Introduktion till att skriva insticksprogram för RKWard

Thomas Friedrichsmeier
Meik Michalke
Översättare: Stefan Asserhäll

Introduktion till att skriva instickspogram för RKWard

Innehåll

Tabeller

Sammanfattning

Det här är en handledning för att skriva instickspogram i RKWard.

Kapitel 1

Inledning

Det här dokumentet beskriver hur man skriver egna insticksprogram. Dokumentationen har växt sig stor med tiden. Låt inte det skrämma dig. Vi rekommenderar att läsa igenom de fyra grundstegen (enligt översikten nedan) för att få en grundidé om hur saker och ting fungerar. Därefter kanske du vill skumma igenom innehållsförteckningen för att se vilka avancerade ämnen som kan vara relevanta för dig.

För frågor och kommentarer, skriv gärna till RKWards e-postlista för utveckling.

Du behöver inte läsa det här för att använda RKWard. Dokumentet handlar om att utöka RKWard. Det är riktat till avancerade användare eller personer som är villiga att hjälpa till att förbättra RKWard.

Att skriva ett standardinsticksprogram är i grunden en process med fyra steg:

- Placera en ny åtgärd i menyhierarkin
- Beskriva utseendet och beteendet hos insticksprogrammets grafiska användargränssnitt
- Definiera hur R-kod ska skapas från inställningarna som användaren gör i det grafiska användargränssnittet
- Lägga till en hjälpsida för insticksprogrammet

De hanteras i tur och ordning.

Vissa avancerade koncept kan användas i de fyra stegen, men hanteras i separata kapitel för att hålla saker och ting enkla:

- Logik för grafiskt användargränssnitt
- Inbädda insticksprogram i insticksprogram
- Användbara koncept för att skapa många serier av liknande insticksprogram

Dessutom visar inga av kapitlen alla alternativ, utan bara grundkoncepten. En fullständig referens av alternativ tillhandahålls separat.

Kapitel 2

Förberedelse: Vad är insticksprogram i RKWard? Hur fungerar de?

Den första frågan man kan ställa sig är naturligtvis: Vilka delar av RKWards funktionalitet åstadkoms genom att använda insticksprogram? Eller: Vad kan insticksprogram göra?

Ett sätt att svara på det är: Avmarkera alla `.pluginmap`-filer under **Inställningar** → **Anpassa RKWard** → **Insticksprogram**, och se vad som saknas. Ett något mer hjälpsamt svar: De flesta verkliga statistikfunktioner som kan kommas åt via det grafiska användargränssnittet är förverkligade med insticksprogram. Du kan också skapa ganska flexibla grafiska användargränssnitt för alla typer av operationer med insticksprogram.

Den grundläggande paradigmen bakom insticksprogram i RKWard är den vi går igenom i det här dokumentet: En XML-fil beskriver hur det grafiska användargränssnittet ser ut. En ytterligare JavaScript-fil används för att skapa R-syntax från inställningarna i det grafiska användargränssnittet. Alltså behöver insticksprogram egentligen inte utföra några statistiska beräkningar. Istället skapar insticksprogram R-syntaxen som behövs för att utföra beräkningarna. R-syntaxen skickas sedan till R-bakgrundsprogrammet för utvärdering, och oftast visas ett resultat i utdatafönstret.

Läs vidare i följande kapitel för att se hur det görs.

Kapitel 3

Skapa menyalternativ

När ett nytt insticksprogram skapas, måste RKWard få reda på det. Den första saken att göra är alltså att skriva en `.pluginmap`-fil (eller ändra en befintlig). Formatet för en `.pluginmap` är XML. Jag leder dig igenom ett exempel (försäkra dig också naturligtvis om att RKWard är inställt att läsa in din `.pluginmap` med **Inställningar** → **Anpassa RKWard** → **Insticksprogram**):

> TIPS
> Efter att ha läst det här kapitlet, ta också en titt på paketet rkwarddev. Det tillhandahåller några R-funktioner för att skapa de flesta av RKWards XML-taggar åt dig.

```
<!DOCTYPE rkpluginmap>
```

Värdet doctype tolkas egentligen inte, men ställ in det till *"rkpluginmap"* ändå.

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

Egenskapen *base_prefix* kan användas om alla insticksprogram befinner sig i en gemensam katalog. Då kan man därmed utelämna katalogen från filnamnen angivna nedan. Det är säkert att låta den vara *""*.

Som du kommer att märka nedan, får alla insticksprogram en unik identifierare, *id*. Att använda *namespace* är ett sätt att organisera sådana id, och göra det mindre troligt att duplicerade identifierare skapas av misstag. Internt läggs namnrymden följt av '::' till före alla identifierare som anges i en `.pluginmap`. I allmänhet, om du avser att distribuera dina insticksprogram i ett R-paket, är det en god idé att använda paketnamnet som parametern *namespace*. Insticksprogram som levereras med den officiella distributionen av RKWard har *namespace="rkward"*.

Egenskapen *id* är valfri, men att ange id för din `.pluginmap` gör det möjligt för andra att låta sina `.pluginmap`:ar läsa in din `.pluginmap` automatiskt (se avsnittet om beroenden).

```
<components>
```

Komponenter? Talar vi inte om insticksprogram? Ja, men i framtiden kommer insticksprogram inte vara mer än en särskild klass av komponenter. Vad vi gör här är då att registrera alla komponenter/insticksprogram med RKWard. Låt oss ta en titt på en exempelpost:

```
<component type="standard" id="t_test_two_vars" file="t_test_two_vars.xml" ↩
    label="Two Variable t-Test" />
```

Först egenskapen *type*: Lämna den som *"standard"* för tillfället. Ytterligare typer är inte implementerade ännu. Vi har redan nämnt *id*. Varje komponent måste ha en unik identifierare (i

sin namnrymd). Välj en som är enkel att känna igen. Undvik mellanslag och specialtecken. De är hittills inte förbjudna, men kan ha särskilda betydelser. Med egenskapen *file* anger man var beskrivningen av själva insticksprogrammet finns. Det är relativt till katalogen där .pluginmap-filen finns, och till *base_prefix* ovan. Ge till sist komponenten en beteckning. Beteckningen var än insticksprogrammet placeras i menyn (eller i framtiden kanske också på andra ställen).

