanih statističnih izrazov. Zato se je zdelo smiselno ohraniti vtičnik obdelava v RKWard tudi precej modularen. Ekipa RKWard ohranja svoje lastno javno skladišče paketov na https://files.kde.org/rkward/R/ , določeno za gostovanje vaših zunanjih vtičnikov.

Praviloma velja, da vtičniki, za katere se zdi, da služijo široko uporabljenemu namenu (npr. t-testi) bi morali postati del osrednjega paketa, medtem ko tisti, ki služijo bolj omejeni skupini ljudi s posebnimi interesi, naj bi bili na voljo kot izbirni paket. Za vas kot avtorja vtičnika je najboljša praksa, da samo začnite z zunanjim vtičnikom.

## 14.3 Struktura paketa vtičnikov

Za pravilno namestitev in pravilno delovanje zunanjih vtičnikov morajo slediti nekaterim strukturnim smernicam glede njihove hierarhije datotek.

### 14.3.1 Hierarhija datotek

Oglejmo si prototipno hierarhijo datotek dodelanega vtičnika v arhivu. Ni vam treba vključiti vseh teh imenikov in/ali datotek za vtičnik za delo (preberite, če želite izvedeti, kaj je nujno potrebno), razmisliteo tem kot na primer 'najboljše prakse':

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

> **OPOMBA**
> V tem primeru morajo vsi pojavi plugin_name,testsuite_name in sometest_name biti ustrezno nadomeščeni s svojimi pravilnimi imeni. Tudi ll je namesto za jezikovno okrajšavo (npr., 'de', 'en' ali 'es').

> **NAMIG**
> Te hierarhije datotek vam ni treba ustvariti ročno. Če uporabljatefunkcijo `rk.plugin.skeleton()` iz rkwarddev paketa, bosamodejno ustvari vse potrebne datoteke in imenike, razen imenika `po`, ki ga ustvari in upravlja skript za prevajanje.

#### 14.3.1.1 Osnovne komponente vtičnika

Obvezno je vključiti vsaj tri datoteke: a `.pluginmap`, vtičnik .xmlopis in datoteko vtičnika .js. to je celo imenik ″plugins″ neobvezen. Morda bi le pomagalo dati vaše datoteke v nekem redu, še posebej, če vključite več kot en vtičnik v pogovorno okno v arhivu, kar seveda ni problem.

Lahko imate imenike za dejanske datoteke vtičnikov, kot se vam zdi primerno, preprosto morajo podobni `.pluginmap`.Prav tako je mogoče celo vključiti več `.pluginmap` datotek, če ustreza za vaše potrebe, vendar jih morate potem vse vključiti v 'plugin_name.pluginmap'.

Vsak R paket mora imeti veljavno datoteko `DESCRIPTION`,kar je ključnega pomena tudi za RKWard ga prepoznajo kot ponudnika vtičnikov. Večina informacij, ki jih nosi, je potreben tudi v vtičniku Meta-information in morda dependencies, vendar v drugem formatu (dokumentacija R pojasnjuje DESCRIPTION datoteka s podrobnostmi).

Poleg splošne vsebine `DESCRIPTION` ne pozabite vključiti tudi vrstice 'Enhances: rkward'.To bo povzročilo RKWard da samodejno pregleda paket za vtičnike, če jeje nameščen. Primer datoteke `DESCRIPTION` je videti takole:

```
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: Squares the circle using 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"))
```

> **NAMIG**
> Te datoteke vam ni treba pisati ročno. Če uporabljate funkcijo`rk.plugin.skeleton()` iz rkwarddev paket in zagotovite vsepotrebne informacije prek možnosti 'o', bosamodejno ustvaril delujočo datoteko `DESCRIPTION` za vas.

#### 14.3.1.2 Dodatne informacije (neobvezno)

`Dnevnik sprememb`, `README`,`AUTHORS`, `LICENSE` bi morale biti samo po sebi razumljive in so v celoti neobvezne. Pravzaprav jih ne bo tolmačil RKWard, zato so bolj namenjeni prenašanju dodatnih informacij, ki bi lahko bile relevantne npr. za distributerje. Večina njihove ustrezne vsebine (avtorji, licenčni pogoji itd.) bodo vključeni v dejanski vtičnik datoteke vseeno (glejte razdelek o metainformacijah). Upoštevajte, da lahko vse te datoteke postavite tudi nekje v imenik "inst", če jih ne želite samo, da je prisoten v izvornem arhivu, ampak tudi v nameščenem paketu.

#### 14.3.1.3 Samodejno testiranje vtičnika (izbirno)

Drug izbirni imenik je "tests", ki je namenjen zagotavljanju datotekpotrebno za samodejno testiranje vtičnikov. Titesti so koristni za hitro preverjanje, ali vaši vtičniki še vedno delujejo z novimirazličicami R ali RKWard. Če želite vključiti teste, se morate res omejiti se na shemo poimenovanja in hierarhijo, prikazano tukaj. To je, da morajo biti testi v imeniku z imenom `tests`, kivključuje datoteko `testsuite.R` in mapo s standardi testov, poimenovani po ustreznem testnem nizu. Lahko pa zagotovite več kot ena zbirka testov; v tem primeru, če jih ne želite

pripeti vseh v eni datoteki `testsuite.R` jih lahko razdelite v npr.eno datoteko za vsak preskusni paket in ustvarite en `testsuite.R`s klici `source()` za vsako datoteko zbirke. V kateremkoli primeru ustvarite ločene podimenike s testnimi standardi za vsak definiran skupek.

Prednost vzdrževanja te strukture je, da je mogoče izvajati teste vtičnikov preprosto s klicem `rktests.makplugintests()` iz paketa rkwardtests brez dodatnih argumentov. Oglejte si spletno dokumentacijo oSamodejno testiranje vtičnikov za nadaljnje podrobnosti.

## 14.4 Gradnja paketa vtičnikov

Kot je bilo že razloženo, so zunanji vtičniki RKWard v veljavi R paketi, zato je postopek pakiranja enak. V nasprotju s ″pravim″ R paketom, čisti paket vtičnikov ne nosi nobenih nadaljnje R Kode (čeprav lahko seveda dodate RKWard vtičniki za običajno R paketi kot, z uporabo istih metod, ki so razložene tukaj). To bi moralo biti celo lažje ustvariti delujoč paket, če imate veljavno`DESCRIPTION` in se držite hierarhije datotek razloženo v prejšnjem razdelku.

Najlažji način, da dejansko zgradite in preizkusite svoj vtičnik, je uporabaukaza R v ukazni vrstici, na primer:

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

> NAMIG
> Paketa vam ni treba graditi v ukazni vrstici. Če uporabite funkcijo `rk.build.package()` iz rkwarddev paket,bo namesto vas zgradil in/ali preveril vaš paket vtičnikov.

# Poglavje 15

# Razvoj vtičnika s paketom rkwarddev

## 15.1 Pregled

Pisanje zunanjih vtičnikov vključuje pisanje datotek v treh jezikih (XML,JavaScript in R) ter ustvarjanje standardizirane hierarhije imenikov. Da bi bilo to veliko lažje za voljne razvijalce vtičnikov, smo zagotovili paket rkwarddev. Zagotavlja število preprostih R funkcij za ustvarjanje datoteke XML koda za vsa pogovorna okna elemente, kot so zavihki, potrditvena polja, spustni seznami ali brskalniki datotek kot funkcije za ustvarjanje JavaScript kode in RKWard datoteke pomoči za začetek. Funkcija `rk.plugin.skeleton()` ustvaripričakovano drevo imenikov in vse potrebne datoteke, kjer naj bi bile biti.

Ta paket ni privzeto nameščen, ampak ga je treba namestiti ročno iz RKWard repozitorija. To lahko storite z GUI vmesnikom (**Nastavitve** → **Konfiguriraj pakete**) ali iz katere koli tekoče R seje:

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

rkwarddev je odvisen od drugega klicanega majhnega paketa'XiMpLe', ki je zelo preprost XML razčlenjevalnik in generator terprisoten tudi v istem skladišču.

Tam lahko najdete tudi celotno dokumentacijo v formatu PDF. Podrobnejši uvod v delo s paketom najdete v vinjeto rkwarddev.

## 15.2 Praktični primer

Da boste dobili idejo, kako izgleda 'skriptiranje vtičnika',v primerjavi z neposrednim pristopom, ki ste ga videli v prejšnjih poglavjih, bomo ponovno ustvarili celoten vtičnik t-test -- tokrat samo z R funkcijami paketa rkwarddev.

> NAMIG
> Paket bo dodal novo GUI pogovorno okno za RKWard pod **Datoteka** → **Izvoz** → **Ustvari RKWard skript vtičnika**. Kot že ime pove, lahko ustvarite okostja vtičnikov za nadaljnje urejanje z njim. To pogovorno okno je nato ustvaril rkwarddev skript, ki ga najdete v 'demo' imeniku nameščenega paketa in izvornih paketov,kot dodaten primer. Zaženete ga lahko tudi tako, da pokličete `demo(˝skeleton_dialog˝)`

### 15.2.1 GUI opis

Takoj boste opazili, da je potek dela precej drugačen: v nasprotju z neposrednim pisanjem kode XML ne začnete z definicijo **<document>**, temveč neposredno z elementi vtičnika, ki jih želite imeti v dialogu. Vsak element vmesnika - naj bodo to potrditvena polja, spustni meniji, spremenljive reže ali karkoli drugega - lahko dodelite posameznim R objektom in nato združite te objekte v dejanski GUI. Paket ima funkcije za vsako oznako XML, ki jo je mogoče uporabiti za definiranje vtičnika GUI, in večina jih ima celo isto ime, le s predpono `rk.XML.*`. Na primer definiranje **<varselector>** in dveh elementov **<varslot>** za *"x"* in *"y"* primera t-testa lahko izvedete tako:

```
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")
```

Najbolj zanimiva podrobnost je verjetno *source=spremenljivke*: Pomembna značilnost paketa je, da lahko vse funkcije ustvarijo samodejne ID-je, tako da vam ni treba razmišljati o nobeni njihovi vrednosti *id* ali si jih zapomniti, da se nanašajo na določeni element vtičnika. Lahko preprosto podate R predmete kot referenca, kot vse funkcije, ki potrebujejo ID nekega drugega elementa, ga lahko tudi preberejo iz teh predmetov. `rk.XML.varselector()` je malo poseben, saj običajno nima posebne vsebine, iz katere bi lahko naredil ID (lahko, vendar le, če določite oznako), zato moramo nastaviti ime ID-ja. Toda `rk.XML.varslot()` ne potrebuje *id.name*argumentov tukaj, tako da bi to zadostovalo:

```
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)
```

Če želite ponovno ustvariti primer kode do točke, bi morali nastavitivse ID vrednosti ročno. Ker pa nam bo paket olajšal življenje,od zdaj naprej nam to ne bo več mar.

> **NAMIG**
> rkwarddev je zmožen veliko avtomatizacije, da vam pomaga zgraditi vaše vtičnike. Vendar je morda bolje, da ga ne uporabite v polni meri. Če je vaš cilj ustvariti kodo, ki ne deluje lelahko pa ga tudi človek zlahka prebere in primerja z vašim generatorskim skriptom bi morali razmisliti o tem, da vedno nastavite uporabne ID-je s *id.name*. Poimenovanje vašega R predmeti, ki so enaki tem ID-jem, bodo prav tako pomagali k pridobivanju skriptne kode, ki je lahko razumljiva.