Typiskt innehåller .pluginmap-filen flera komponenter, så här är några fler:

```
<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, det var första steget. RKWard känner nu till att insticksprogrammen finns. Men hur aktiverar man dem? De måste läggas till i en menyhierarki:

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

Direkt under taggen **<hierarchy>** börjar man beskriva i vilken meny (**<menu>**) som insticksprogrammet ska finnas. Med raden ovan säger man att insticksprogrammet ska vara i menyn **Analysis** (inte nödvändigtvis direkt i den, utan i en undermeny). Menyn **Analysis** är standard i RKWard, så den behöver i själva verket inte skapas från början. Om den dock inte fanns ännu, skulle egenskapen *label* användas för att ge den sitt namn. Till sist, identifierar återigen *id* den här menyn (**<menu>**). Det behövs så att flera .pluginmap-filer kan placera sina insticksprogram i samma menyer. De gör det genom att leta efter en meny (**<menu>**) med angivet *id*. Om id inte ännu finns, skapas en ny meny. Annars läggs alternativen till i den befintliga menyn.

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

Egentligen samma sak här: Nu definierar vi en undermeny i menyn **Analysis**. Den ska heta **Means**.

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

Och en sista nivå i menyhierarkin: En undermeny i undermenyn **Väntevärden**.

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

Nu till sist är det menyn vi vill placera insticksprogrammet i. Taggen **<entry>** signalerar att det här är det verkliga värdet istället för en annan undermeny. Egenskapen *component* refererar till *id* som angavs i insticksprogrammet/komponenten ovan.

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

Om du har förlorat spåret: Det är en annan undermeny i menyn **Analys**. Se skärmbilden nedan. Vi hoppar över en del av det som inte syns, markerat med [...].

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

De är de slutliga alternativen synliga på skärmbilden nedan.

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

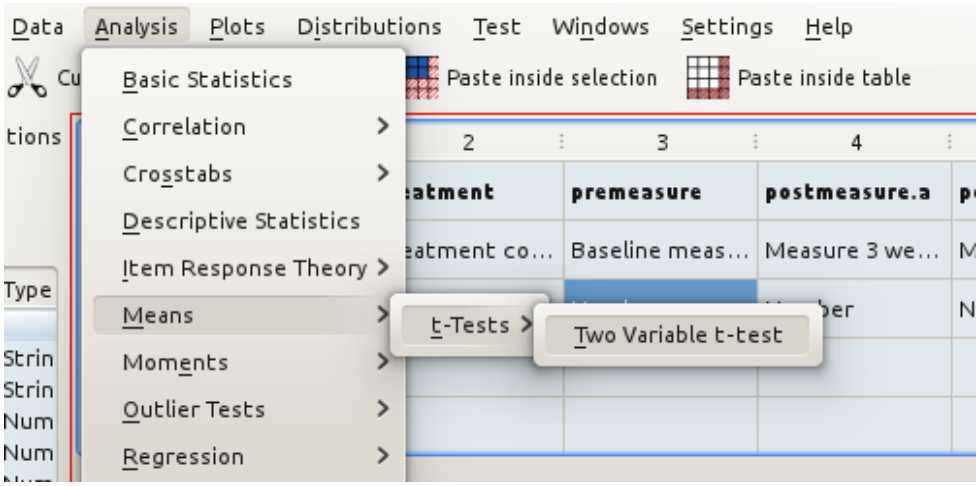
Det går naturligtvis också att placera insticksprogrammen i andra menyer än **Analys**.

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

Till och med i standardmenyer såsom **Arkiv**. Allt som behövs är rätt `id`.

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

Det är så man gör, och skärmbilden visar resultatet:



Förvirrad? Det enklaste sättet att komma igång är troligen att ta några av de befintliga `.pluginmap`-filerna som levereras med distributionen och ändra dem enligt dina behov. Dessutom, om du behöver hjälp, tveka inte att skriva till utvecklarnas e-postlista.

3.1 Bestämma menyalternativens ordning

Normalt sorteras automatiskt alla poster (alternativ, undermenyer) alfabetiskt inne i en meny. I *vissa* fall kan man vilja ha bättre kontroll. I sådana fall kan man gruppera element enligt det följande:

- Det går att definiera grupper i vilken meny som helst så här. Alla element som hör till samma grupp kommer att grupperas ihop:

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

- Om gruppen ska vara visuellt separerad från andra alternativ, använd:

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

- Alternativ, menyer och grupper kan läggas till sist i en angiven grupp genom att använda:

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

- Det är i själva verket också möjligt att definiera grupper (utan avskiljande linjer) implicit:

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

- Gruppnamn är specifika för varje meny. Grupp ″a″ i menyn ″Data″ ger exempelvis ingen konflikt med grupp ″a″ i menyn ″Analys″.
- Det vanligaste användarfallet är att definiera grupper längst upp eller längst ner i en meny. De fördefinierade grupperna ″top″ och ″bottom″ finns i alla menyer för det.
- Poster inom varje grupp sorteras alfabetiskt. Grupper visas i den ordning de deklareras (om de inte läggs till sist i en annan grupp naturligtvis).
- Menyer och alternativ utan gruppspecifikation utgör också logiskt en grupp (″″).

Kapitel 4

Definiera det grafiska användargränssnittet

4.1 Definiera en dialogruta

I föregående kapitel har du sett hur man registrerar ett insticksprogram i RKWard. Den viktigaste ingrediensen är att ange sökvägen till en XML-fil med en beskrivning av hur insticksprogrammet faktiskt ser ut. I detta kapitel lär du dig hur XML-filen skapas.

> TIPS
> Efter att ha läst det här kapitlet, ta också en titt på paketet rkwarddev. Det tillhandahåller några R-funktioner för att skapa de flesta av RKWards XML-taggar åt dig.

Återigen leder vi dig genom ett exempel. Exemplet är en (något förenklad) version av en tvåvariablers t-test.

```
<!DOCTYPE rkplugin>
```

Värdet doctype tolkas egentligen inte, ännu. Ställ in det till *rkplugin* ändå.

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

Alla insticksprogram genererar någon kod. För närvarande är det enda sättet att göra det att använda JS, som beskrivs i detalj i nästa kapitel. Det här definierar var JS-koden kan hittas. Filnamnet är relativt katalogen som insticksprogrammets XML finns i.

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

Det är oftast en god idé att också tillhandahålla en hjälpsida för insticksprogrammet. Filnamnet på hjälpsidan anges här, relativt till katalogen där insticksprogrammets XML-fil finns. Att skriva hjälpsidor är dokumenterat här. Utelämna raden om du inte tillhandahåller en hjälpfil.

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

Som du känner till, kan insticksprogram antingen ha ett dialog- eller guidegränssnitt, eller båda. Här börjar vi definiera ett dialoggränssnitt. Egenskapen *label* anger dialogrutans rubrik.

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

Element i det grafiska användargränssnittet kan organiseras med en flikbok. Här definierar vi en flikbok som det första elementet i dialogrutan. Använd **<tabbook>**[...]**</tabbook>** för att definiera flikboken och använd därefter **<tab>**[...]**</tab>** för varje sida i flikboken. Egenskapen *label* i elementet **<tab>** låter dig ange en rubrik för den sidan i flikboken.

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

Taggarna **<row>** och **<column>** anger utläggningen av elementen i det grafiska användargränssnittet. Här talar du om att du vill placera några element sida vid sida (vänster till höger). Egenskapen *id* är inte helt nödvändig, men vi använder den senare när vi lägger till ett guidegränssnitt i vårt insticksprogram. Det första elementet att placera i raden är:

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

Genom att använda den här enkla taggen skapar du en lista där användaren kan välja variabler. Du måste ange en *id* för elementet så att RKWard vet hur man hittar det.

> VARNING
> Det går inte att använda en punkt (.) i strängen *id*.

```
<column>
```

Därefter nästlar vi en kolumn, **<column>**, inne i raden. Det vill säga att följande element placeras ovanpå varandra (uppifrån och ner), och alla är till höger om **<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"/>
```

De här elementen är motsvarigheten till **<varselector>**. De representerar 'platser' där användaren kan placera variabler. Observera att *source* tilldelas samma värde som *id* i **<varselector>**. Det betyder att varje **<varslot>** tar sina variabler från en varselector. Varje **<varslot>** måste också ges en *id*. De kan ha en *label*, och de kan sättas till *required*. Det betyder att knappen **Submit** inte aktiveras förrän dess **<varslot>** har ett giltigt värde. Till sist tolkas inte egenskapen *type* ännu, men den kommer att användas för att försäkra att bara korrekta typer av variabler tillåts i en **<varslot>**.

Om du undrar över egenskapen *i18n_context*, finns den för att ge sammanhang för att hjälpa till att översätta ordet "against" på ett riktigt sätt, använt som rubrik för **<varslot>**, men det påverkar inte insticksprogrammets funktionalitet. Mer om det i ett separat kapitel.

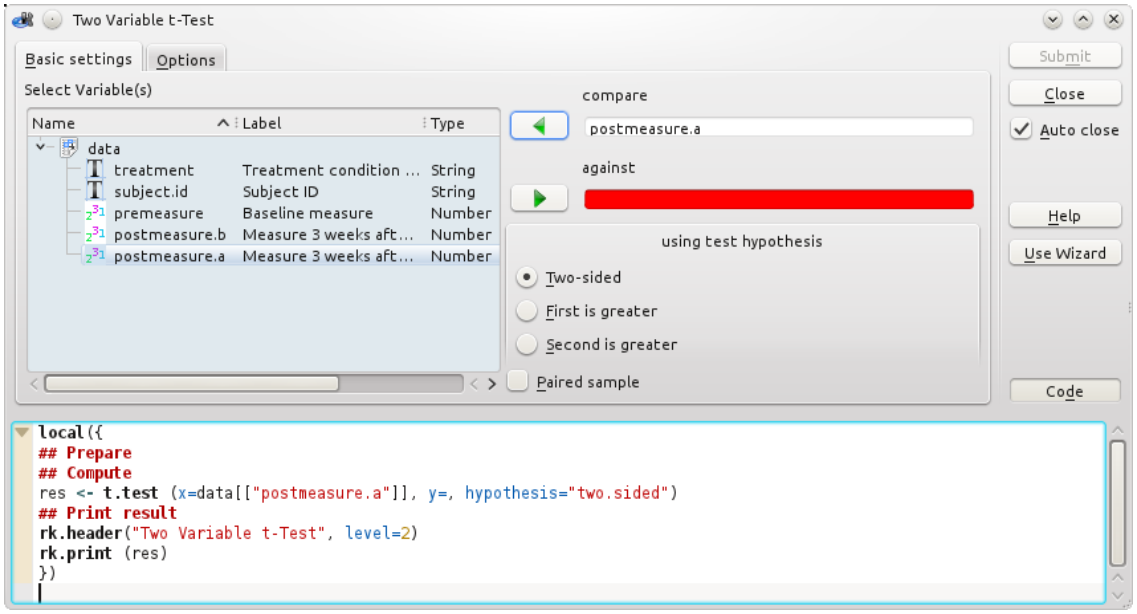
```
<radio id="hypothesis" label="using test hypothesis">
                                                        <option value="two. ↩
                                                            sided" label=" ↩
                                                            Two-sided"/>
              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