Če želite videti, kako je videti koda XML definiranega elementa, če želite jo izvozite v datoteko, lahko predmet preprosto pokličete po njegovem imenu. Torej, če ste klicali 'var.x' v vaši R seji, bi morali videti nekaj podobnega:

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

Nekatere oznake so uporabne le v kontekstu drugih. Zato je npr.ne boste našli funkcije za oznako **<option>**.Namesto tega so opredeljeni izbirni gumbi in spustni seznami, vključno z njihovimi možnostmi kot poimenovan seznam, kjer imena predstavljajo oznake, ki bodo prikazane vpogovornem oknu, njihova vrednost pa je imenovan vektor, ki ima lahko dva vnosa,*val* za vrednost možnosti in logično vrednost*chk*, da določite, ali je ta možnost privzeto označena.

```
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")
        )
)
```

Rezultat je videti takole:

```
<radio id="rad_usngtsth" label="using test hypothesis">
        <option label="Two-sided" value="two.sided" />
        <option label="First is greater" value="greater" />
        <option label="Second is greater" value="less" />
</radio>
```

Vse, kar manjka elementom zavihka 'Osnovne nastavitve', je potrditveno polje za seznanjene vzorce in strukturiranje vseh teh elementov v vrsticah in stolpcih:

```
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))
```

`rk.XML.cbox()` je redka izjema, kjer ime funkcije ne vsebuje celotnega imena oznake, da prihranite nekaj tipkanja za ta pogosto uporabljen element. To zdaj vsebuje `basic.settings`:

```
<row id="row_vTFSPP10TF">
        <varselector id="vars" />
        <column id="clm_vrsTFSPP10">
                <varslot id="vrsl_compare" label="compare" source="vars"  ↩
                    types="number" required="true" />
                <varslot id="vrsl_against" label="against" i18n_context=" ↩
                    compare against" source="vars" types="number" required=" ↩
                    true" />
                <radio id="rad_usngtsth" label="using test hypothesis">
                        <option label="Two-sided" value="two.sided" />
                        <option label="First is greater" value="greater" />
                        <option label="Second is greater" value="less" />
                </radio>
                <checkbox id="chc_Pardsmpl" label="Paired sample" value="1" ↩
                     value_unchecked="0" />
        </column>
</row>
```

Na podoben način bodo naslednje vrstice ustvarile R objekti za elemente zavihka 'Možnosti', ki predstavljajo funkcije za spinboxe, okvirje in raztezanje:

```
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")
```

Vse, kar moramo storiti, je, da objekte združimo v zavihek in ga postavimo v pogovorno okno:

```
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)))
)
```

Ustvarimo lahko tudi razdelek čarovnika z dvema stranema z uporabo istih predmetov, tako da bodo njuni ID-ji ekstrahirani za oznake **<copy>**:

```
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 is ↩
                             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 is 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)))
```

To je to za GUI. Globalni dokument bo na koncu združil `rk.plugin.skeleton()`.

### 15.2.2 JavaScript Koda

Do zdaj se je morda zdelo, da uporaba paketa rkwarddev ni veliko pomagala. To se bo prav zdaj spremenilo.

Prvič, tako kot nam ni bilo treba skrbeti za ID-je elementov pri definiranju GUI postavitve, nam ni treba skrbeti za JavaScript imena spremenljivk v naslednjem koraku. Če želite več nadzora, lahko napišete navadno kodo JavaScriptin jo prilepite v ustvarjeno datoteko. Verjetno pa je veliko učinkoviteje to narediti na način rkwarddev.

Predvsem vam ni treba sami definirati nobene spremenljivke, saj lahko `rk.plugin.skeleton()` skenira vaš XML kodo in samodejno definira vse spremenljivke, ki jih boste verjetno potrebovali -- na primer, ne da bi se trudili vključiti potrditvenega polja, če pozneje ne uporabite njegove vrednosti ali stanja. Tako lahko začnemo pisati dejansko R kodo, ki takoj ustvari JS.

> NAMIG
> Funkcija `rk.JS.scan()` lahko skenira tudi obstoječe XML datoteke za spremenljivke.

Paket ima nekaj funkcij za konstrukte kode JS, ki se pogosto uporabljajo v RKWard vtičniki, kot je funkcija `echo()` ali pogoji `if() {...} else {...}`. Med JS in R je nekaj razlik, npr. za `paste()` v R vejico uporabljate za združevanje znakovnih nizov, medtem ko za `echo()` v JS uporabljate '+', vrstice pa se morajo končati s podpičjem. Z uporabo R funkcije, lahko skoraj pozabite na te razlike in še naprej pišete R kodo.

Te funkcije lahko sprejmejo različne razrede vhodnih objektov: bodisi golo besedilo, R predmeti z XML kodo, kot je zgoraj, ali pa rezultate nekaterih drugih funkcij JS paketa. Na koncu boste vedno poklicali `rk.paste.JS()`, ki se obnaša podobno kot `paste()`, vendar jih bo glede na vhodne objekte nadomestil z njihovimi XML ID, JavaScript ime spremenljivke ali celo celoten JavaScript kodni bloki.

Za primer t-testa potrebujemo dva predmeta JS: enega za izračun rezultatov in enega za njihovo tiskanje v funkciji `printout()`:

```
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)
```

Kot lahko vidite, rkwarddev nudi tudi R implementacija funkcije `echo()`. Vrne natanko en niz znakov s svojo veljavno različico JS. Morda boste tudi opazili, da so vsi R predmeti tukaj tisti, ki smo jih ustvarili že prej. Samodejno bodo nadomeščeni s svojimi imeni spremenljivk, zato bi moralo biti to precej intuitivno. Kadarkoli potrebujete samo to zamenjavo, lahko uporabite funkcijo `id()`, ki prav tako vrne natanko en niz znakov iz vseh objektov, ki jih je prejel (lahko bi rekli, da se obnaša kot `paste( )` z zelo specifično zamenjavo predmeta).

Funkcija `js()` je ovoj, ki vam omogoča uporabo **if(){...} else {...}** pogojev R, kot ste jih že uporabljali. Prevedeni bodo neposredno v kodo JS. Ohranja tudi nekatere operatorje, kot so **<**, **>=** ali **||**, tako da lahko logično primerjate svoje R predmetov, ne da bi večino časa morali navajati. Oglejmo si nastali objekt 'JS.calc', ki zdaj vsebuje niz znakov s to vsebino:

```
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");
```

OPOMBA
Druga možnost je za ugnezdene pogoje `if()` v`js()`, lahko uporabite `ite()`funkcijo, ki se obnaša podobno kot R-jeva `ifelse()`.Vendar so pogojni stavki, izdelani z uporabo `ite()`običajno težje berljivi in jih je treba zamenjati s `js()` kadar koli je to mogoče.

### 15.2.3 Mapa vtičnikov

Ta razdelek je zelo kratek: ni nam treba napisati `.pluginmap` sploh, ker ga samodejno lahko ustvari `rk.plugin.skeleton()`. Hierarhijo menija je mogoče določiti prek *`pluginmap`* možnosti:

```
[...]
        pluginmap=list(
                name="Two Variable t-Test",
                hierarchy=list("analysis", "means", "t-Test"))
        [...]
```

### 15.2.4 Stran s pomočjo

Tudi ta razdelek je zelo kratek: `rk.plugin.skeleton()` ne more napisati celotne strani pomoči iz informacij, ki jih ima. Lahko pa skenira XML dokument tudi za elemente, ki si verjetno zaslužijo vnos strani s pomočjo, in samodejno ustvari predlogo strani s pomočjo za naš vtičnik. Vse, kar moramo storiti kasneje, je, da napišemo nekaj vrstic za vsak naveden razdelek.

NAMIG
Funkcija `rk.rkh.scan()` lahko skenira tudi obstoječe XML datotek za ustvarjanje okostja datoteke pomoči.

### 15.2.5 Ustvarite datoteke vtičnika

Zdaj prihaja zadnji korak, v katerem bomo vse ustvarjene objekte predali `rk.plugin.skeleton()`:

```
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)
```

Datoteke bodo privzeto ustvarjene v časovnem imeniku. Zadnje tri možnosti niso potrebne, vendar so zelo priročne: *`load=TRUE`* samodejno doda nov vtičnik RKWard konfiguraciji (ker je v začasnem imeniku in zato bo prenehal obstajati, ko je RKWard zaprt, ga bo RKWard ob naslednjem zagonu znova samodejno odstranil), *`edit=TRUE`* bo odprl vse ustvarjene datoteke za urejanje v

RKWard zavihki urejevalnika in *show=TRUE* bo poskušal neposredno zagnati vtičnik, tako da lahko brez klika preverite, kako je videti. Razmislite o dodajanju *overwrite=TRUE*, če boste svoj skript večkrat zagnali (npr. po spremembah kode), saj privzeto nobena datoteka ne bo prepisana.

Objekt rezultata 'plugin.dir' vsebuje pot do imenika, v katerem je bil ustvarjen vtičnik. To je lahko uporabno v kombinaciji s funkcijo `rk.build.package()` za izdelavo dejanskega R paket za skupno rabo vtičnika z drugimi -- npr. tako, da ga pošljete na RKWard razvojno ekipo, ki bo dodana v naše skladišče vtičnikov.

### 15.2.6 Celoten scenarij

Če povzamemo vse zgoraj navedeno, je tukaj celoten skript za ustvarjanje delujočega primera t-testa. Poleg že razložene kode tudi naloži paket, če je potrebno, in uporablja okolje `local()`, tako da vsi ustvarjeni objekti ne bodo končali v vašem trenutnem delovnem prostoru (razen 'plugin.dir'):

```
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 is 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 is 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 Dodajanje strani s pomočjo

Če želite napisati stran s pomočjo za svoj vtičnik, najbolj preprosti način za to je, da dodate posebna navodila neposredno v definicije XML elementi, ki jim pripadajo:

```
variables <- rk.XML.varselector(
  id.name="vars",
  help="Select the data object you would like to analyse.",
  component="Data"
)
```

Besedilo, podano parametru *help*, lahko nato pridobi `rk.rkh.scan()` in ga zapiše na stran s pomočjo te komponente vtičnika. Da bi to tehnično delovalo, pa mora `rk.rkh.scan()` vedeti, kateri R objekti pripadajo eni komponenti vtičnika. Zato morate zagotoviti tudi parameter *component* in zagotoviti, da je enak za vse predmete, ki pripadajo skupaj.

Ker boste običajno združili veliko predmetov v eno pogovorno okno in boste morda želeli tudi ponovno uporabiti predmete, kot je **<varslot>**, v več komponentah vaših vtičnikov, je mogoče komponento globalno definirati s funkcijo `rk.set.comp()`. Če je nastavljeno, se predpostavlja, da vsi naslednji objekti, uporabljeni v vašem skriptu, pripadajo tej določeni komponenti, dokler `rk.set.comp()` ni ponovno poklicana z drugim imenom komponente. Nato lahko izpustite parameter *component*:

```
rk.set.comp("Data")
variables <- rk.XML.varselector(
  id.name="vars",
  help="Select the data object you would like to analyse."
)
```

Če želite na stran s pomočjo dodati globalne razdelke, kot je **<summary>** ali **<usage>**, uporabite funkcije, kot je `rk.rkh.summary()` ali `rk.rkh.usage()` ustrezno. Njihovi rezultati se nato uporabijo za nastavitev elementov seznama, kot je *summary* ali *usage* v parametru *rkh* funkcije `rk.plugin.component()` /`rk.plugin.skeleton()`.

## 15.4 Prevajanje vtičnikov

Paket rkwarddev lahko ustvari zunanje vtičnike s polno podporo za i18n. Na primer, vse ustrezne funkcije, ki generirajo objekte XML, ponujajo izbirni parameter za podajanje *i18n_context* ali *noi18n_label*:

```
varComment <- rk.XML.varselector(id.name="vars", i18n=list(comment="Glavni ↩
    izbirnik spremenljivk")) varContext <- rk.XML.varselector(id.name="vars ↩
    ", i18n =list(context="Izbirnik glavne spremenljivke")) cboxNoi18n <- rk ↩
    .XML.cbox(label="Moč", id.name="moč", i18n=FALSE)
```

Zgornji primeri ustvarijo rezultat, kot je ta:

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

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

# cboxNoi18n
<checkbox id="power" noi18n_label="Power" value="true" />
```

Obstaja tudi podpora za prevedljivo kodo JS. Pravzaprav poskuša paket privzeto dodati klice `i18n()` na mestih, kjer je to običajno koristno. Funkcija `rk.JS.header()` je dober primer:

```
jsHeader <- rk.JS.header("Rezultati testa")
```

To ustvari naslednjo kodo JS:

```
new Header(i18n("Rezultati testa")).print();
```

Lahko pa tudi ročno označite nize v vaši kodi JS kot prevedljive z uporabo funkcije `i18n()` tako kot bi to storili, če bi datoteko JS napisali neposredno.

# Dodatek A

# Referenca

## A.1 Vrste lastnosti/Modifikatorji

Na nekaterih mestih v tem uvodu smo govorili o 'lastnostih' GUI elementov ali kako drugače. Pravzaprav obstaja več različnih vrst nepremičnin. Običajno vam to ni treba skrbeti, saj lahko po zdravi pameti katero koli nepremičnino povežete s katero koli drugo lastnino. Vendar pa znotraj obstajajo različne vrste lastnosti. To je pomembno pri pridobivanju nekaterih posebnih vrednosti v predlogi JS. V stavkih getString (˝id˝)/getBoolean (˝id˝)/getList (˝id˝) lahko podate tudi nekaj tako imenovanih 'modifikatorjev', kot je ta: `getString (˜id.modifier˜ )`. Ta modifikator bo vplival na način tiskanja vrednosti. Preberite si seznam lastnosti in modifikatorje, ki jih dajejo na voljo:

**Lastnosti niza**

Najbolj preprosta vrsta lastnosti, ki se uporablja za preprosto držanje dela besedila. Modifikatorji:

**Brez modifikatorja (˝˝)**

Niz, kot je definiran / nastavljen.