```
                                                    <option value="less ↩
                                                        " label="Second  ↩
                                                        is greater"/>
                                                </radio>
```

Här definieras en grupp alternativknappar med **<radio>**. Gruppen har en rubrik, *label*, och en *id*. Varje knapp, **<option>**, har en rubrik, *label* och är tilldelad ett värde, *value*. Det är värdet som elementet **<radio>** returnerar när alternativet väljes.

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

Varje tagg måste avslutas. Vi har lagt till alla element vi vill ha (två **<varslots>** och **<radio>**) i kolumnen **<column>**. Vi har lagt till alla element vi ville (**<varselector>** och **<column>** med de här elementen) i raden **<row>**. Och vi har lagt till alla elementen vi ville på första sidan i flikboken **<tabbook>**. Vi är inte klara med att definiera flikboken **<tabbook>** ännu (fler sidor tillkommer), och naturligtvis tillkommer det också mer i dialogrutan **<dialog>**. Men den här skärmbilden visar i stort vad vi har gjort så här långt:



Observera att vi inte har angivit knapparna **Skicka**, **Stäng**, etc. eller kodvyn. De här elementen genereras automatiskt. Men vi måste förstås också definiera den andra sidan i flikboken **<tabbook>**:

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

Normalt läggs element till uppifrån och ner som i en kolumn **<column>**. Eftersom det är vad vi vill här, behöver vi inte explicit ange radlayout med **<row>** eller kolumnlayout med **<column>**. Det första elementet vi definierar är en kryssruta. Precis som med **<radio><option>**, har kryssrutan en rubrik *label* och ett värde *value*. Värdet *value* är det som returneras om kryssrutan markeras. Naturligtvis behöver kryssrutan också ett *id*.

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

Här är ytterligare ett layoutelement: För att signalera att de två elementen nedan hör ihop, ritar vi en ram **<frame>** (ruta). Ramen kan ha en rubrik *label*. Eftersom ramen bara är ett passivt

layoutelement, behöver den inte ett *id*. Vi definierar ändå ett här, eftersom vi refererar till det senare, när vi definierar ett ytterligare guidegränssnitt.

```
<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>
```

Inne i ramen **<frame>** placerar vi en annan kryssruta **<checkbox>** (och genom att använda *checked="true"*, signalerar vi att kryssrutan normalt ska vara markerad), och en nummerruta med **<spinbox>**. Nummerrutan gör det möjligt för användaren att välja ett värde mellan *"min"* och *"max"* med förvalda startvärdet *"0.95"*. Att ange typen *type* som *"real"* signalerar att reella tal accepteras i motsats till typen *type="integer"* som bara skulle acceptera heltal.

> NOT
> Det är också möjligt, och ofta att föredra, att göra själva ramen **<frame>** valbar, istället för att lägga till en kryssruta med **<checkbox>** inne i den. Se referensen för detaljerad information. Det görs inte här i illustrationssyfte.

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

Det är allt för den andra sidan i flikboken **<tabbook>** och alla element i dialogrutan **<dialog>**. Vi är klara med att definiera hur dialogrutan ser ut.

```
</document>
```

Till sist avslutar vi taggen **<document>**, och det är allt. Det grafiska användargränssnittet är definiera. Nu går det att spara filen. Men hur skapas R-syntax från inställningarna i det grafiska användargränssnittet? Det tar vi itu med senare i nästa kapitel. Först undersöker vi dock hur ett guidegränssnitt kan läggas till, och några allmänna hänsynstaganden.

4.2 Lägga till ett guidegränssnitt

I själva verket behöver vi inte definiera något ytterligare guidegränssnitt med **<wizard>**, men så här är hur man skulle göra det. Man lägger till taggen **<wizard>** på samma nivå som taggen **<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>
```

En del av det här är rätt självförklarligt: Vi lägger till taggen **<wizard>** med en *label* för guiden. Eftersom en guide kan innehålla flera sidor som visas en i taget, definierar vi därefter den första sidan, **<page>**, och lägger till en förklarande anmärkning där med **<text>**. Därefter använder vi taggen **<copy>**. Vad den gör är att vi slipper att återigen definiera vad vi redan gjorde för dialogrutan **<dialog>**: Taggen letar efter en annan tagg med samma *id* tidigare i XML-koden. Den råkar vara definierad i sektionen **<dialog>**, och är raden **<row>** där det finns en **<varselector>**, **<varslots>** och 'hypotesen' med alternativknapparna **<radio>**. Allt det kopieras ett-till-ett och infogas direkt vid elementet **<copy>**.

Nu till den andra sidan:

```
<page id="secondpage">
                                <text>Below are some advanced options. It's ↩
                                    generally safe not to assume the
                                        variables have equal variances. An  ↩
                                            appropriate correction will be  ↩
                                            applied then.
                                        Choosing "assume equal variances"  ↩
                                            may increase test-strength,  ↩
                                            however.</text>
                                <copy id="varequal"/>
                                <text>Sometimes it's helpful to get an  ↩
                                    estimate of the confidence interval of
                                        the difference in means. Below you  ↩
                                            can specify whether one should  ↩
                                            be shown, and
                                        which confidence-level should be  ↩
                                            applied (95% corresponds to a 5% ↩
                                             level of
                                        significance).</text>
                                <copy id="frame_conf_int"/>
                        </page>
                </wizard>
```

I stort sett samma sak här. Vi lägger till en del texter, och däremellan kopierar ytterligare sektioner från dialoggränssnittet med **<copy>**.

Du kan förstås låta guidegränssnittet se mycket annorlunda ut än den enkla dialogrutan, och inte använda taggen **<copy>** alls. Försäkra dig dock om att motsvarande element tilldelas samma *id* i båda gränssnitten. Det används inte bara för att överföra inställningarna från dialoggränssnittet till guidegränssnittet och tillbaka, när användaren byter gränssnitt (vilket inte sker ännu i den nuvarande versionen av RKWard), men förenklar också att skriva kodmallen (se nedan).

4.3 Några hänsynstaganden vid konstruktion av det grafiska användargränssnittet

Det här avsnittet innehåller några allmänna hänsynstaganden om vilka element i det grafiska användargränssnittet som ska användas var. Om det här är ditt första försök att skapa ett instickspogram, hoppa gärna över avsnittet, eftersom det inte är relevant för att få ett grundläggande grafiska användargränssnitt att fungera. Kom tillbaka hit senare, för att se om du kan förfina instickspogrammets grafiska användargränssnitt på ett eller annat sätt.

4.3.1 <radio> mot <checkbox> mot <dropdown>

De tre elementen **<radio>**, **<checkbox>**, **<dropdown>** har alla liknande funktion, att välja ett av flera olika alternativ. Naturligtvis tillåter en kryssruta bara att välja mellan två alternativ:

markerad eller inte markerad, så du kan inte använda den om det finns fler än två alternativ att välja mellan. Men när ska vilket av elementen användas? Några tumregler:

Om du märker att du skapar en alternativknapp, **<radio>** eller kombinationsruta, **<dropdown>** med bara två alternativ, fråga dig då om valet i stort sett är en fråga med svaret ja eller nej. Är det t.ex. ett val mellan att 'justera resultat' och 'justera inte resultat', eller mellan 'ta bort saknade värden' och 'behåll saknade värden'. I detta fall är en kryssruta, **<checkbox>**, det bästa valet: Den använder lite utrymme, har så få ord i beteckningen som möjligt, och är lättast att läsa för användaren. Det finns några få situationer där en alternativknapp, **<radio>**, bör användas istället för kryssrutan **<checkbox>**, när det bara finns två alternativ. Ett exempel kan vara: 'Beräkningsmetod: 'pearson'/'spearman''. Här kan det tänkas finnas fler metoder, och de är egentligen inte ett motsatspar.

Att välja mellan alternativknappen **<radio>** och kombinationsrutan **<dropdown>** är i huvudsak en fråga om utrymme. En **<dropdown>** har fördelen att ta upp lite utrymme även om det finns många alternativ att välja mellan. Å andra sidan, har en **<radio>** fördelen att alla möjliga alternativ är synliga för användaren på en gång, utan att klicka på kombinationsrutans pil. I allmänhet, om det finns sex eller fler alternativ att välja mellan är en **<dropdown>** att föredra. Om det finns fem eller färre alternativ är en **<radio>** det bättre valet.

Kapitel 5

Generera R-kod från inställningar i det grafiska användargränssnittet

5.1 Använda JavaScript i RKWard-instickspogram

Nu har vi definierat ett grafiskt användargränssnitt, men vi måste fortfarande skapa R-kod från det. För att göra det behöver vi en textfil till, `code.js`, placerad i samma katalog som `description.xml`. Du kanske eller kanske inte är bekant med JavaScript (eller, för att vara tekniskt precis: ECMA-script). Dokumentation om JS finns i överflöd, både på tryckt form och på Internet (t.ex. https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide). Men för de flesta syften behöver du inte kunna så mycket om JS alls, eftersom vi bara använder några mycket grundläggande funktioner.

> TIPS
> Ta också en titt på paketet rkwarddev efter att ha läst det här kapitlet. Det tillhandahåller några R-funktioner för att skapa JavaScript-kod som ofta används i RKWard. Det kan också automatiskt detektera variabler som används i ett instickspogrograms XML-fil och skapa grundläggande JavaScript-kod från det som en startpunkt för dig.

> NOT
> Instickspogrammets `.js`-filer antas vara kodade med UTF-8. Var noga med att kontrollera editorns kodning, om du använder några tecken som inte är ASCII.

För tvåvariablers t-testen, ser filen `code.js` ut som följer (med kommentarer emellan):

5.1.1 preprocess()

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

JS-filen är organiserad i tre separata funktioner: `preprocess()`, `calculate()` och `printout()`. Det beror på att all kod inte behövs i alla steg. För närvarande används inte funktionen preprocess på många ställen (typiskt utelämnas den helt och hållet).

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');
}
```

Funktionen skapar själva R-syntaxen som körs från inställningarna i det grafiska användargränssnittet. Låt oss ta en titt på den i detalj: Koden att använda skapas med satsen `echo()`. Vid en närmare titt på satsen `echo()` steg för steg, är den första delen

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

som enkel text. Därefter måste vi fylla i värdet som användaren valde som första variabel. Vi hämtar det genom att använda `getString(~x~)` och lägger till det i strängen som ska 'ekas'. Det skriver ut värdet på elementet i det grafiska användargränssnittet med *`id=~x~`*: vår första kryssruta, **<checkbox>**. Därefter lägger vi till ',', och gör samma sak för att hämta värdet av elementet *`~y~`*, den andra kryssrutan **<checkbox>**. För hypotesen (gruppen **<radio>**) och kryssrutorna med varianserna, **<checkbox>**, är proceduren nästan likadan.

Observera att istället för att sammanfoga de utskrivna delarna med '+', kan också flera olika `echo()` satser användas. Allt skrivs ut på en enda rad. För att skapa en ny rad i den skapade koden, infoga *`~\n~`* i strängen som ekas. Teoretiskt sett kan du till och med skapa många rader med en enda ekosats, men håll den helst till en rad (eller mindre) skapad god per `echo()`.

> NOT
> Förutom `getString()` finns också funktionerna `getBoolean()`, som försöker returnera ett logiskt värde (lämpligt för användning i en villkorssats, `if()`), och `getList()`, som försöker returnera data som liknar listan i en JS `Array()`. Vi visar exempel på dem senare.
> Om du tittar på befintliga instickspogram finner du också många som använder `getValue()` istället för `getString()`, och i själva verket är de två *nästan* identiska. Dock är rekommenderad användning sedan version 0.6.1 att använda `getString()`, `getBoolean()` och `getList()`.

Det blir lite krångligare för konfidensnivån. Av estetiska skäl vill vi inte explicit ange konfidensnivån som ska användas, om den motsvarar förvalt värde. Sålunda lagrar vi den först till en variabel istället för att ovillkorligt skriva ut värdet. Därefter kontrollerar vi om variabeln skiljer sig från *`~0.95~`* och i så fall skriver vi ut ett ytterligare argument. Till sist ekar vi en avslutande parentes och en ny rad: *`~)\n~`*. Det är allt i funktionen calculate.

5.1.3 printout()

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

Och det är allt i funktionen printout i de flesta fall. `rk.header()` skriver ut en standardrubrik för resultaten. Observera att alla översättningsbara strängar måste markeras för hand i `.js`-filen, genom att använda **i18n()** eller några alternativa kommandon. Mer om detta i kapitlet om internationalisering. Du kan också lägga till en del ytterligare information här, om du vill, exempelvis:

```
function printout () {
        new Header (i18n ("Two Variable t-Test"))
                  .addFromUI ("varequal")
                  .add (i18n ("Confidence level"), getString ("conflevel")) ↩
                        // Observera: skrivet såhär i illustrationssyfte.  ↩
                     Mer automatiskt:
        //        .addFromUI ("conflevel")
                  .print ();
echo ('rk.print (res)\n');
}
```

`rk.print()` utnyttjar paketet R2HTML för att tillhandahålla utmatning formaterad med HTML. En annan hjälpsam funktion är `rk.results()`, som också kan skriva ut olika sorters resultattabeller. Om du dock tvekar, använd bara `rk.print()`, så är du klar. JS-klassen `Header` är en hjälpklass på JS-nivå för att skapa ett anrop till `rk.header()` (ta bara en titt på den genererade R-koden). I vissa fall kan du behöva anropa `echo ('rk.header (...)')` direkt för att skriva ut en rubrik för utmatningen.

Observera att internt är utmatningen för närvarande bara ett enkelt HTML-dokument. Därför kan du bli frestad att lägga till egen HTML genom att använda `rk.cat.output()`. Även om det fungerar, gör helst inte det. Utdataformatet kan ändras i framtiden (t.ex. till ODF), så det är bäst att inte introducera HTML-specifik kod. Behåll hellre allt enkelt med `rk.header()`, `rk.print()`, `rk.results()`, och vid behov `rk.print.literal()`. Om det inte verkar uppfylla dina formateringsbehov, kontakta oss på e-postlistan för att få hjälp.

Gratulerar! Du har skapat ditt första instickspogram. Läs vidare i nästa kapitel om mer avancerade begrepp.

5.2 Konventioner, principer och bakgrund

Det finns många sätt att skriva R-kod för en viss uppgift, och det finns ännu fler sätt att generera R-koden från JS. Exakt hur du gör det, bestämmer du själv. Det finns ändå ett antal hänsynstaganden som du bör följa, och bakgrundsinformation som du bör förstå.

5.2.1 Förstå omgivningen `local()`

Vanligtvis måste man skapa en eller flera tillfälliga R-objekt i koden som genereras av instickspogrammet. Normalt vill man inte att de ska placeras i användarens arbetsrymd, och potentiellt till och med skriva över användarvariabler. Därför görs all kod genererad av instickspogram i en lokal omgivning, `local()` (se hjälpsidan i R om funktionen `local()`). Det betyder att alla variabler som skapas är tillfälliga och sparas inte permanent.

Om användaren explicit ber om att en variabel ska sparas, måste tilldelningar till objektet göras genom att använda `.GlobalEnv$objectname <- value`. I allmänhet ska inte operatorn `<<-` användas. Den tilldelar inte nödvändigtvis i .GlobalEnv.

En viktig fallgrop är användning av `eval()`. Här måste man observera att eval normalt använder den aktuella omgivningen för utvärdering, dvs. den lokala. Det fungerar oftast bra, men inte alltid. Om du sålunda behöver använda `eval()`, bör du ange parametern *`envir`*: `eval(..., envir=globalenv())`.

5.2.2 Kodformatering

Det viktigaste är att den genererade R-koden fungerar, men den bör också vara lättläst. Håll därför också ett öga på formateringen. Några överväganden:

Normala toppnivå R-satser ska vara vänsterjusterade.

Satser i ett lägre block ska indenteras med en flik (se exemplet nedan).

Om du utför mycket komplexa beräkningar, lägg till en kommentar här och där, i synnerhet för att markera logiska avsnitt. Observera att det finns en särskild funktion, **comment()**, för att infoga översättningsbara kommentarer i genererad kod.

Generad kod kan exempelvis se ut så här. Samma kod utan indentering eller kommentarer skulle vara ganska svårläst, trots dess blygsamma svårighetsgrad:

```
# bestäm först svaj och rotation
my.wobble <- wobble (x, y)
my.rotation <- wobble.rotation (my.wobble, z)

# vacklingsmetod måste väljas enligt 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 Hantera komplexa alternativ

Många insticksprogram kan göra mer än en sak. Exempelvis kan insticksprogrammet 'Beskrivande statistik' beräkna medelvärde, intervall, summa, produkt, median, längd, etc. Dock väljer användaren typiskt att bara utföra vissa av beräkningarna. Försök hålla den genererade koden så enkel som möjligt i sådana fall. Den ska bara innehålla delarna som är relevanta för alternativen som faktiskt har valts. För att åstadkomma det är här ett exempel på ett vanligt konstruktionsmönster som det skulle användas (i JS skulle elementen ″domean″, ″domedian″ och ″dosd″ här vara kryssrutor, <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 Tips och trick

Här är några blandade trick som kan göra det mindre arbetsamt att skriva insticksprogram:

Om värdet av en inställning i det grafiska användargränssnittet behövs på flera platser i insticksprogrammets kod, överväg att tilldela det till en variabel i JS, och använda den istället för att hämta det upprepade gånger med `getString()/getBoolean()/getList()`. Det är snabbare, mer läsbart, och mindre att skriva på en gång:

```
function calculate () {
        var narm = "";        // na.rm=FALSE är förvalt värde i alla ↩
            funktioner nedan
        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");
        // ...
}
```

Den enkla hjälpfunktionen `makeOption()` kan göra det enklare att utelämna parametrar som har sina förvalda värden, i många fall:

```
function calculate () {
        var options
        //...
        // Den här gör ingenting, om VALUE är 0,95 (förvalt värde). Annars ↩
            lägger den till ', conf.int=VALUE' i alternativen.
        options += makeOption ("conf.int", getString ("confint"), "0.95");
        //...
}
```

Kapitel 6

Skriva en hjälpsida

När insticksprogrammet i grund och botten fungerar, är det dags att tillhandahålla en hjälpsida. Även om man typiskt inte vill förklara alla bakomliggande koncepten från grunden, kanske du vilja lägga till några fler förklaringar för vissa av alternativen, och länka till relaterade insticksprogram och R-funktioner.