**citiran**

Niz v narekovajih (primeren za prenos v R kot znak).

**Logične lastnosti**

Lastnosti, ki so lahko vklopljene ali izklopljene, resnične ali napačne. Na primer lastnosti, ustvarjene z oznakami <convert>, tudi lastnost, ki spremlja <potrditveno polje> (glej spodaj). Glede na dani modifikator bodo vrnjene naslednje vrednosti:

**Brez modifikatorja (˝˝)**

Lastnost bo privzeto vrnila 1, če je res, in 0 v nasprotnem primeru. Priporočen način pridobivanja logičnih vrednosti je uporaba `getBoolean()`. Upoštevajte, da bo za `getString()` niz ˝0˝ vrnjen, ko je lastnost napačna. Ta niz bi bil v JS ovrednoten kot true, ne kot false.

**˝labeled˝**

Vrne niz »true«, če je true, »false«, če je false ali kateri koli niz po meri, ki je bil določen (običajno v <potrditvenem polju>).

**˝true˝**

Vrne niz, kot da bi bila lastnost true, tudi če je false

**˝false˝**

Vrne niz, kot da bi bila lastnost false, tudi če je true

**"not"**
To dejansko vrne drugo logično lastnost, ki je obratna od trenutne (tj. false, če je true, true, če je false)

**"numeric"**
Zastarelo, predvideno za združljivost za nazaj. Enako kot brez modifikatorja "". Vrni "1", če je lastnost true, ali "0", če je false.

**Lastnosti celega števila**

Lastnost, zasnovana tako, da hrani celoštevilsko vrednost (seveda pa še vedno vrne številski niz znakov predlogi JS). Ne sprejema nobenih modifikatorjev. Uporablja se v <spinbox>es (glejte spodaj)

**Lastnosti realnega števila**

Lastnost, zasnovana tako, da hrani vrednost realnega števila (seveda pa še vedno vrne številski niz znakov v predlogo JS). Uporablja se v <spinbox>es (glejte spodaj)

**Brez modifikatorja ("")**
Za `getValue() / getString()` to vrne enako kot "formatted". V prihodnjih različicah bo namesto tega mogoče pridobiti številčno predstavitev.

**"formatted"**
Vrne oblikovano število (kot niz).

**Lastnosti RObject**

Lastnost je oblikovala izbor enega ali več R predmetov. Najpogosteje se uporablja v varselectors in varslots. Glede na dani modifikator bodo vrnjene naslednje vrednosti:

**Brez modifikatorja ("")**
Privzeto bo lastnost vrnila polno ime izbranega predmeta. Če je izbran več kot en predmet, bodo imena predmetov ločena s prelomi vrstic ("\n").

**"shortname"**
Kot zgoraj, vendar vrne le kratka imena za objekte. Na primer, predmet znotraj seznama bi dobil samo ime, ki ga ima znotraj seznama, brez imena seznama.

**"label"**
Kot zgoraj, vendar vrne RKWard oznaka(e) predmeta(ov) (če oznaka ni na voljo, je to isto kot kratko ime)

**Lastnosti seznama nizov**

Ta lastnost vsebuje seznam nizov.

**Brez modifikatorja ("")**
Za `getValue()/getString()` to vrne vse nize, ločene z "\n". Any "\n" v vsakem elementu so ubežni kot dobesedni "\n". Vendar je priporočena uporaba namesto tega pridobivanje vrednosti s funkcijo `getList()`, ki bo vrnila niz nizov.

**"joined"**
Vrne seznam kot en sam niz z elementi, združenimi z "\n". V nasprotju z brez modifikatorja ("") posamezni nizi _ne_ ubežijo.

**Lastnosti kode**

Lastnost vtičnikov, ki so ustvarili kodo. To je pomembno za vdelovalne vtičnike, da lahko vdelate kodo, ki jo ustvari vdelani vtičnik, v kodo, ki jo ustvari vtičnik (najvišje ravni). Glede na dani modifikator bodo vrnjene naslednje vrednosti:

**Brez modifikatorja ("")**
Vrne celotno kodo, tj. razdelki "preprocess", "calculate", "printout" in (toda ne "preview") sestavljeni v en niz.

**"preprocess"**
: Vrne le predprocesni del kode

**"calculate"**
: Vrne samo odsek kode za izračun

**"printout"**
: Vrne le izpisni del kode

**"preview"**
: Vrne razdelek predogleda kode

## A.2 Elementi splošnega namena za uporabo v kateri koli XML datoteki (`.xml`, `.rkh`, `.pluginmap`)

**<snippets>**

Dovoljeno kot neposredni podrejeni element <dokumenta> vozlišče in samo tam. Postavljen naj bo blizu vrha datoteke. Glejte razdelek o uporabi odlomkov. Samo en <odlomek> element je lahko prisoten. Izbirno, brez atributov.

**<snippet>**

Določa en sam delček. Dovoljeno samo kot neposredni podrejeni element <nippets/> element. Lastnosti:

**<id>**
: Identifikacijski niz za delček. Obvezno.

**<insert>**

Vstavi vsebino <snippet>. Dovoljeno povsod. Lastnosti:

**<snippet>**
: Identifikacijski niz delčka, ki ga želite vstaviti. Obvezno.

**<include>**

Vključi vsebino druge XML datoteka (vse znotraj elementa <document> te datoteke). Dovoljeno povsod. Lastnosti:

**<file>**
: Ime datoteke glede na imenik, v katerem je trenutna datoteka. Zahtevano.

## A.3 Elementi za uporabo v XML opis vtičnika

Lastnosti elementov so navedene v ločenem razdelku.

### A.3.1 Splošni elementi

**<document>**

Prisoten mora biti v vsaki datoteki description.xml kot korensko vozlišče. Brez posebne funkcije. Brez atributov

**<about>**
Informacije o tem vtičniku (avtor, licenca itd.). Ta element je dovoljen v datoteki `.xml` posameznega vtičnika in v `.pluginmap` datoteke. Glejte `.pluginmap` sklic na datoteko za podrobnosti o sklicu, poglavje o informacijah 'o' za uvod.

**<code>**
Določa, kje iskati predlogo JS za vtičnik. Uporabite samo enkrat na datoteko kot neposredni podrejeni element oznake dokumenta. Lastnosti:

***file***
Ime datoteke predloge JS glede na imenik, v katerem je plugin-xml

**<help>**
Določa, kje iskati datoteko pomoči za vtičnik. Uporabite samo enkrat na datoteko kot neposredni podrejeni element oznake dokumenta. Lastnosti:

***file***
Ime datoteke za pomoč glede na imenik, v katerem je plugin-xml

**<copy>**
Lahko se uporablja kot podrejeni (neposredni ali posredni) glavnih elementov postavitve, tj. <pogovorno okno> in <čarovnik>. To se uporablja za kopiranje celotnega bloka XML elementi 1:1. Lastnosti:

***id***
ID, ki ga je treba iskati. <copy> bo iskal prejšnji XML element, ki je dobil isti ID, in ga kopirajte, vključno z vsemi podrejenimi elementi.

***copy_element_tag_name***
V nekaj primerih boste želeli skoraj dobesedno kopijo, vendar spremenite ime oznake elementa za kopiranje. Najpomembnejši primer tega je, ko želite kopirati celoten <tab> iz pogovornega vmesnika na <stran> vmesnika čarovnika. V tem primeru bi nastavili copy_element_tag_name="stran" za samodejno pretvorbo.

### A.3.2 Definicije vmesnikov

**<dialog>**
Definira vmesnik pogovornega okna tipa dialog. Postavite GUI definicija znotraj te oznake. Uporabite samo enkrat na datoteko kot neposredni podrejeni element oznake dokumenta. Za vtičnik je potrebna vsaj ena od oznak "dialog" ali "wizard". Lastnosti:

***label***
Napis za pogovorno okno

***recommended***
Ali naj se pogovorno okno uporabi kot "recommended - priporočeni" vmesnik (tj. vmesnik, ki bo privzeto prikazan, razen če je uporabnik RKWard konfiguriral tako, da privzeto uporablja določen vmesnik)? Ta atribut trenutno nima učinka, saj je implicitno "true", razen če je priporočen čarovnik.

**<wizard>**
Definira vmesnik vrste čarovnika. Postavite GUI definicijo znotraj te oznake. Uporabite samo enkrat na datoteko kot neposredni podrejeni element oznake dokumenta. Za vtičnik je potrebna vsaj ena od oznak "dialog" ali "wizard". Sprejme samo <page> ali <embed>-oznake kot neposredni podrejene. Lastnosti:

***label***
Napis za čarovnika

***recommended***
Ali naj se čarovnik uporablja kot "recommended - priporočen" vmesnik (tj. vmesnik, ki bo prikazan privzeto, razen če je uporabnik konfiguriral RKWard za privzeti vmesnik)? Izbirno, privzeto je "false".

### A.3.3 Elementi postavitve

Vsi elementi v tem razdelku sprejemajo atribut id=″identifierstring″. Ta atribut je neobvezen za vse elemente. Uporabite ga lahko na primer za skrivanje/onemogočanje celotnega elementa postavitve in vseh elementov, ki jih vsebuje (glejte poglavje GUI logic). Id-string ne sme vsebovati ″.″ (pike) ali ″;″ (podpičja) in bi moralo biti na splošno omejeno na alfanumerične znake in podčrtaj (»_«). Navedeni so le dodatni atributi.

**<page>**

Določa novo stran v čarovniku. Dovoljeno samo kot neposredni podrejeni element elementa <wizard>.

**<row>**

Vsi neposredni podrejeni elementi oznake ″row - vrstica″ bodo postavljeni od leve proti desni.

**<column>**

Vsi neposredni podrejeni elementi oznake ″column - stolpec″ bodo postavljeni od zgoraj navzdol.

**<stretch>**

Privzeto so elementi v GUI zasedejo ves razpoložljivi prostor. Na primer, če imate dva stolpca drug ob drugem, je levi poln elementov, desni pa vsebuje samo en ukaz **<radio>**, nadzor **<radio>** se bo razširil navpično, čeprav v resnici ne potrebuje razpoložljivega prostora, in videti bo bo grdo. V tem primeru res želite dodati ″blank″ pod **<radio>**. Za to uporabite <stretch> element. Preprosto bo porabil nekaj prostora. Ne pretiravajte s tem elementom, običajno je dobra ideja za GUI elemente, da dobijo ves razpoložljivi prostor, le včasih bo postavitev razmaknjena. <stretch> element ne sprejema nobenih argumentov, niti ″id″. Prav tako ne smete postaviti otrok v <stretch> element (z drugimi besedami, vedno ga boste uporabljali samo kot ″<stretch/>″)

**<frame>**

Nariše okvir/polje okoli svojih neposrednih podrejenih elementov. Uporablja se lahko za vizualno združevanje povezanih možnosti. Postavitev znotraj okvirja je od zgoraj navzdol, razen če postavite <row> znotraj. Lastnosti:

***label***

Napis za okvir (neobvezno)

***checkable***

Okvirje je mogoče narediti checkable - označljive. V tem primeru bodo vsi vsebovani elementi onemogočeni, ko okvir ni odkljukan, in omogočeni, ko je odkljukan. (izbirno, privzeto je ″false″)

***checked***

Samo za okvirje vrste chackable: Ali naj bo okvir privzeto odkljukan? Privzeto je ″true″. Ni interpretirano za okvirje, ki niso checkable.

**<tabbook>**

Organizira elemente v tabbook - knjižico zavihkov. Sprejema le oznake <tab> kot neposredne podrejene.

**<tab>**

Določa stran v tabbook - knjižici zavihkov. Postavite GUI definicija za zavihek znotraj te oznake. Lahko se uporablja samo kot neposredni podrejeni element <tabbook> oznaka. <tabbook> mora imeti vsaj dva definirana zavihka. Lastnosti:

***label***

Napis za stran zavihka (obvezno)

**<text>**

Prikaže text besedilo, obdano s to oznako v GUI. Podprto je nekaj preprostih HTML slogovnih oznak (zlasti <b>, <i>, <p> in <br/>). Kljub temu naj bo formatiranje čim manjše. Vstavljanje popolnoma prazne vrstice doda trd prelom vrstice. Lastnosti:

***type***

Vrsta besedila. Eno izmed ″normalno″, ″warning″ ali ″error″. To vpliva na videz besedila (neobvezno, privzeto je normalno)

### A.3.4 Aktivni elementi

Vsi elementi v tem razdelku sprejemajo atribut id=″identifierstring″. Ta atribut je obvezen za vse elemente. Navedeni so le dodatni atributi. Id-string ne sme vsebovati ″.″ (pik).

**<varselector>**

Zagotavlja seznam razpoložljivih predmetov, med katerimi lahko uporabnik izbere enega ali več. Za uporabnost potrebuje enega ali več <varslot> kot protipostavko. Lastnosti:

***label***

Oznaka za varselector - izbirnik spremenljivk (neobvezno, privzeto je ″Izberi spremenljivko(e)″)

**<varslot>**

Uporablja se v povezavi z ″varselector″, da uporabniku omogoči izbiro ene ali več spremenljivk. Lastnosti:

***label***

Oznaka za varslot (priporočeno, privzeto je ″Variable:″)

***source***

Varselektor iz katerega pridobite izbor (obvezno, razen če se povežete ročno ali z uporabo source_property)

***source_property***

Poljubna lastnost za kopiranje vrednosti, ko kliknete gumb za izbiro. Če je navedeno, to preglasi atribut ″source″.

***required***

Ali je za oddajo kode potrebno, da ta varslot vsebuje veljavno vrednost. Oglejte si required-property (izbirno, privzeto je false)

***multi***

Ali varslot vsebuje samo enega (privzeto, ″false″) ali več predmetov

***allow_duplicates***

Ali sme varslot sprejeti samo edinstvene objekte (privzeto, »false«) ali če je lahko isti objekt dodan večkrat.

***min_vars***

Pomembno samo, če je multi=″true″: Najmanjše število spremenljivk, ki jih je treba izbrati, da bo izbor veljaven (neobvezno, privzeto je ″1″)

***min_vars_if_any***

Pomembno samo, če je multi=″true″: nekatere reže varslot se lahko štejejo za veljavne, če je na primer reža varslot prazna ali vsebuje vsaj dve vrednosti. To določa, koliko spremenljivk mora biti izbranih, če sploh katere (2 v primeru). (izbirno, privzeto je ″1″)

***max_vars***

Smiselno samo, če je multi=″true″: največje število spremenljivk za izbiro (neobvezno, privzeto je ″0″, kar pomeni, da ni največjega števila)

***razredi***
: Če podate enega ali več R imen razredov (ločena s presledki (″ ″)), bo varslot tukaj sprejemal samo predmete, ki pripadajo tem razredom (neobvezno, *uporabljajte zelo previdno*, uporabniku ne sme biti preprečeno sprejemanje veljavnih izbir in R ima *veliko* različnih razredov)

***vrste***
: Če podate eno ali več vrst spremenljivk (ločenih s presledki (″ ″)), bo tukaj varslot sprejel samo objekte teh vrst. Veljavni tipi so ″unknown″, ″number″, ″string″, ″factor″, ″invalid″. (Izbirno, *uporabljajte zelo previdno*, uporabniku ne bi smeli preprečiti veljavnih izbir in RKWard ne pozna vedno vrste spremenljivke)

***num_dimensions***
: Število dimenzij, ki jih mora imeti objekt. ″0″ (privzeto) pomeni, da je sprejemljivo poljubno število dimenzij. (izbirno, privzeto je ″0″)

***min_length***
: Najmanjša dolžina, ki jo mora imeti predmet, da je sprejemljiv. (izbirno, privzeto je ″0″)

***max_length***
: Največja dolžina, ki jo mora imeti predmet, da je sprejemljiv. (izbirno, privzeto je največje celo število, ki ga je mogoče predstaviti v sistemu)

**<valueselector>**
: Zagotavlja seznam razpoložljivih nizov (ne R objektov), ki jih je treba izbrati v enem ali več spremljajočih <valuesslot>s. Možnosti nizov je mogoče definirati z uporabo oznak <option> kot neposredne podrejene (glejte spodaj) ali nastaviti z uporabo dinamičnih lastnosti. Lastnosti:

***label***
: Oznaka za izbirnik vrednosti (neobvezno, privzeto ni oznake)

**<valueslot>**
: Uporablja se v povezavi z <valueselector> da uporabniku omogoči izbiro enega ali več elementov niza. Ta element je večinoma identičen <varslot> in ima enake atribute, razen tistih, ki se nanašajo na lastnosti sprejemljivih elementov (tj. razredi, tipi, num_dimensions, min_length, max_length).

**<radio>**
: Določa skupino radijskih ekskluzivnih gumbov (hkrati je mogoče izbrati samo enega). Zahteva vsaj dve oznaki <option> kot neposredne podrejene. Nobene druge oznake niso dovoljene kot podrejene. Lastnosti:

***label***
: Oznaka za radijski nadzor (priporočeno, privzeto je »Izberite eno:«)

**<dropdown>**
: Določa skupino možnosti, od katerih je mogoče hkrati izbrati eno in samo eno s spustnim seznamom. To je funkcionalno enakovredno <radio>, vendar je videti drugače. Zahteva vsaj dve oznaki <option> kot neposredne podrejene. Nobene druge oznake niso dovoljene kot podrejene. Lastnosti:

***label***
: Oznaka za spustni seznam (priporočeno, privzeto je »Izberite enega:«)

**<select>**
: Zagotavlja seznam razpoložljivih nizov, iz katerih lahko uporabnik izbere poljubno število. Možnosti niza je mogoče definirati z uporabo oznak <option> kot neposredne podrejene (glejte spodaj) ali nastaviti z uporabo dinamičnih lastnosti. Lastnosti:

***label***
: Oznaka za <select> (izbirno, privzeto ni oznake)

***single***
: Če je nastavljeno na true, bo mogoče izbrati samo eno vrednost namesto več vrednosti hkrati (boolean, privzeto je false)

**<option>**

Lahko se uporablja samo kot neposredni podrejeni element <radio>, <dropdown>, <valueselector> ali <select> element. Predstavlja eno izbirno možnost v radijskem nadzoru ali spustnem seznamu. Kot <option> elementi so vedno del enega od izbirnih elementov, običajno nimajo lastnega ″id-ja″, vendar glejte spodaj. Lastnosti:

***label***
: Oznaka za možnost (obvezno)

***value***
: Vrednost niza, ki jo bo nadrejeni element vrnil, če je ta možnost označena/izbrana (obvezno)

***checked***
: Ali naj bo možnost označena/izbrana privzeto ″true″ ali ″false″. V <radio> ali <dropdown>, je lahko samo ena možnost nastavljena na `checked="true"`, in če nobena možnost ni označena, bo prva možnost v nadrejenem elementu samodejno označeno/izbrano. V <select> je lahko izbranih poljubno število možnosti. (izbirno, privzeto je ″false″)

***id***
: Določanje parametra ″id″ za <option> elementov ni obvezen (in pravzaprav je priporočljivo, da ne nastavite ″id-ja″, razen če ga res potrebujete). Vendar pa vam bo podajanje ″id-ja″ omogočilo, da dinamično omogočite/onemogočite <option>, tako da se povežete z logično lastnostjo `id_of_radio.id_of_optionX.enabled`. Trenutno to deluje za možnosti znotraj <radio> ali <dropdown> samo elementi; <valueselector> in <select> možnosti trenutno ne podpirajo ID-jev.

**<checkbox>**

Določa potrditveno polje, tj. eno samo možnost, ki jo je mogoče vklopiti ali izklopiti. Lastnosti:

***label***
: Oznaka za potrditveno polje (obvezno)

***value***
: Vrednost, ki jo potrditveno polje vrne, če je označeno (obvezno)

***value_unchecked***
: Vrednost, ki bo vrnjena, če potrditveno polje ni potrjeno (izbirno, privzeto je ″″, tj. prazen niz)

***checked***
: Ali naj bo možnost privzeto potrjena ″true″ ali ″false″ (neobvezno, privzeto je ″false″)

**<frame>**

Element okvirja se na splošno uporablja kot čisti element postavitve in je naveden v razdelku o elementih postavitve. Lahko pa ga tudi preverite, tako da hkrati deluje kot preprosto potrditveno polje.

**<input>**

Določa polje za prosti vnos besedila. Lastnosti:

***label***
: Oznaka za vnosno polje (obvezno)

***initial***
: Začetno besedilo besedilnega polja (neobvezno, privzeto je ″″, tj. prazen niz)

***size***
: Eden izmed ″small″, ″medium″ ali ″large″. ″large″ definira večvrstično vnosno polje, ″small in ″medium″ sta enovrstični polji (izbirno, privzeto je ″medium″)

***required***
Ali se za oddajo kode zahteva, da ta vnos ni prazen. Oglejte si required-property (izbirno, privzeto je false)

**<matrix>**
Tabela za vnos matričnih podatkov (ali vektorjev) v GUI.

> OPOMBA
> Ta vnosni element *ni* optimiziran za vnos/urejanje velikih količin podatkov. Čeprav ni stroge omejitve glede velikosti <matrix>, na splošno ne sme preseči približno desetih vrstic/stolpcev. Če pričakujete večje podatke, dovolite uporabnikom, da jih izberejo kot R objekt (kar je lahko dobra ideja kot alternativna možnost, v skoraj *vseh* primerih, kjer uporabljate matrični element).

Lastnosti:

***label***
Oznaka za tabelo (obvezno)

***način***
Eno od ″integer″, ″real″ ali ″string″. Vrsta podatkov, ki bodo sprejeti v tabeli (obvezno)

***min***
Najmanjša sprejemljiva vrednost (za matrike tipa ″integer″ ali ″real″) (neobvezno, privzeta vrednost je najmanjša predstavljiva vrednost)

***max***
Največja sprejemljiva vrednost (za matrike tipa ″integer″ ali ″real″) (neobvezno, privzeta vrednost je največja predstavljiva vrednost)

***allow_missings***
Ali so v matriki dovoljene manjkajoče (prazne) vrednosti. To je implicirano za matrike ali način ″string″ (neobvezno, privzeto je false).

***allow_user_resize_columns***
Ko je nastavljeno na true, lahko uporabnik dodaja stolpce tako, da vnese v skrajno desne (neaktivne) celice (neobvezno, privzeto je true).

***allow_user_resize_rows***
Ko je nastavljeno na true, lahko uporabnik dodaja vrstice s tipkanjem v najnižje (neaktivne) celice (izbirno, privzeto je true).

***rows***
Število vrstic v matriki. Nima učinka za allow_user_resize_rows=″true″.

> OPOMBA
> To lahko nadzirate tudi z nastavitvijo lastnosti ″rows″.

(izbirno, privzeto je 2).

***columns***
Število stolpcev v matriki. Nima učinka za allow_user_resize_columns=″true″.

> OPOMBA
> To lahko nadzirate tudi z nastavitvijo lastnosti ″columns″.

(izbirno, privzeto je 2).

***min_rows***
Najmanjše število vrstic v matriki. Matrica se ne bo skrčila pod to velikost. (izbirno, privzeto je 0; glejte tudi: `allow_missings`.).

***min_columns***
Najmanjše število stolpcev v matriki. Matrica se ne bo skrčila pod to velikost. (izbirno, privzeto je 0; glejte tudi: `allow_missings`.).

***fixed_height***
: Prisilite da GUI element ostane na začetni višini. Ne uporabljajte v kombinaciji z matricami, kjer se lahko število vrstic kakor koli spremeni. Uporabno, posebej pri ustvarjanju vektorskega vhodnega elementa (columns="1"). Če je ta možnost nastavljena na true, ne bo prikazan noben vodoravni drsnik, tudi če matrika presega razpoložljivo širino (ker bi to vplivalo na višino). (neobvezno, privzeto je false).

***fixed_width***
: Nekoliko napačno imenovan: Predpostavimo, da se število stolpcev ne bo spremenilo. Zadnji (ali običajno samo) stolpec bo raztegnjen, da zavzame razpoložljivo širino. Ne uporabljajte v kombinaciji z matrikami, kjer se lahko število stolpcev kakor koli spremeni. Uporabno, posebej pri ustvarjanju vektorskega vhodnega elementa (rows="1"). (neobvezno, privzeto je false).

***horiz_headers***
: Nizi za vodoravno glavo, ločeni z ";". Glava bo skrita, če je nastavljena na "". (izbirno, privzeto je številka stolpca).

***vert_headers***
: Nizi, ki se uporabljajo za navpično glavo, ločeni z ";". Glava bo skrita, če je nastavljena na "". (izbirno, privzeto številka vrstice).

**<optionset>**

Uporabniški vmesnik za ponavljanje nabora možnosti za poljubno število elementov (uvod v nabore možnosti). Lastnosti:

***min_rows***
: Če je naveden, bo nabor označen kot neveljaven, razen če ima vsaj toliko vrstic (neobvezno, celo število).

***min_rows_if_any***
: Kot min_rows, vendar bo preizkušen le, če obstaja vsaj ena vrstica (neobvezno, celo število).

***max_rows***
: Če je določen, bo niz označen kot neveljaven, razen če ima največ to število vrstic (neobvezno, celo število).

***keycolumn***
: ID stolpca, ki bo deloval kot ključni stolpec. Nabor možnosti z (veljavnim) ključnim stolpcem bo deloval kot "nastavljen" nabor možnosti. Nabor možnosti brez ključnega stolpca bo omogočil ročno vstavljanje/odstranjevanje elementov. Ključni stolpec mora biti označen kot zunanji. (izbirno, privzeto ni ključnega stolpca).