> **TIPS**
> Efter att ha läst det här kapitlet, ta också en titt på paketet rkwarddev. Det tillhandahåller några R-funktioner för att skapa de flesta av RKWards XML-taggar åt dig. Det klarar också av att skapa grundmallar för hjälpfiler från insticksprogrammets befintliga XML-filer att utgå ifrån.

Du kanske minns att du lagt till det här i insticksprogrammets XML (om du inte har det, lägg till det nu):

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

Där du naturligtvis ersätter `filnamn` med ett lämpligare namn. Nu är det dags att skapa `.rkh`-filen. Här är ett självbeskrivande exempel:

```
<!DOCTYPE rkhelp>
<document>
        <summary>
I det här avsnittet skriver man in kort, mycket grundläggande information ↩
    om vad insticksprogrammet gör.
Avsnittet visas alltid allra längst upp på hjälpsidan.
        </summary>

        <usage>
Sektionen usage kan innehålla lite mer praktisk information. Den ska dock ↩
    inte förklara
alla inställningarna i detalj (det görs i sektionen "settings").

Infoga en ny rad för att påbörja ett nytt stycke, som visas ovan.
I motsats till det, ingår den här raden i samma stycke.

Enkel HTML-kod kan infogas i alla avsnitt, såsom text med <b>fetstil</b> ↩
    eller
```

```
<i>kursiv stil</i>. Använd dock helst minimal formatering som är nödvändig.

Avsnittet usage är alltid det andra avsnittet som visas på en hjälpsida.
        </usage>

        <section id="sectionid" title="Generic section" short_title=" ↩
            Generic">
Om det behövs, kan ytterligare avsnitt läggas till mellan avsnitten usage  ↩
    och settings.
Dock behövs det oftast inte när insticksprogram dokumenteras. Egenskapen " ↩
    id"
tillhandahåller en ankringspunkt för att gå till avsnittet i  ↩
    navigeringsmenyn. Egenskapen "short_title"
tillhandahåller en kort rubrik att använda i navigeringsraden. Den är  ↩
    valfri, normalt
använd "title" både som rubrik för avsnittet, och som länknamnet i
navigeringsraden.

Man kan vilja infoga länkar till ytterligare information i vilket avsnitt  ↩
    som helst. Det gör man genom att lägga till

<link href="webbadress">länknamn</link>

Där webbadress skulle kunna vara en extern adress som http://rkward.kde.org ↩
    .
Flera speciella webbadresser stöds i hjälpsidorna:

<link href="rkward://sida/sökväg/sid-id"/>

Länkar till en rkward hjälpsida på toppnivå (inte för ett insticksprogram).

<link href="rkward://component/[namnrymd/]komponent-id"/>

Länkar till hjälpsidan för ett annat insticksprogram. Delen [namnrymd/] kan ↩
    utelämnas
(i så fall, antas rkward som standardnamnrymd, exempelvis är
<link href="rkward://component/import_spss"/> och
<link href="rkward://component/rkward/import_spss"/> ekvivalenta).
Komponent-id är samma som anges av .pluginmap.

<link href="rkward://rhelp/r-funktion"/>

Länkar till R-hjälpsidan om "r-funktion".

Observera att länknamn skapas automatiskt för följande typer av länkar.
        </section>

        <settings>
                <caption id="id_of_tab_or_frame"/>
                <setting id="id_of_element">
Beskrivning av elementet i det grafiska användargränssnittet identifierat  ↩
    av angivet id
                </setting>
                <setting id="id_of_elementb" title="description">
Oftast extraherar rubriken för elementet i det grafiska  ↩
    användargränssnittet automatiskt från
insticksprogrammets XML-definition. Dock
kanske beskrivningen inte är nog för att tillförlitligt identifiera dem för ↩
```

```
    vissa element i det grafiska användargränssnittet.
I så fall kan en explicit rubrik läggas till med egenskapen "title".
                </setting>
                <setting id="id_of_elementc">
Beskrivning av elementet i det grafiska användargränssnittet identifierat ↩
    av "id_of_elementc"
                </setting>
                [...]
        </settings>

        <related>
Avsnittet related innehåller oftast bara några länkar, såsom:

<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>
Avsnittet technical (valfritt, alltid sist) kan innehålla några tekniska ↩
    detaljer om insticksprogrammets
implementering, som bara är av intresse för utvecklare av RKWard. Det är ↩
    särskilt relevant
för insticksprogram som är konstruerade för att inbäddas i många andra ↩
    insticksprogram, och kan ange vilka
alternativ är tillgängliga för att anpassa det inbäddade insticksprogrammet ↩
    , och vilka kodsektioner som innehåller vilken
R-kod.
        </technical>
</document>
```

Kapitel 7

Logisk interaktion mellan element i det grafiska användargränssnittet

7.1 Logik för grafiskt användargränssnitt

Alla grundkoncepten för att skapa ett instickspogram för RKWard har beskrivits i de tidigare kapitlen. Grundkoncepten bör vara tillräckliga för många fall, kanske de allra flesta. Dock vill man ibland ha mer kontroll över hur instickspogrammets grafiska användargränssnitt beter sig.

Antag exempelvis att du vill utöka t-test exemplet som används i den här dokumentationen för att både tillåta att en variabel jämförs med en annan variabel (som visas här), och jämföra en variabel mot ett konstantvärde. Ja, ett sätt att göra det skulle vara att lägga till en alternativknapp som byter mellan de två lägena och lägga till en nummerruta för att skriva in konstantvärdet att jämföra med. Betrakta det här förenklade exemplet:

```
<!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>
```

Så långt är allt gott och väl, men det finns ett antal problem med det här grafiska användargränssnitt. För det första visas både varslot elementet och nummerrutan, medan bara en av de två verkligen används. Vad värre är, varslot elementet kräver alltid ett giltigt val, även vid jämförelse med en konstant. Om vi skapar ett grafiskt användargränssnitt med flera syften, vill vi uppenbarligen ha mer flexibilitet. Lägg då till sektionen **<logic>** (infogad på samma nivå som **<code>**, **<dialog>** eller **<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">
        [...]
```

Den första raden inne i sektionen logic är taggen **<convert>**. Den tillhandahåller egentligen en ny Boolesk egenskap (på eller av, sann eller falsk), som kan användas senare. Egenskapen (*˜varmode˜*) är sann så snart den övre alternativknappen är vald, och falsk så snart den nedre alternativknappen är vald. Hur görs det?

För det först listas källegenskaperna att arbeta med under *sources* (i detta fall bara en vardera, flera skulle kunna listas som *sources="mode.string;någonting-annat"*, då skulle *˜varmode˜* bara vara sann om både *˜mode.string˜* och *˜någonting-annat˜* är lika med strängen *˜variable˜*). Observera att i detta fall skriver vi inte bara *˜mode˜* (som vi skulle göra i `getString(˜mode˜)`), utan *˜mode.string˜*. Det är det faktiska interna sättet som en alternativknapp fungerar: Den har egenskapen 'string' som innehåller dess strängvärde. `getString(˜mode˜)` är bara en kortform, och ekvivalent med `getString(˜mode.string˜)`. Se referensen för alla egenskaper hos de olika elementen i det grafiska användargränssnittet.

För det andra, ställer vi in konverteringens läge till *mode="equals"*. Det betyder att vi vill kontrollera om källan eller källorna är lika med ett visst värde. Till sist är standard värdet att jämföra med, så med *standard="variable"* kontrollerar vi om egenskapen *˜mode.string˜* är lika med strängen *˜variable˜* (värdet av den övre alternativknappen). Om de är lika, är egenskapen varmode sann, annars är den falsk.

Nu till själva kärnan: Vi använder **<connect>** för att ansluta egenskapen *˜varmode˜* till y.visible, vilket bestämmer om varslot *˜y˜* visas eller inte. Observera att eventuella element som görs osynliga implicit inte krävs. Sålunda, om den övre alternativknappen väljes, krävs varslot *˜y˜* och är synlig. Annars krävs den inte och är dold.

För nummerrutan vill vi ha exakt det motsatta. Som tur är behöver vi inte ett annat **<convert>** för den: Booleska egenskaper kan enkelt negeras genom att lägga till modifieraren *˜not˜*, så vi använder **<connect>** *˜varmode.not˜* för nummerrutans egenskap visibility. På så sätt krävs antingen elementet varslot och visas, *eller* krävs nummerrutan och visas, beroende på vilket alternativ som är valt av alternativknapparna. Det grafiska användargränssnittet ändras enligt alternativknapparna. Prova exemplet, om du har lust.

Se referenskapitlet för en fullständig lista över egenskaper. Ytterligare en egenskap är dock speciell på det sättet att alla element i det grafiska användargränssnittet har den: 'enabled'. Den är något mindre drastisk än 'visible'. Den visar eller döljer inte elementet i det grafiska användargränssnittet, utan bara aktiverar eller inaktiverar det. Inaktiverade element visas typiskt med grått, och reagerar inte på användarinmatning.

NOT
Förutom **<convert>** och **<connect>**, finns det flera ytterligare element att använda i sektionen **<logic>**. Villkorliga konstruktioner kan exempelvis också implementeras genom att använda elementet **<switch>**. Se referenskapitlet om logiska element för detaljerad information.

7.2 Skriptbaserad logik för det grafiskt användargränssnittet

Medan anslutning av egenskaper som beskrivs ovan ofta är nog, är det flexiblare eller bekvämare att använda JS för att hantera det grafiska användargränssnittets logik med ett skript. På detta sätt skulle exemplet ovan kunna skrivas om som:

```
[...]
        <code file="code.js"/>
'
        <logic>
                <script><![CDATA[
                        // ECMA-skript kod i blocket
                        // toppnivåsatsen anropas bara en gång
                        gui.addChangeCommand ("mode.string", "modeChanged ↩
                            ()");

                        // funktionen anropas så fort "mode" ändras
                        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">
        [...]
```

Den första kodraden talar om för RKWard att funktionen `modeChanged()` ska anropas så fort värdet på alternativknappen *`id="mode"`* ändras. Inne i funktionen definierar vi hjälpvariabeln *`"varmode"`* som är sann när läget är *`"variable"`* och falsk när det är *`"constant"`* . Sedan använder vi `gui.setValue()` för att ställa in egenskaperna 'enabled' för *`"y"`* och *`"constant"`*, precis på samma sätt som vi tidgare gjorde med satsen **<connect>**.

Skriptmetoden för logik i det grafiska användargränssnittet är särskilt användbart när de tillgängliga alternativen behöver ändras enligt typ av objekt som användaren väljer. Se referenskapitlet för tillgängliga funktioner.

Observera att skriptmetoden för logik i det grafiska användargränssnittet kan blandas med satserna **<connect>** och **<convert>** om du vill. Observera också att taggen **<script>** tillåter ett skriptfilnamn som tillägg till eller alternativ till att inbädda skriptkoden. Oftast är det dock bekvämast att inbädda den som visas ovan.

Kapitel 8

Inbädda insticksprogram i insticksprogram

8.1 Användarfall för inbäddning

När du skriver insticksprogram, märker du ofta att du skapar ett antal insticksprogram som bara skiljer sig i några avseenden, men har mycket mer gemensamt. Exempelvis för att rita diagram finns ett antal generella R-alternativ som kan användas med nästan alla sorters diagram. Ska man skapa ett grafiskt användargränssnitt och JS-mall för dem gång på gång?

Uppenbarligen skulle det vara rätt besvärligt. Som tur är behöver man inte göra det. Istället skapar man kärnan med den gemensamma funktionen en gång, och kan senare inbädda den i flera olika insticksprogram. I själva verket är det möjligt att inbädda vilket insticksprogram som helst i vilket annat insticksprogram som helst, även om de ursprungliga upphovsmännen till de inbäddade insticksprogrammen aldrig trodde att någon skulle vilja inbädda deras insticksprogram i något annat.

8.2 Inbäddning inne i en dialogruta

OK, nog sagt. Hur fungerar det? Enkelt: Använd bara taggen **<embed>**. Här är ett avkortat exempel:

```
<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>
```

Vad som händer här är att hela det grafiska användargränssnittet för instickspogrammet med diagramalternativ (utom förstås standardelementen som knappen **Verkställ**, etc.) inbäddas direkt i ditt instickspogram (prova det!).

Som du märker är syntaxen för taggen **<embed>** rätt enkel. Den har en *id* som de flesta element. Parameterkomponenten anger vilket instickspogram som ska inbäddas, som definierad i `.plug inmap`-filen (*˜rkward::plot_options˜* är resultatet av att sammanfoga namnrymden 'rkward', en avskiljare '::' och komponentens namn 'plot_options').

8.3 Kodgenerering vid inbäddning

Så långt är allt gott och väl, men vad händer med den genererade koden? Hur sammanfogas koden för det inbäddande och det inbäddade instickspogrammen? Skriv helt enkelt någonting som liknar det här i det inbäddande instickspogrammet:

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

I grund och botten hämtar vi alltså koden som skapas av det inbäddade instickspogrammet precis som vi hämtar alla andra inställningar av det grafiska användargränssnittet. Här kan strängen *˜plotoptions.code.printout˜* tolkas del för del som: 'Utskriftssektionen av den skapade koden för elementet med *id* plotoptions' (plotoptions är den id som vi angav för taggen **<embed>** ovan). Och jovisst, om du behöver avancerad kontroll, kan du till och med hämta värden av enskilda element i det grafiska användargränssnittet för det inbäddade instickspogrammet (men inte tvärtom, eftersom det inbäddade instickspogrammet inte vet någonting om sin omgivning).

8.4 Inbäddning inne i en guide

Om instickspogrammet tillhandahåller ett grafiskt användargränssnitt med en guide, fungerar inbäddning i stort sett på samma sätt. I allmänhet använder man:

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

Om det inbäddade grafiska användargränssnittet tillhandahåller ett guidegränssnitt, infogas dess sidor direkt mellan *page12* och *page13* i ditt instickspogram. Om det inbäddade grafiska användargränssnittet bara tillhandahåller ett dialoggränssnitt, infogas en enda ny sida mellan dina sidor *page12* och *page13*. Användaren märker aldrig något.

8.5 Mindre inbäddad inbäddning: Knappen Ytterligare alternativ

Med inbäddning är häftigt, bör man vara försiktig så att man inte överdriver. För många funktioner i ett grafiskt användargränssnitt gör det bara svårt att hitta de relevanta alternativen. Man kan naturligtvis ibland vilja inbädda ett stort antal alternativ (som alla alternativ i `plot()`), men eftersom de är helt valfria, vill man inte att de ska synas på en framträdande plats i det grafiska användargränssnittet.

Ett alternativ är att inbädda alternativen 'som en knapp':

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

I detta fall har en enda tryckknapp lagts till i instickspogrammet, med beteckningen **Specify plotting options**. När knappen klickas, dyker en separat dialogruta upp med alla det inbäddade instickspogrammets alternativ. Även om det inbäddade grafiska användargränssnittet inte är synligt för det mesta, kan inställningarna hämtas precis som beskrevs ovan.

> OBSERVERA
> Troligen bör metoden med en 'knapp' enbart användas för instickspogram som aldrig kan vara ogiltiga (för saknade eller felaktiga inställningar). Annars skulle inte användaren kunna verkställa koden, men kan ha svårt att få reda på det, eftersom orsaken är dold bakom någon knapp.

8.6 Inbädda eller definiera ofullständiga instickspogram

Vissa instickspogram är inte fullständiga i sig själva, och i själva verket är plot_options använt som exempel ovan ett av dem. De har helt enkelt inte elementen i det grafiska användargränssnittet för att välja vissa viktiga värden. De är bara avsedda att användas inbäddade i andra instickspogram.

Hur långt är instickspogrammet plot_options ofullständigt? Jo, för vissa inställningsalternativ behöver det veta namnet på objekten eller uttrycken för x- och y-axlarna (i själva verket fungerar det bra om det har endera, men det behöver minst ett för att fungera som det ska). Dock har det inte någon mekanism för att välja objekten, eller mata in dem på något annat sätt. Så hur känner det då till dem?

I sektionen logic i instickspogrammet plot_options finns ytterligare två rader, som inte behandlats ännu:

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

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

Det definierar ytterligare två egenskaper i instickspogrammet plot_options, vars enda syfte är att anslutas till några (ännu okända) egenskaper i det inbäddande instickspogram. I instickspogrammet plot_options är de två egenskaperna använda precis som vilka andra som helst, och det finns exempelvis anrop till `getString(~xvar~)` i JS-mallen i plot_options.

För det ofullständiga instickspogrammet finns det inget sätt att veta var det inbäddas, och vad de relevanta inställningarna i det inbäddande instickspogram heter. Vi måste alltså dessutom lägga till två ytterligare rader i det inbäddande instickspogrammets sektion logic:

```
<logic>
                [...]

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

Det är principiellt ingenting nytt, vi har behandlat satsen **<connect>** i kapitlet om logik i det grafiska användargränssnittet. Man ansluter helt enkelt värdena i två varslots (benämnda *~xvarslot~* och *~yvarslot~* i exemplet) till de mottagande 'externa' egenskaperna i det inbäddade instickspogrammet. Det är allt. Allting annat hanteras automatiskt.