Podrejeni elementi:

**<optioncolumn>**
: Določa en neobvezen stolpec niza. Za vsako vrednost, ki jo želite pridobiti iz nabora možnosti, morate deklarirati ločen <optioncolumn>. Lastnosti:

  ***id***
  : ID neobveznega stolpca (obvezno, niz).

  ***external***
  : Nastavite na true, če je neobvezni stolpec nadzorovan zunaj nabora možnosti optionset (neobvezno, logično, privzeto na false).

  ***label***
  : Če je podan, bo optioncolumn prikazan v stolpcu s to oznako (neobvezno, niz, privzeto ni prikazano).

  ***connect***
  : Lastnost za povezavo tega stolpca z možnostmi, podana kot id znotraj območja <content>. Za zunanji <optioncolumn> bo ustrezna vrednost nastavljena na zunanje nastavljeno vrednost. Za navadne (nezunanje) <optioncolumn> bo ustrezna vrstica lastnosti <optioncolumn> nastavljena, ko se lastnost spremeni znotraj področja vsebine. (izbirno, niz, privzeto ni povezano).

***default***
Samo za zunanje stolpce: vrednost, ki naj se predpostavi za ta stolpec, če za vnos ni znana nobena vrednost. Redko uporabno. (Izbirno, privzeto prazen niz)

**<content>**
Deklarirajte vsebino / UI nabora. Brez atributov. Kot podrejeni elementi so dovoljeni vsi običajni aktivni, pasivni in postavitveni elementi. Poleg tega je v starejših različicah RKWard (do 0.6.3) bil dovoljen poseben podrejeni element **<optiondisplay>**. To je zastarelo v RKWard 0.6.4 in ga je treba preprosto odstraniti iz obstoječih vtičnikov.

**<logic>**
Izbirna specifikacija logike uporabniškega vmesnika za uporabo *znotraj* območja vsebine nabora možnosti. Oglejte si sklic na <logic>

**<browser>**
Element, zasnovan za izbiro enega imena datoteke (ali imena imenika). Upoštevajte, da bo to polje sprejelo poljuben niz, čeprav je namenjen uporabi za datoteke, samo:

***label***
Oznaka za brskalnik (neobvezno, privzeto je ″Vnesite ime datoteke″)

***initial***
Začetno besedilo brskalnika (neobvezno, privzeto je ″″, tj. prazen niz)

***type***
Eden izmed ″file″, ″dir″ ali ″savefile″. Za izbiro obstoječe datoteke, obstoječega imenika ali neobstoječe datoteke (neobvezno, privzeto je ″file″)

***allow_urls***
Ali je mogoče izbrati (nelokalne) URL (neobvezno, privzeto je ″false″)

***filter***
Filter vrste datoteke, npr. (″*.txt *.csv″ za datoteke .txt in .csv). Samodejno je dodan ločen vnos za ″Vse datoteke″ (izbirno, privzeto je ″″, tj. Vse datoteke)

***required***
Ali je za oddajo kode potrebno, da polje ni prazno. Upoštevajte, da to ne pomeni nujno, da je izbrano ime datoteke veljavno. Oglejte si required-property (neobvezno, privzeto na true)

**<saveobject>**
Element, zasnovan za izbiro imena R predmet, v katerega želite shraniti (tj. na splošno še ne obstaja, v nasprotju z varslot):

***label***
Oznaka za vnos (neobvezno, privzeto je ″Shrani v:″)

***initial***
Začetno besedilo vnosa (neobvezno, privzeto je ″my.data″)

***required***
Ali je za oddajo kode zahtevano, da polje vsebuje dovoljeno ime objekta. Oglejte si required-property (neobvezno, privzeto na true)

***checkable***
V mnogih primerih uporabe je shranjevanje v R predmet neobvezno. V teh primerih lahko s tem atributom v element saveobject-element vključite potrditveno polje. Ko je nastavljeno na true, bo saveobject aktiviran / deaktiviran s potrditvenim poljem. Oglejte si active-property saveobject (neobvezno, privzeto je false)

***checked***
Samo za preverljive saveobject-elemente: Ali je kontrolnik privzeto označen / omogočen (neobvezno, privzeto je false)

**<spinbox>**
Vrtilno polje, v kateri lahko uporabnik izbere številsko vrednost z uporabo neposrednega vnosa s tipkovnice ali majhnih puščic gor/dol. Lastnosti:

***label***
Oznaka vrtilnega polja (priporočeno, privzeto ″Vnesite vrednost:″)

***min***
Najnižja vrednost, ki jo lahko uporabnik vnese v vrtilno polje (neobvezno, privzeto je najnižja vrednost, ki jo je tehnično mogoče predstaviti v vrtilnem polju)

***max***
Največja vrednost, ki jo lahko uporabnik vnese v vrtilno polje (neobvezno, privzeta vrednost je najvišja vrednost, ki jo je tehnično mogoče predstaviti v vrtilnem polju)

***initial***
Začetna vrednost, prikazana v vrtilnem polju (neobvezno, privzeto je ″0″)

***type***
En od ″real″ ali ″integer″. Ali bo vrtilno polje sprejelo realna števila ali samo cela števila (neobvezno, privzeto je ″real″)

***default_precision***
Smiselno samo, če ima vrtilno polje type=″real″. Podaja privzeto število decimalnih mest, prikazanih v vrtilnem polju (prikazano bo samo toliko ničel na koncu). Ko uporabnik pritisne puščice gor/dol, se to decimalno mesto spremeni. Vendar bo uporabnik morda še vedno lahko vnesel vrednosti z večjo natančnostjo (glejte spodaj) (neobvezno, privzeto je ″2″)

***max_precision***
Največje število števk, ki jih je mogoče smiselno predstaviti (neobvezno, privzeto je ″8″)

**<formula>**

Ta napredni element omogoča uporabniku, da iz izbranih spremenljivk izbere formulo/niz interakcij. Na primer za GLM lahko ta element uporabite, da uporabniku omogočite določitev interakcijskih pogojev v modelu. Lastnosti:

***fixed_factors***
ID reže varslot, ki vsebuje izbrane fiksne faktorje (obvezno)

***dependent***
ID reže varslot, ki vsebuje izbrano odvisno spremenljivko (obvezno)

**<embed>**

V ta vtičnik vdelajte drug vtičnik (glejte poglavje o vdelavi). Lastnosti:

***component***
Registrirano ime komponente za vdelavo (glejte poglavje o registraciji komponent) (obvezno)

***as_button***
Če je nastavljeno na ″true″, bo v vdelani GUI nameščen samo gumb pushbutton, vdelani GUI bo prikazan samo (v ločenem oknu), ko pritisnete gumb pushbutton (neobvezno, privzeto je ″false″)

***label***
Smiselno samo, če je as_button=″true″: oznaka gumba (priporočljivo, privzeto je ″Options″)

**<preview>**

Potrditveno polje za preklop funkcije predogleda. Upoštevajte, da od različice 0.6.5 RKWard so elementi predogleda **<preview>** označeni s posebnimi črkami v pogovornih oknih vtičnikov (ne v čarovnikih): postavljeni bodo v stolpec z gumbi, ne glede na to, kje so točno definirani v uporabniškem vmesniku. Še vedno je dobra ideja, da jih definirate na smiselnem mestu v postavitvi zaradi združljivosti za nazaj. Lastnosti:

***label***
Oznaka polja (neobvezno, privzeto je ″Preview″)

***način***
Vrsta predogleda. Podprte vrste so ″plot″ (glejte poglavje o predogledih grafov), ″output″ (glejte poglavje o (HTML) predogledih izhodov), ″data″ (glejte predoglede podatkov) in ″custom″ (glejte predoglede po meri). (izbirno, privzeto je ″plot″)

***placement***
Postavitev predogleda: ″attached″ (na glavno delovno mesto), ″detached″ (samostojno okno), ″docked″ (priloženo pogovornemu oknu vtičnika) in ″default″ (trenutno je to enako kot ″docked″, lahko pa na neki točki lahko konfigurira uporabnik). Na splošno je priporočljivo, da to pustite kot privzeto nastavitev za najboljšo skladnost uporabniškega vmesnika (izbirno, privzeto je ″default″)

***active***
Ali je predogled privzeto aktiven. Na splošno bi morali biti privzeto aktivni samo zasidrani predogledi in tudi za te obstaja razlog, zakaj so privzeti predogledi neaktivni (neobvezno, privzeto je ″false″)

### A.3.5 Logic section

**<logic>**

Element, ki vsebuje logični odsek - Logic section. Vsi spodnji elementi so dovoljeni samo znotraj elementa <logic>. Element <logic> je dovoljen samo kot neposredni podrejeni element elementa <document> (največ enkrat na dokument) ali elementov <optionset> (največ enkrat na nabor možnosti optionset). Logični razdelek dokumenta se nanaša na oba <dialog> in <wizard> GUI na enak način.

**<external>**

Ustvari novo lastnost (niz), ki naj bi bila povezana z zunanjo lastnostjo, če bo vtičnik vdelan. Oglejte si razdelek o nedokončanih - ″incomplete″ vtičnikih. Lastnosti:

***id***
ID nove lastnosti (obvezno)

***default***
Privzeta vrednost niza nove lastnosti, tj. uporabljena vrednost, če lastnost ni povezana z zunanjo lastnostjo (neobvezno, privzeto je prazen niz)

**<i18n>**

Ustvari novo lastnost (niz), ki naj bi zagotavljala internacionalizirano oznako. Lastnosti:

***id***
ID nove lastnosti (obvezno)

***label***
Oznaka. To bo prevedeno. (obvezno)

**<set>**

Lastnost nastavite na fiksno vrednost (seveda, če lastnost dodatno povežete s kakšno drugo lastnostjo, vrednost ne ostane fiksna). Na primer, če vdelate vtičnik, vendar želite skriti nekatere njegove elemente, lahko nastavite lastnost vidnosti teh elementov na false. Uporabno npr. za vdelane/vdelane vtičnike. Opomba: če obstaja več <set> elementov za en sam `id`, ima prednost tisti, ki je zadnji definiran. To se bo včasih koristno zanesti pri uporabi <include>d delov. Lastnosti:

***id***
ID lastnosti za nastavitev (obvezno)

***to***
Vrednost niza, na katero želite nastaviti lastnost to (obvezno). Opomba: Za logične lastnosti, kot sta vidnost, omogočenost, boste atribut običajno nastavili na ==″true″ ali na=″false″.

**<convert>**

Ustvarite nove logične lastnosti, ki so odvisne od stanja ene ali več različnih lastnosti. Lastnosti:

***id***
: ID nove lastnosti (obvezno)

***sources***
: ID-ji lastnosti, od katerih bo ta lastnost odvisna. Določite lahko eno ali več lastnosti, ločenih z ";" (obvezno)

***način***
: Način pretvorbe/delovanja. Eden izmed "equals", "notequals", "range", "and", "or". Če je v načinu equals, bo lastnost resnična le, če je vrednost vseh njenih virov enaka standardu atributa (glejte spodaj). Če je v načinu notequals, bo lastnost resnična samo, če se vrednost vseh njenih virov razlikuje od standarda atributa (glejte spodaj). Če je v območju načina "range", morajo biti viri numerični (celoštevilo ali realno). Lastnost bo resnična samo, če so vsi viri v območju, ki ga določata atributa min in max (glejte spodaj). Če je v načinu in, morajo biti viri logične lastnosti. Lastnost bo resnična le, če so resnični vsi viri hkrati. Če je v načinu ali, morajo biti viri logične lastnosti. Lastnost bo resnična le, če je resničen vsaj eden od virov. (obvezno)

***standard***
: Pomembno samo v načinih equals ali notequals: vrednost niza za primerjavo (obvezno, če je v enem od teh načinov)

***min***
: Pomembno samo v načinu range: najmanjša vrednost za primerjavo (neobvezno, privzeto je najnižje število s plavajočo vejico, ki ga lahko predstavi stroj)

***max***
: Pomembno samo v načinu range: največja vrednost za primerjavo (neobvezno, privzeto je največje število s plavajočo vejico, ki ga lahko predstavi stroj)

***require_true***
: Če je nastavljeno na "true", bo lastnost postala obvezna in bo veljavna samo, če je njeno stanje true/on. Torej, če je lastnost false, bo blokirala gumb **Pošlji** (izbirno, privzeto je "false").

> OPOZORILO
> Če uporabljate to, poskrbite, da lahko uporabnik zlahka zazna, kaj je narobe, na primer s prikazom pojasnjevalnega <besedila>.