Kapitel 9

Hantera många liknande insticksprogram

9.1 Översikt av olika tillvägagångssätt

Ibland kan man vilja utveckla insticksprogram för en serie liknande funktioner. Som ett exempel, fundera på fördelningsdiagram. De genererar ganska lika kod, och det är naturligtvis önskvärt att få det grafiska gränssnitten att likna varandra. Till sist kan stora delar av hjälpfilerna vara identiska. Bara några få parametrar är olika i varje insticksprogram.

Det naiva tillvägagångssättet är att utveckla ett insticksprogram, därefter i stort sett kopiera och klistra in hela innehållet i `.js`-, `.xml`- och `.rkh`-filerna, och ändra de få delarna som är olika. Men vad händer om du någon gång senare hittar ett stavfel som har kopierats och klistrats in i alla insticksprogrammen? Vad händer om du vill lägga till stöd för en ny funktion? Du måste hitta alla insticksprogram igen, och ändra vart enda ett. En tröttsam och långdragen process.

Ett annat tillvägagångssätt skulle vara att använda inbäddning. Dock lämpar sig inte det väl för det aktuella problemet i vissa fall, eftersom bitarna som kan inbäddas ibland är för stora för att vara användbara, och det innebär vissa begränsningar av layouten. I sådan fall kan koncepten att inkludera `.js` filer, inkludera `.xml` filer och använda kodsnuttar vara mycket användbara (med se funderingarna om när det är att föredra att använda inbäddning).

Dock ett varningens ord innan du börjar läsa: Koncepten kan göra det enklare att hantera många liknande insticksprogram, och kan förbättra insticksprogrammens underhåll och läsbarhet. Att gå för långt kan dock enkelt ge motsatt effekt. Använd med viss försiktighet.

9.2 Använda JS include-sats

Det är enkelt att inkludera en skriptfil i en annan med RKWard insticksprogram. Värden av det är omedelbart uppenbart om vissa delar av JS-koden är likadan mellan insticksprogram. Det går helt enkelt att definiera sådana delar i en separat `.js`-fil och inkludera den i alla insticksprograms `.js`-filer. Exempelvis så här:

```
// det här är en fil som heter "common_functions.js"

function doCommonStuff () {
        // kanske hämta några alternativ, etc.
        // ...
        comment ("This is R code you want in several different plugins\n");
```

```
        // ...
}
```

```
// det här är en av de vanliga .js filerna i ett instickspogram

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

function calculate () {
        // gör någonting
        // ...

        // infoga den gemensamma koden
        doCommonStuff ();
}
```

Observera att det ibland är ännu mer användbart att vända på det, och definiera 'mallar' av funktionerna `preprocess()`, `calculate()` och `printout()` i en gemensam fil, och låta dem anropa tillbaka för delarna som är olika mellan instickspogram. Exempelvis:

```
// det här är en fil som heter "common_functions.js"

function calculate () {
        // gör någonting som är samma i alla instickspogram
        // ...

        // lägg till något som är olika mellan instickspogram
        getSpecifics ();

        // ...
}
```

```
// det här är en av de vanliga .js filerna i ett instickspogram

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

// observera: funktionen calculate() definieras inte här.
// den finns istället i common_functions.js.

function getSpecifics () {
        // skriv ut någon R-kod
}
```

Ett problem som man måste vara medveten om när tekniken används är variabelräckvidd. Se JS-manualen om variabelräckvidd (variable scope).

Tekniken används flitigt i instickspogrammen för fördelningsdiagram (distribution plot) och CLT-fördelningsdiagram (distribution CLT plot), så det kan vara värt att titta där för exempel.

9.3 Inkludera `.xml`-filer

I stort sett samma funktion för att inkludera filer är också tillgänglig för användning i `.xml`, `.pluginmap` och `.rkh`-filer. På vilket ställe som helst i filerna kan taggen **<include>** placeras, som visas nedan. Effekten är att hela innehållet i den XML-filen (för att vara exakt: allting inom taggen **<document>** i den filen) inkluderas ordagrant på det stället i filen. Observera att det bara går att inkludera en annan XML-fil.

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

Egenskapen *file* är filnamnet relativt katalogen där den aktuella filen finns.

9.4 Använda <snippets>

Medan det är ganska kraftfullt att inkludera filer som visas i föregående avsnitt, blir det som mest användbart när det kombineras med **<snippets>**. De är egentligen mindre delar som kan infogas på ett annat ställe i filen. Det åskådliggörs bäst av ett exempel:

```
<document>
        <snippets>
                <snippet id="note">
                        <frame>
                                <text>
                Det här infogas på två ställen i det grafiska  ↩
                    användargränssnittet
                                </text>
                        </frame>
                </snippet>
        </snippets>
        <dialog label="test">
                <column>
                        <insert snippet="note"/>
                        [...]
                        <insert snippet="note"/>
                </column>
        </dialog>
</document>
```

Sålunda definieras delen på ett ställe längst upp i XML-filen, och sedan infogas den på vilket eller vilka ställen som man vill med **<insert>**.

Medan exemplet inte är alltför användbart i sig, tänk på att kombinera det med en `.xml`-fil inkluderad med **<include>**. Observera att det också går att placera delar för `.rkh`-filen i samma fil. Man inkluderar helt enkelt filen där också med **<include>**, och infogar relevanta delar med **<insert>**:

```
<!-- Det här är en fil som heter "common_snippets.xml" -->
<document>
        <snippet id="common_options">
                <spinbox id="någonting" [...]/>
                [...]
        </snippet>
        <snippet id="common_note">
                <text>En viktig anmärkning för den här typen av  ↩
                    insticksprogram</text>
        </snippet>

        <snippet id="common_help">
                <setting id="something">Det här gör någonting</setting>
                [...]
```

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

```
<!-- Det här är insticksprogrammets .xml-fil -->
<document>
        <snippets>
                <!-- Importera common snippets -->
                <include file="common_snippets.xml"/>
        </snippets>

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

I likhet med att inkludera i JS, är det omvända tillvägagångssättet ofta ännu mer användbart:

```
<!-- Det här är en fil som heter "common_layout.xml" -->
<document>
        <column>
                <insert snippet="note">
                [...]
                <insert snippet="plugin_parameters">
        </column>
        [...]
</document>
```

```
<!-- Det här är insticksprogrammets .xml-fil -->
<document>
        <snippets>
                <snippet id="note">
                        <text>Anmärkningen använd för det här specifika  ↩
                            insticksprogrammet</text>
                </snippet>

                <snippet id="plugin_parameters">
                        <frame label="Parametrar specifika för det här  ↩
                            insticksprogrammet">
                                [...]
                        </frame>
                </snippet>
        </snippets>

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

Till sist, är det också möjligt att infoga delar med **<insert>** i andra delar, under förutsättning att det för det första bara finns en nivå av inkapsling, och för det andra att sektionen med **<snippets>** placeras längst upp i filen (innan en inkapslad del infogas), beroende på att satser med **<insert>** hanteras uppifrån och ner.

9.5 <include> och <snippets> mot <embed>

Vid första ögonkastet tillhandahåller **<include>** och **<snippets>** funktionalitet som är ganska lik inbäddning: De möjliggör återanvändning av vissa delar av koden mellan insticksprogram. Vad är då skillnaden mellan tillvägagångssätten, och när ska man använda vilket?

Den avgörande skillnaden mellan koncepten är att inbäddningsbara insticksprogram utgör ett tätare kopplat paket. De kombinerar ett fullständigt grafiskt användargränssnitt, kod för att generera R-kod från det och en hjälpsida. I motsats till det, tillåter include och insert mycket finkornigare kontroll, men till priset av mindre modularitet.

Det vill säga, ett insticksprogram som inbäddar ett annat insticksprogram behöver typiskt inte veta mycket om det inbäddade insticksprogrammets interna detaljer. Ett utmärkt exempel är insticksprogrammet plot_options. Insticksprogram som vill inbädda det behöver inte nödvändigtvis känna till alla alternativ som tillhandahålls, och hur de tillhandahålls. Det är bra, eftersom annars skulle en ändring av insticksprogrammet plot_options göra det nödvändigt att justera alla insticksprogram som inbäddar det (många). I kontrast till det, exponerar include och insert alla de interna detaljerna, och insticksprogram som använder dem, måste exempelvis känna till exakt id och kanske till och med typ för de använda elementen.

Sålunda är tumregeln följande: include och insert är utmärkta om de relevanta alternativen bara behövs för en tydligt begränsat grupp av insticksprogram. Inbäddade insticksprogram är bättre om gruppen av insticksprogram som det kan vara användbart för inte är klart definierad, och om funktionaliteten enkelt kan modulariseras. En annan tumregel: om det går att placera de gemensamma delarna i ett enda 'block', gör det och använd inbäddning. Om det behövs många små delar för att definiera allt gemensamma, använd då **<snippets>**. Ett sista sätt att se på det: Om alla insticksprogram tillhandahåller *mycket* liknande funktionalitet, är include och insert troligen en god idé. Om de bara delar en eller två gemensamma 'moduler' är inbäddning sannolikt bättre.