**<switch>**

Ustvarite novo lastnost, ki se bo nanašala na različne ciljne lastnosti (ali fiksne nize) na podlagi vrednosti lastnosti pogoja. To omogoča ustvarjanje logike, podobne konstruktom `if()` ali `switch()`. Lastnosti:

***id***
: ID nove lastnosti (obvezno)

***condition***
: ID lastnosti pogoja (obvezno)

Podrejeni elementi:

**<true>**
: Če je lastnost pogoja boolovs vrednost, lahko podate dva podrejena elementa <true> in <false> (in samo te). (Obvezno, če je podan tudi <false>)

**<false>**
: Če je lastnost pogoja boolova vrednost, lahko podate dva podrejena elementa <true> in <false> (in samo te). (Obvezno, če je podan tudi <true>)

**<case>**
Če lastnost pogoja ni boolova vrednost, lahko navedete poljubno število elementov <case>, enega za vsako vrednost lastnosti pogoja, ki jo želite ujemati (potreben je vsaj en tak element, če lastnost pogoja ni boolova)

**<default>**
Če lastnost pogoja ni boolova vrednost, neobvezni element <default> omogoča določitev vedenja, če se noben <case> element ne ujema z vrednostjo lastnosti pogoja (neobvezno, dovoljeno samo enkrat, v kombinaciji z enim ali več elementi <case>).

Podrejeni elementi <true>, <false>, <case> in <default> prevzamejo naslednje atribute:

***standard***
Samo za elemente <case>: Vrednost, ki se ujema z lastnostjo pogoja (obvezno, niz).

***fixed_value***
Fiksni niz, ki mora biti naveden kot vrednost lastnosti <switch>, če se trenutni pogoj ujema (obvezno, če dinamična_vrednost ni podana).

***dynamic_value***
*id* ciljne lastnosti, ki naj bo naveden kot vrednost lastnosti <switch>, če se trenutni pogoj ujema (obvezno, če ni podana fixed_value).

**<connect>**
Povezuje dve lastnosti. Lastnost odjemalca bo spremenjena vsakič, ko se spremeni lastnost guvernerja (vendar ne obratno). Lastnosti:

***client***
ID lastnosti stranke, tj. lastnost, ki bo prilagojena (obvezno)

***governor***
ID lastnosti guvernerja, tj. lastnost, ki bo prilagodila lastnost stranke. To lahko vključuje modifikator (obvezno)

***reconcile***
Če je ″true″, bo lastnost odjemalca prilagodila lastnosti guvernerja ob povezavi tako, da bo lastnost guvernerja sprejemala le vrednosti, ki jih sprejema tudi odjemalec (npr. predpostavimo, da je regulator številska lastnost z najmanjšo vrednostjo ″0″ ″, odjemalec pa je številska lastnost z najmanjšo vrednostjo ″100″. Najmanjša vrednost obeh lastnosti bo prilagojena na 100, če je reconcile=″true″). Na splošno deluje samo za lastnosti istega osnovnega tipa (neobvezno, privzeto je ″false″)

**<dependency_check>**

Ustvari boolovo lastnost, ki je true, če so podane odvisnosti izpolnjene, sicer pa false. Sintaksa elementa XML je enaka sintaksi elementa **<dependencies>**, ki je opisan v referenci `.pluginmap`. Od RKWard 0.6.1, samo RKWard in R upoštevane so specifikacije različice, ne pa odvisnosti od paketov ali pluginmap.

**<skript>**

Definirajte kodo skripta za nadzor logike uporabniškega vmesnika. Za podrobnosti glejte razdelek o skriptirani logiki GUJI. Kodo skripta za zagon je mogoče podati z uporabo atributa *″file″* ali kot (komentirano) besedilo elementa. Element **<script>** ni dovoljen v razdelku **<logic>** nabora možnosti optionset. Lastnosti:

***file***
Ime datoteke skriptne datoteke. (obvezno)

## A.4 Lastnosti elementov vtičnika

Vsi elementi postavitve in vsi aktivni elementi imajo naslednje lastnosti, dostopne prek ″id_of_element.name_of_property″:

**visible**
Ali je GUI element je viden ali ne (boolean)

**enabled**
Ali je GUI element je omogočen ali ne (boolean)

**required**
Ali je GUI zahtevan (za veljavno nastavitev) ali ne. Upoštevajte, da je vsak element, ki je onemogočen ali skrit, tudi implicitno neobvezen (boolean).

Poleg tega imajo nekateri elementi dodatne lastnosti, s katerimi se lahko povežete. Večina aktivnih elementov ima tudi lastnost privzeto ″default″, katere vrednost bo vrnjena pri klicih `getBoolean/getString/getList (″...″)`, če nobena posebna lastnost ni bila imenovana, kot je opisano spodaj.

**<text>**
Privzeta lastnost je besedilo

**text**
Prikazano besedilo (besedilo)

**<varselector>**
Ni privzete lastnosti

**selected**
Trenutno izbrani predmeti. Verjetno tega ne želite uporabiti. Uporablja se interno (RObject)

**root**
Korenski/nadrejeni objekt predmetov, ponujenih za izbor (RObject)

**<varslot>**
Privzeta lastnost je ″available″

**available**
Vsi predmeti, shranjeni v reži varslot (RObject)

**selected**
Od predmetov, shranjenih v varslot, tisti, ki so trenutno izbrani. Verjetno tega ne želite uporabiti. Uporablja se interno (RObject)

**source**
Kopija predmetov, izbranih v ustreznem izbirniku spremenljivk varselector. Verjetno tega ne želite uporabiti. Uporablja se interno (RObject)

**<valueselector>**
Privzeta lastnost je ″selected″

**selected**
Trenutno izbrani nizi. Modifikator ″labeled″ za pridobitev ustreznih oznak. V <valueselector> tega verjetno ne želite uporabiti neposredno (samo v <select>). (read/write StringList)

**available**
Seznam vrednosti niza, med katerimi lahko izbirate. (read/write StringList)

**labels**
Oznake za prikaz vrednosti niza. (read/write StringList)

**<valueslot>**
Enako kot <varslot>, vendar so lastnosti seznami nizov namesto RObjects.

**<radio>**
Privzeta lastnost je ″string″

**string**
Vrednost trenutno izbrane možnosti (string)

**number**
Število trenutno izbrane možnosti (možnosti so oštevilčene od zgoraj navzdol, z začetkom pri 0) (celo število)

**<dropdown>**
Enako kot <radio>

**<select>**
Enako kot <valueselector>

**<option>**
Ni privzete lastnosti. ″enabled″ je *edina* lastnost in trenutno ni na voljo za možnosti znotraj <select> ali <valueselector>. <option> nima ″visible″ ali ″required″ lastnosti.

**enabled**
Ali naj bo ta posamezna možnost omogočena ali onemogočena. V večini primerov boste omogočili/onemogočili celoten <radio< ali namesto tega <dropdowni<. Toda to je mogoče uporabiti za dinamično nastavitev omogočenosti posamezne možnosti znotraj <radio< ali <dropdowni< (bool)

**<checkbox>**
Privzeta lastnost je ″state.labeled″, kar pomeni, da so vrnjene vrednosti, določene z atributoma `value` in `value_unchecked`, *ne* prikazane oznake potrditvenega polja.

**state**
Stanje potrditvenega polja (vklopljeno ali izklopljeno). Upoštevajte, da so uporabni modifikatorji te lastnosti (kot vseh logičnih lastnosti) ″not″ in ″labeled″ (glejte vrste lastnosti). Vendar je pogosto najbolj uporabno povezati se z lastnostjo brez modifikatorja, tj. *″checkbox_id*.state″, ki bo vrnil stanje potrditvenega polja v obliki, primerni za uporabo v stavku if (0 ali 1). (boolova vrednost)

**<frame>**
Privzeta lastnost je ″checked″, če - in samo če - je okvir mogoče preveriti. Za okvirje, ki jih ni mogoče preveriti, ni privzete lastnosti.

**checked**
Na voljo samo za okvirje, ki jih je mogoče preveriti: stanje potrditvenega polja (vklopljeno ali izklopljeno). Upoštevajte, da sta uporabna modifikatorja te lastnosti (kot vseh logičnih lastnosti) ″not″ in ″numeric″ (glejte vrste lastnosti). (boolova vrednost)

**<input>**
Privzeta lastnost je ″text″

**text**
Trenutno besedilo v vnosnem polju (niz)

**<matrix>**
Privzeta lastnost je ″cbind″.

**rows**
: Število vrstic v matriki (celo število). Če matrika uporabniku omogoča dodajanje / odstranjevanje vrstic, je treba to lastnost obravnavati kot samo za branje. V nasprotnem primeru bo sprememba spremenila velikost matrike.

**columns**
: Število stolpcev v matriki (celo število). Če matrika dovoljuje uporabniku dodajanje/odstranjevanje stolpcev, je treba to lastnost obravnavati kot samo za branje. V nasprotnem primeru bo sprememba spremenila velikost matrike.

**tsv**
: Podatki v matriki v formatu tsv (string; branje-pisanje). Upoštevajte, da so v primerjavi z običajno postavitvijo tsv stolpci *columns*, ne vrstice, ločeni z znaki za novo vrstico, celice v stolpcu pa so ločene z znaki tabulatorja.

**0,1,2...**
: Podatki iz enega samega stolpca (0 za skrajno levi stolpec). `getValue()`/`getString()` to vrne kot en sam niz, ločen z ″\n″. Vendar pa je priporočeni način za to uporaba `getList()`, ki ta stolpec vrne kot niz nizov.

**row.0,row.1,row.2...**
: Podatki iz ene same vrstice (0 za najvišjo vrstico). `getValue()`/`getString()` to vrne kot en sam niz, ločen z ″\n″. Vendar pa je priporočeni način za to uporaba `getList()`, ki vrne to vrstico kot niz nizov.

**cbind**
: Podatki v formatu, primernem za lepljenje v R, zaviti v stavek cbind (niz; samo za branje).

**<optionset>**

Ni privzete lastnosti.

**row_count**
: Število elementov v naboru možnosti optionset (celo število). Samo za branje.

**current_row**
: Trenutno aktivni element v naboru možnosti optionset (celo število). -1 za noben aktivni element. Branje-pisanje.

***optioncolumn_ids***
: Za vsak <optioncolumn>, ki ga definirate, bo ustvarjena lastnost seznama nizov s podanim ID-jem.

**<browser>**

Privzeta lastnost je ″selection″

**selection**
: Trenutno besedilo (ime izbrane datoteke) v brskalniku (niz)

***prepiši***
: Ali je potrjena možnost ″prepiši″ (logična vrednost, samo za branje, tj. lahko stanje potrditvenega polja samo preberete, programsko pa ga ne morete spremeniti)

**<saveobject>**

Privzeta lastnost je ″selection″

**selection**
: Polno ime izbranega predmeta (niz; samo za branje – če želite to nastaviti programsko, uporabite ″parent″ in ″objectname″)

**parent**
: Nadrejeni predmet izbranega predmeta. To je vedno obstoječi R objekt vrste, ki lahko vsebuje druge objekte (npr. seznam ali data.frame). Ko je nastavljen na prazen niz ali neveljaven predmet, se predpostavlja ″.GlobalEnv″ (RObject)

**objectname**
Osnovno ime izbranega predmeta, tj. niz, ki ga je vnesel uporabnik (po potrebi spremenjen v veljavno R ime) (niz)

**active**
Samo za objekte shranjevanja, ki jih je mogoče preveriti: ali je kontrolnik označen/omogočen. Vedno velja za predmete shranjevanja, ki jih ni mogoče preveriti (bool)

**<spinbox>**
Privzeta lastnost je ″int″ ali ″real.formatted″, odvisno od načina vrtilnega polja

**int**
Vrednost celega števila, ki jo hrani vrtilno polje, ali najbližje celo število, če je v realnem načinu (celo število)

**real**
Realna vrednost, ki jo hrani vrtilno polje (in celo število, če je celo število) (realno)

**<formula>**
Privzeta lastnost je ″model″

**model**
Trenutni niz modela (niz)

**table**
Data.frame, ki vsebuje zahtevane spremenljivke. Če so uporabljene spremenljivke samo iz enega data.frame, se vrne ime tega data.frame. V nasprotnem primeru se po potrebi sestavi nov data.frame (niz)

**labels**
Če so vključene spremenljivke iz več data.frameov, se lahko njihova imena pokvarijo (na primer, če oba data.frames vsebujeta spremenljivko z imenom ″x″). To vrne seznam s pokvarjenimi imeni kot indeksi in opisno oznako kot vrednost (niz)

**fixed_factors**
Fiksni faktorji. Verjetno tega ne želite uporabiti. Uporablja se interno (RObject)

**dependent**
Odvisna spremenljivka(e). Verjetno tega ne želite uporabiti. Uporablja se interno (RObject)

**<embed>**
Ni privzete lastnosti

**code**
Koda, ki jo ustvari vdelani vtičnik (code)

**<preview>**
Privzeta lastnost je ″state″

**state**
Ali je polje za predogled potrjeno (ni nujno, ali je bil predogled že prikazan) (boolean)

**<convert>**
Ta element (uporabljen v razdelku <logic>) je poseben, ker je tehnično *je* lastnost, namesto da samo drži eno ali več lastnosti. Je boolove vrste. Upoštevajte, da sta uporabna modifikatorja te lastnosti (kot vseh boolovih lastnosti) ″not″ in ″numeric″ (glejte vrste lastnosti)

**<switch>**
Ta element (uporabljen v razdelku <logic>) je poseben, ker je tehnično *je* lastnost (niz), namesto da samo drži eno ali več lastnosti. Omogoča preklapljanje med več ciljnimi lastnostmi glede na vrednost lastnosti pogoja ali ponovno preslikavo vrednosti lastnosti pogoja. Vsi modifikatorji, ki jih podate, se posredujejo ciljnim lastnostim, tako npr. če so vse ciljne lastnosti lastnosti RObject, lahko na stikalu uporabite tudi modifikator ″shortname″. Če pa so ciljne lastnosti različnih vrst, lahko uporaba modifikatorjev povzroči napake. Za `fixed_value`s je vsak modifikator tiho opuščen. Upoštevajte, da so ciljne lastnosti, ko do njih dostopate prek stikala, vedno samo za branje.

## A.5 Vstavljivi vtičniki, dobavljeni z uradno izdajo RKWard

Številni vtičniki, ki jih je mogoče vdelati, so priloženi RKWard in jih lahko uporabite v svojih lastnih vtičnikih. Podrobna dokumentacija je trenutno na voljo samo v teh izvornih datotekah vtičnikov ali v datotekah pomoči. Vendar pa je tukaj seznam za hiter pregled tega, kar je na voljo:

| ID | Pluginmap | Opis | Primer uporabe |
|---|---|---|---|
| rkward::plot_options | embedded.pluginmap | Zagotavlja široko paleto možnosti za grafikone. Večina vtičnikov za risanje uporablja to. | Plots->Barplot, večina drugih vtičnikov za risanje |
| rkward::color_chooser | embedded.pluginmap | Zelo preprost vtičnik za določanje barve. Trenutna izvedba ponuja seznam imen barv. Prihodnje izvedbe lahko zagotovijo bolj dovršeno izbiranje barv. | Grafi->Histogram |
| rkward::plot_stepfun_options | embedded.pluginmap | Možnosti izrisa stopenjske funkcije | Parcele->risba ECDF |
| rkward::histogram_options | embedded.pluginmap | Možnosti histograma (risbe) | Grafi->Histogram |
| rkward::barplot_embed | embedded.pluginmap | Možnosti stolpičnega grafa | Grafikoni->Stolpični grafikon |
| rkward::one_var_tabulation | embedded.pluginmap | Zagotavlja tabeliranje ene same spremenljivke. | Grafikoni->Stolpični grafikon |
| rkward::limit_vector_length | embedded.pluginmap | Omejite dolžino vektorja (na n največjih ali najmanjših elementov). | Grafikoni->Stolpični grafikon |
| rkward::level_select | embedded.pluginmap | Zagotavlja <valueselector> napolnjen z ravnmi (ali edinstvenimi vrednostmi) vektorja. | Podatki->Prekodiraj kategorične podatke |
| rkward::multi_input | embedded.pluginmap | Združuje vrtilno polje, vnosne in radijske kontrole za zagotavljanje vnosa znakovnih, številskih in logičnih podatkov. | Podatki->Prekodiraj kategorične podatke |

Tabela A.1: Standardni vstavljivi vtičniki

## A.6 Elementi za uporabo v datotekah `.pluginmap`

**<document>**

Prisoten mora biti v vsaki datoteki `.pluginmap` kot korensko vozlišče (natanko enkrat). Lastnosti:

**base_prefix**

Imena datotek, navedena v datoteki `.pluginmap`, so pričakovano relativne glede na imenik datoteke `.pluginmap` + predpono, ki jo navedete tukaj. Uporabno, zlasti če se vse vaše komponente nahajajo pod enim samim podimenikom.

**namespace**

Imenski prostor za ID-je komponent. Ko iščete komponente za vdelavo, jih bo mogoče pridobiti prek niza ″namespace::component_id″. Zaenkrat nastavite na ″rkward″.

**id**

Izbirni identifikatorski niz za ta `.pluginmap`. Če navedete to možnost, lahko tretji avtorji sklicujejo in naložijo vašo datoteko `.pluginmap` od njihovih (glejte poglavje o ravnanju z odvisnostmi).

**priority**

En od *˜hidden˜*, *˜low˜*, *˜medium˜* ali *˜high˜*. `.pluginmap` s prioriteto ″medium″ ali ″high″ se aktivirajo samodejno, ko jih RKWard prvič najde. Uporabite *priority=˜hidden˜* za `.pluginmap`, ki niso namenjeni za aktiviranje, imenik (namenjen samo za vključitev). V trenutni izvedbi pa to dejansko ne skrije `.pluginmap`. (Izbirno, privzeto je ″medium″).

**<dependencies>**

Ta element, ki določa odvisnosti, je dovoljen kot neposredni podrejeni element elementa <document> (enkrat) in kot podrejeni element <component> elementov (enkrat za vsak element <component>). Podaja odvisnosti, ki morajo biti izpolnjene za uporabo vtičnika(-ov). Za pregled glejte poglavje o odvisnostih. Lastnosti:

**rkward_min_version, rkward_max_version**

Najmanjša in največja dovoljena različica RKWard. Specifikacije različice lahko vključujejo neštevilske pripone, kot je ″0.5.7z-devel1″. Če določena odvisnost ni izpolnjena, bodo vtičniki, na katere se nanaša, *prezrti*. Več informacij. Neobvezno; če ni navedeno, ni najmanjše/največje različice RKWard bo zahtevano.

**R_min_version, R_max_version**

Najmanjša in največja dovoljena različica R. Specifikacije različice morda *ne* vključujejo neštevilske pripone, kot je ″0.5.7z-devel1″. R odvisnost od različice bo prikazana na straneh s pomočjo vtičnikov, vendar nima neposrednega učinka, saj od RKWard 0.6.1. Več informacij. Neobvezno; če ni navedeno, ni najmanjše/največje različice R bo obvezno.

**platforme**

Platforme, kjer je ta vtičnik na voljo. Podprte vrednosti so *˜unix˜*, *˜windows˜*, *˜macos˜*, *˜any˜* in kombinacije ločene z dvopičjem (npr. *˜unix:macos˜*). *˜unix˜* vključuje vse različice Linuxa in BSD, vendar *ne* MacOS. Če vaš vtičnik ni odvisen od platforme, preprosto izpustite ta atribut.

Podrejeni elementi:

**<package>**

Doda odvisnost od določenega R paketa. Lastnosti:

**name**

Ime paketa (obvezno).

**min_version, max_version**
Najmanjša / največja dovoljena različica (neobvezno).

**repository**
Repozitorij, kjer je paket mogoče najti. Izbirno, vendar zelo priporočljivo, če paket ni na voljo na CRAN.

**<pluginmap>**
Doda odvisnost od določene RKWard `.pluginmap`. Lastnosti:

**name**
Id niz zahtevane `.pluginmap` (obvezno).

**min_version, max_version**
Najmanjša / največja dovoljena različica (neobvezno).

**url**
URL kjer je `.pluginmap` mogoče najti. Obvezno.

**<about>**
Lahko je prisoten natanko enkrat kot neposredni podrejeni element elementa <document>. Vsebuje meta informacije o `.pluginmap` (ali o vtičniku). Za pregled si oglejte poglavje o informacijah 'o'. Lastnosti:

**name**
Uporabniku vidno ime. Neobvezno. Ni nujno, da je enak "id".

**version**
Številka različice. Neobvezno. Oblika zapisa ni omejena, vendar zaradi varnosti upoštevajte običajne sheme za ustvarjanje različic, kot je "x.y.z".

**releasedate**
Specifikacija datuma izdaje. Izbirno v obliki "YYYY-MM-DD".

**shortinfo**
*short* opis vtičnika / `.pluginmap`. Neobvezno.

**url**
URL kjer lahko najdete več informacij. Neobvezno, vendar priporočljivo.

**copyright**
Specifikacija avtorskih pravic, npr. "2012-2013 avtor John Doe". Neobvezno, vendar priporočljivo.

**licence**
Specifikacija licence, npr. "GPL" ali "BSD". Svojim datotekam ne pozabite priložiti popolne kopije ustrezne licence. Neobvezno, vendar priporočljivo.

**category**
Kategorija vtičnikov, npr. ""Item response theory". Od RKWard 0.6.1, nobena kategorija ni vnaprej določena. Neobvezno.

Podrejeni elementi:

**<author>**
Doda podatke o avtorju. Lastnosti:

**ime, ime2, družina**
Bodisi navedite polno ime za *`name`* ali pa navedite tako *`given`* kot *`family`* ločeno.

**role**
Opis vloge avtorja (neobvezno).

**email**
Elektronski naslov, na katerega je avtor dosegljiv. Obvezno. Lahko se nastavi na poštni seznam rkward-devel, če ste naročeni in je vaš vtičnik namenjen vključitvi v uradno izdajo RKWard.

**url**
URL z več podatki o avtorju, npr. domača stran (neobvezno).

**<components>**

Prisoten mora biti natanko enkrat kot neposredni podrejeni element <document> element. Vsebuje posamezne elemente <component>, opisane spodaj. Brez atributov.

**<component>**

Ena ali več elementov <component> je treba podati kot neposredne podrejene elemente <components> element (in samo tam). Registrira komponento/vtičnik z rkward. Lastnosti:

**type**
: Za prihodnjo razširitev: Vrsta komponente/vtičnika. Zaenkrat vedno nastavljen na ″standard″ (edina trenutno podprta vrsta).

**id**
: ID, po katerem je mogoče pridobiti to komponento (za postavitev v meni (glejte spodaj) ali za vdelavo). Glejte <document>-namespace zgoraj.

**file**
: Zahtevano vsaj za komponente type=″standard″: ime datoteke XML datoteka, ki opisuje GUI.

**label**
: Oznaka za to komponento, ko je postavljena v hierarhijo menija.

**neobvezno**
: Smiselno samo za komponente z definiranimi odvisnostmi: Običajno velja za napako, ki jo je treba prijaviti, če komponenta ni združljiva s to različico RKWard. Vendar, če komponenta v trenutnem okolju res ni potrebna, nastavitev tega atributa na `″true″` privzeto zavrne vsa opozorila (`″false″`).

**<attribute>**

Določa atribut komponente. Zaenkrat smiselno samo za vtičnike za uvoz. Dovoljeno samo kot neposredni podrejeni <component>. Lastnosti:

**id**
: ID atributa

**value**
: Vrednost atributa

**labels**
: Oznaka, povezana z atributom

**<hierarchy>**

Prisoten mora biti natanko enkrat kot neposredni podrejeni element <document>. Opisuje, kje v hierarhiji menija naj bodo postavljene zgoraj navedene komponente. Sprejme samo elemente <menu> kot neposredno podrejene. Brez atributov.

**<menu>**

En ali več elementov <menu> je treba podati kot neposredne potomce elementa <hierarchy>. Razglasi nov (pod)meni. Če meni z danim ID-jem (glejte spodaj) že obstaja, se oba menija združita. Element <menu> je dovoljen kot neposredni podrejeni element <hierarchy> element (meni najvišje ravni) ali kot neposredni podrejeni element katerega koli drugega elementa <menu> (podmeni). Nasprotno pa element <menu> sprejme druge elemente <menu> ali elemente <entry> kot podrejene. Lastnosti:

**id**
: Identifikacijski niz menija. Uporabno, ko so definicije menija prebrane iz več datotek `.pluginmap`, da zagotovite, da je vtičnike mogoče postaviti v iste menije. Nekateri ID-ji menijev, kot je ″file″, se nanašajo na vnaprej določene menije (v tem primeru meni ″File″). Ne pozabite preveriti z obstoječo datoteko `.pluginmap` za uporabo doslednih ID-jev.

**label**
: Oznaka za meni.

**group**

Omogoča nadzor vrstnega reda vnosov v meniju. Glejte razvrstitev elementov menija. Neobvezno.

**<entry>**

Vnos v meni, tj. možnost menija za priklic vtičnika. Lahko se uporablja samo kot neposredni podrejeni element <menu> elementa, ne sprejema podrejenih elementov. Lastnosti:

**component**

ID komponente, ki jo je treba priklicati, ko je aktiviran ta menijski vnos.

**group**

Omogoča nadzor vrstnega reda vnosov v meniju. Glejte razvrstitev elementov menija. Neobvezno.

**<group>**

Razglasi skupino elementov v meniju. Glejte razvrstitev elementov menija. Lastnosti:

**id**

Ime te grupe.

**separated**

Neobvezno. Če je nastavljeno na "true", bo element v tej grupi vizualno ločen od okoliških elementov.

**group**

Ime grupe, ki ji želite dodati to grupo (neobvezno).

**<context>**

Razglasi vnose v context. Dovoljeno samo kot neposredni podrejeni element oznake <document>. Sprejme samo oznake <menu> kot neposredno podrejene. Lastnosti:

**id**

ID konteksta. Zaenkrat sta implementirana samo dva konteksta: "x11" in "import".

**<require>**

Vključi še eno datoteko `.pluginmap`. Ta datoteka `.pluginmap` se naloži samo enkrat, tudi če je zahtevana - <require>d iz več drugih datotek. Najpomembnejši primer uporabe je vključitev datoteke pluginmap, ki deklarira nekatere komponente, ki so vdelane s komponentami, deklariranimi v tem `.pluginmap`. Elementi <require> so dovoljeni samo kot neposredno podrejeni vozlišča <document>. Lastnosti:

**file**

Ime datoteke vključeno v `.pluginmap`. To je vidno glede na imenik trenutne datoteke `.pluginmap` + base_prefix (glejte zgoraj, element <document>). Če ne poznate relativne poti do datoteke `.pluginmap` če jo želite vključiti, uporabite atribut *`map`*, da se nanj sklicujete z ID-jem.

**map**

Če želite vključiti `.pluginmap` datoteke iz drugega paketa (ali RKWard `.pluginmap` iz vaše zunanje `.pluginmap`), se lahko nanjo sklicujete po njenem *`namespacename::id`*, kot je določeno v zahtevanem elementu `.pluginmap` <document>. Vključitev ne bo uspela, če `.pluginmap` s tem ID-jem ni znan (npr. ni nameščen v uporabnikovem sistemu). To metodo uporabite samo za vključitev `.pluginmap` zunaj vašega paketa. Za mape v vašem paketu je podajanje relativne poti (atribut *`file`*) hitrejše in zanesljivejše.

## A.7 Elementi za uporabo v datotekah .rkh (pomoč)

**<document>**

Prisoten mora biti v vsaki datoteki `.xml` kot korensko vozlišče (natančno enkrat). Brez atributov.

**<title>**

Naslov strani s pomočjo. To se *ne* interpretira za strani s pomočjo za vtičnik (to vzame naslov iz samega vtičnika), samo za samostojne strani. Brez atributov. Besedilo v <title> bo postalo naslov strani s pomočjo. Lahko se definira samo enkrat, kot neposredni podrejeni element vozlišča <document>.

**<summary>**

Kratek povzetek strani s pomočjo (ali za kaj se ta vtičnik uporablja). To bo vedno prikazano na vrhu strani s pomočjo. Brez atributov. Besedilo v oznaki <summary> bo prikazano. Priporočeno, vendar ni obvezno. Lahko se definira samo enkrat, kot neposredni podrejeni element vozlišča <document>.

**<usage>**

Nekoliko bolj podroben povzetek uporabe. To bo vedno prikazano neposredno za <summary>. Brez atributov. Besedilo v oznaki <usage> bo prikazano. Priporočeno za strani s pomočjo vtičnikov, ni pa obvezno. Lahko se definira samo enkrat, kot neposredni podrejeni element vozlišča <document>.

**<section>**

Razdelek za splošne namene. Lahko se uporabi poljubno število krat kot neposredni podrejeni element vozlišča <document>. Ti razdelki so prikazani po vrstnem redu njihove definicije, vendar vsi *za* <usage> in *pred* razdelkom <settings>. Besedilo v znaki <section> bo prikazano.

**id**

Identifikator, potreben za skok na ta razdelek iz navigacijske vrstice (ali povezave). V datoteki mora biti edinstven. Zahtevano, ni privzeto.

**title**

Naslov (napis) tega razdelka. Zahtevano, ni privzeto.

**short_title**

Kratek naslov, primeren za prikaz v navigacijski vrstici. Neobvezno, privzeto je polni naslov.

**<settings>**

Določa razdelek, ki vsebuje reference na različne nastavitve GUI. Samo smiselno in uporabljeno samo za strani s pomočjo, povezane z vtičniki. Uporabi kot neposrednega podrejenega elementa <document>. Lahko vsebuje samo elemente <setting> in <caption> kot neposredne podrejene. Brez atributov.

**<setting>**

Razlaga posamezno nastavitev v GUI. Dovoljeno samo kot neposredni podrejeni element elementa <settings>. Prikaže se besedilo v elementu.

**id**

ID nastavitve v vtičniku `.xml`. Zahtevano, ni privzete vrednosti.

**title**

Neobvezni naslov za nastavitev. Če je izpuščen (v večini primerov je priporočljiva izpustitev), bo naslov vzet iz vtičnika `.xml`.

**<caption>**

Napis za vizualno združevanje več nastavitev. Lahko se uporablja le kot neposredni podrejeni element elementa <settings>.

**id**
ID ustreznega elementa (običajno <frame>, <page> ali <tab>) v vtičniku `.xml`.

**title**
Izbirni naslov za napis. Če je izpuščen (v večini primerov je priporočljiva izpustitev), bo naslov vzet iz vtičnika `.xml`.

**<related>**
Določa razdelek s povezavami do nadaljnjih povezanih informacij. Bo vedno prikazano za razdelkom <settings>. Brez atributov. Besedilo v oznaki <related> bo prikazano. Običajno bo to vsebovalo seznam v slogu HTML. Priporočeno za strani s pomočjo vtičnikov, ni pa obvezno. Lahko se definira samo enkrat, kot neposredni podrejeni element vozlišča <document>.

**<technical>**
Določa razdelek, ki vsebuje tehnične informacije, ki niso pomembne za končne uporabnike (kot je notranja struktura vtičnika). Vedno bo prikazan zadnji na strani s pomočjo. Brez atributov. Besedilo v oznaki <related> bo prikazano. Ni potrebno in ni priporočljivo za večino strani s pomočjo vtičnikov. Lahko se definira samo enkrat, kot neposredni podrejeni element vozlišča <document>.

**<link>**
Povezava. Lahko se uporablja v katerem koli od zgoraj opisanih razdelkov.

**href**
Ciljni URL. Upoštevajte, da je na voljo več specifičnih URL za RKWard. Za podrobnosti si oglejte razdelek o pisanju strani za pomoč.

**<label>**
Vstavi vrednost oznake uporabniškega vmesnika. Lahko se uporablja v katerem koli od zgoraj opisanih razdelkov.

**id**
ID elementa v vtičniku, za katerega želite kopirati atribut *`label`*.

**<various HTML tags>**
Najbolj osnovne oznake HTML so dovoljene v razdelkih. Prosimo, da minimalno uporabljate ročno oblikovanje.

## A.8 Funkcije, ki so na voljo za GUI logično skriptiranje

**Razred "Component"**
Razred, ki predstavlja posamezno komponento ali lastnost komponente. Najpomembnejši pojavek tega razreda je spremenljivka "gui", ki je vnaprej določen kot korenska lastnost trenutne komponente. Za pojavke razreda "Component" so na voljo naslednje metode:

**absoluteId(base_id)**
Vrne absolutni ID *base_id* ali – če je base_id izpuščen – identifikator komponente.

**getValue(id)**
Odsvetovano. Namesto tega uporabite `getString(), getBoolean() ali getList()`. Vrne vrednost dane podrejene lastnosti. Vrne vrednost te lastnosti, če je ID izpuščen.

**getString(id)**
Vrne vrednost dane podrejene lastnosti kot niz. Vrne vrednost te lastnosti, če je ID izpuščen.

**getBoolean(id)**
Vrne vrednost dane podrejene lastnosti kot boolovo vrednost (če je mogoče). Vrne vrednost te lastnosti, če je ID izpuščen.

**getList(id)**
Vrne vrednost podane podrejene lastnosti kot matriko nizov (če je mogoče). Vrne vrednost te lastnosti, če je ID izpuščen.

**setValue(id, value)**
Nastavite vrednost dane podrejene lastnosti na *value*.

**getChild(id)**
Vrne pojavek podrejene lastnosti z danim *id*.

**addChangeCommand(id, command)**
Izvedi *command* vsakič, ko se spremeni podrejena lastnost, podana z *id*.

**Razred "RObject"**

Razred, ki predstavlja en predmet v R. Pojavek tega razreda lahko dobite z uporabo **makeRObject(objectname)**. Za pojavke razreda "RObject" so na voljo naslednje metode:

> POZOR
> Če so kateri koli ukazi še vedno na čakanju v ozadju, so lahko informacije, ki jih zagotovijo te metode, zastarele do trenutka, ko se zažene koda vtičnika. *Ne* zanašajte se nanj pri kritičnih operacijah (tveganje izgube podatkov).

**getName()**
Vrne absolutno ime predmeta.

**exists()**
Vrne, ali predmet obstaja. Na splošno morate to preveriti, preden uporabite katero koli od spodaj navedenih metod.

**dimensions()**
Vrne niz dimenzij (podobno kot **dim()** v R).

**classes()**
Vrne matriko razredov (podobno kot **class()** v R).

**isClass(class)**
Vrne true, če je objekt razreda *class*.

**isDataFrame()**
Vrne true, če je objekt data.frame.

**isMatrix()**
Vrne true, če je objekt matrix.

**isList()**
Vrne true, če je objekt seznam.

**isFunction()**
Vrne true, če je objekt funkcija.

**isEnvironment()**
Vrne true, če je objekt okolje.

**isDataNumeric()**
Vrne true, če je objekt vektor številskih podatkov.

**isDataFactor()**
Vrne true, če je objekt vektor faktorskih podatkov.

**isDataCharacter()**
Vrne true, če je objekt vektor znakovnih podatkov.

**isDataLogical()**
Vrne true, če je objekt vektor logičnih podatkov.

**parent()**
Vrne pojavek "RObject", ki predstavlja nadrejenega elementa tega predmeta.

**child(childname)**
Vrne pojavek "RObject", ki predstavlja podrejeno *childname* tega predmeta.

**Razred "RObjectArray"**

Niz primerkov RObject. Primerek tega razreda lahko dobite z uporabo **makeRObjectArray(objectnames)**. Še posebej je uporabno pri delu z varsloti, ki omogočajo izbiro več predmetov.

**include()-function**

**include(ime datoteke)** lahko uporabite za vključitev ločene datoteke JS.

**doRCommand()-function**

**doRCommand(command, callback)** se lahko uporablja za poizvedbo R za informacijo. Preberite razdelek o poizvedovanju R znotraj vtičnika za podrobnosti in opozorila.

# Dodatek B

# Odpravljanje težav med razvojem vtičnika

Torej ste prebrali vso dokumentacijo, naredili vse pravilno, pa še vedno ne morete spraviti v delo? Brez skrbi, uredili bomo. Prva stvar, ki jo morate storiti, je: Aktivirajte **RKWard Sporočila o odpravljanju napak** - okno (na voljo v meniju **Windows** - ali z desnim klikom na eno od orodnih vrstic) in nato znova zaženite svoj vtičnik. Splošno pravilo je, da v oknu s sporočili ne smete videti nobenega izhoda, ko je priklican vaš vtičnik ali kadar koli drugje. Če obstaja, je verjetno povezan z vašim vtičnikom. Poglejte, če vam pomaga.

Če se zdi, da je na konzoli vse v redu, poskusite povečati raven odpravljanja napak (v ukazni vrstici z uporabo **rkward --debug-level 3** ali tako, da nastavite raven odpravljanja napak na 3 v **Nastavitve** → **Konfiguriraj RKWard** → **Odpravljanje napak**). Ni nujno, da vsa sporočila, prikazana na višjih ravneh odpravljanja napak, kažejo na težavo, vendar je verjetno, da se vaša težava pojavi nekje med sporočili.

Če še vedno ne morete ugotoviti, kaj je narobe, ne obupajte. Vemo, da je to zapletena stvar in – navsezadnje – morda ste tudi vi naleteli na napako v RKWard in RKWard je treba popraviti. Samo pišite na poštni seznam za razvoj in nam povejte o težavi. Z veseljem vam bomo pomagali.

Nazadnje, tudi če ste izvedeli, kako to storiti sami, vendar ste ugotovili, da dokumentacija ni preveč koristna ali celo napačna v nekaterih pogledih, nam to sporočite tudi na poštnem seznamu, da bomo lahko popravili/izboljšali dokumentacijo.

# Dodatek C

# Licenca

Prevod: Matjaž Jeranmatjaz.jeran@amis.net