Kapitel 10

Koncept för användning i specialiserade instickspogram

Kapitlet innehåller information om några ämnen som bara är användbara för vissa klasser av instickspogram.

10.1 Instickspogram som skapar ett diagram

Att skapa ett diagram från ett instickspogram är lätt att göra. Dock finns det några subtila misstag att undvika, och också en del utmärkt generell funktionalitet som man bör vara medveten om. Det här avsnittet visar grundkoncepten, och avslutas med ett standardexempel som bör följas så snart diagraminstickspogram skapas.

10.1.1 Rita ett diagram i utmatningsfönstret

Använd `rk.graph.on()` direkt innan diagrammet skapas, och `rk.graph.off()` direkt efteråt, för att rita ett diagram i utmatningsfönstret. Det liknar t.ex. anrop av `postscript()` och `dev.off()` i en vanlig R-session.

Det är dock viktigt att *alltid* anropa `rk.graph.off()` efter att ha anropat `rk.graph.on()`. Annars lämnas utdatafilen i ett felaktigt tillstånd. För att försäkra att `rk.graph.off()` verkligen anropas, måste *alla* R-kommandon mellan de två anropen omges med en `try()`-sats. Har du aldrig hört talas om det? Var inte orolig, det är enkelt. Allt du behöver göra är att följa mönstret som visas i exemplet nedan.

10.1.2 Lägga till funktionalitet för förhandsgranskning

> NOT
> Det här avsnittet beskriver hur förhandsgranskningsfunktionalitet läggs till i instickspogram som skapar diagram. Det finns separata avsnitt om förhandsgranskning av (HTML-)utmatning, förhandsgranskningar av (importerad) data och anpassade förhandsgranskningar. Dock rekommenderas du att läsa det här avsnittet först, eftersom tillvägagångssättet är liknande i alla tre fallen.

En mycket användbar funktion för alla instickspogram som skapar ett diagram eller graf är att tillhandahålla en förhandsgranskning med automatisk uppdatering. Det behövs två saker för att göra det: Tillägg av kryssrutan **<preview>** i definitionen av det grafiska användargränssnittet, och justering av den genererade koden för förhandsgranskningen.

Att lägga till kryssrutan **<preview>** är enkelt. Placera bara följande någonstans i det grafiska användargränssnittet. Det tar hand om all magi bakom kulisserna för att skapa en förhandsgranskningsenhet, uppdatera förhandsgranskningen så fort inställningarna har ändrats, etc. Exempel:

> NOT
> Från version 0.6.5 av RKWard hanteras förhandsgranskningselement med **<preview>** speciellt i instickspogrammens dialogrutor (inte guider). De placeras i knappkolumnen, oberoende av exakt var de definieras i användargränssnittet. Det är ändå en bra idé att definiera dem på ett vettigt ställe i layouten, för bakåtkompatibilitet.

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

Och det är allt för definitionen av det grafiska användargränssnittet.

Att justera JS-mallen är bara lite mer arbete. Här måste man säkerställa att bara själva diagrammet skapas, och visas på en skärmenhet, istället för att skickas som utdata, dvs. ingen utskrift av rubriker, `rk.graphics.on()`, eller liknande anrop. För att hjälpa till med det, anropar RKWard funktionerna `preprocess()`, `calculate()` och `printout()` med en ytterligare parameter som ställs in till *`true`* när kod genereras för en förhandsgranskning. (Parametern utelämnas när den slutliga koden skapas. I Javascript utvärderas det som *`false`* när det används inne i en `if`-sats.) Se exemplet nedan för det typiska mönstret man ska använda.

Som ett alternativ, om mer kontroll behövs än så här, kan man istället lägga till en ny funktion vid namn `preview()` i JS-mallen, och generera koden som krävs för en förhandsgranskning där (troligen, åtminstone delvis, igen genom att anropa `calculate()`, etc.).

10.1.3 Generella diagramalternativ

Du har märkt att de flesta diagraminstickspogram i RKWard tillhandahåller ett stort antal generella alternativ, t.ex. för att anpassa axlarnas rubriker eller figurmarginaler. Det är enkelt att lägga till alternativen i ett instickspogram. De tillhandahålls av ett inbäddningsbart instickspogram som heter **rkward::plot_options**. Inbädda det i instickspogrammets användargränssnitt så här:

```
<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>
```

Det lägger till en knapp i användargränssnittet för att visa ett fönster med diagramalternativ. Sektionen logic är bara ett exempel. Den ger en viss kontroll över instickspogrammet för diagramalternativ. Läs mer på instickspogrammet plot_options hjälpsida (länkad från hjälpsidan i alla instickspogram som tillhandahåller de generella alternativen).

Därefter måste du försäkra dig om att koden som motsvarar diagramalternativen läggs till i koden som genereras för ditt diagram. Hämta egenskaperna **code.preprocess**, **code.printout** och **code.calculate** från det inbäddade instickspogrammet för diagramalternativ för att göra det, och infoga dem i din kod som visas i exemplet nedan.

10.1.4 Ett standardexempel

Här är ett exempel på en .JS-fil som bör användas som mall, så fort ett diagraminstickspogram skapas:

```
function preprocess () {
    // "somepackage" behövs för att skapa diagrammet
    echo ("require (somepackage)\n");
  }

  function printout (is_preview) {
    // Om "is_preview" tilldelas false/odefinierad,skapas fullständig kod,  ↩
        inklusive rubriker.
    // Om "is_preview" tilldelas true, skapas bara det viktigaste.

    if (!is_preview) {
      echo ('rk.header (' + i18n ("Ett exempeldiagram") + ')\n\n');
      echo ('rk.graph.on ()\n');
    }
    // bara följande del skapas när is_preview==true

    // kom ihåg att allting mellan rk.graph.on() och rk.graph.off() ska  ↩
        omges med en try()-sats:
    echo ('try ({\n');
    // infoga eventuell kod för inställning av alternativ som ska köras  ↩
        innan själva uppritningskommandona.
    // Själva koden tillhandahålls av det inbäddade instickspogrammet för  ↩
        uppritningsalternativ. printIndentedUnlessEmpty() tar hand om snygg  ↩
        formatering.
    printIndentedUnlessEmpty ('\t', getString ("plotoptions.code.preprocess ↩
        "), '', '\n');

    // skapa själva diagrammet. plotoptions.code.printout tillhandahåller  ↩
        de generella uppritningsalternativen
    // som måste läggas till i själva uppritningsanropet.
    echo ('plot (5, 5' + getString ("plotoptions.code.printout") + ')\n');
```

```
    // infoga eventuell kod för inställning av alternativ som ska köras  ↩
        efter själva uppritningen.
    printIndentedUnlessEmpty ('\t', getString ("plotoptions.code.calculate ↩
        "), '\n');
    echo ('})'\n);  // avslutning av try()-satsen

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

10.2 Förhandsgranskningar av data, utmatning och andra resultat

10.2.1 Förhandsgranskning av (HTML-)utmatning

> NOT
> Det här avsnittet beskriver hur förhandsgranskningsfunktionalitet läggs till i instickspogram som skapar utmatning eller HTML-utskrifter. Vi rekommenderar att du läser det separata avsnittet om förhandsgranskning av diagram innan det här avsnittet.

Att skapa en förhandsgranskning av HTML-utmatning använder nästan samma procedur som att skapa en förhandsgranskning av ett diagram. I det här fallet måste man bara försäkra sig om att **preview()** genererar relevanta kommandon med **rk.print()/rk.results()**. Det är i allmänhet en god idé att utelämna rubriksatser i förhandsgranskningen. Här är ett avkortat exempel:

```
<!-- I instickspogrammets XML-fil -->>
        <dialog label="Import CSV data" >
                <browser id="file" type="file" label="File name"/>
                <!-- [...] -->>
                <preview id="preview" mode="output"/>
        </dialog>
>
```

Observera specifikationen av *mode="output"* i elementet **<preview>**.

```
// In the plugin's JS file
        function preview () {
                // skapa koden som används för förhandsgranskningen
                printout (true);
        }

        function printout (is_preview) {
                // genererar bara en rubrik om is_preview==false
                if (!is_preview) {
                        new Header ("This is a caption").print ();
                }
                echo ('rk.print (result)');
        }
```

10.2.2 Förhandsgranskningar av (importerad) data

> NOT
> Det här avsnittet beskriver hur förhandsgranskningsfunktionalitet läggs till i instickspogram som skapar (importerar) data. Vi rekommenderar att du läser det separata avsnittet om förhandsgranskning av diagram innan det här avsnittet.

Att skapa en förhandsgranskning av importerad data (vilken typ av data som helst som **rk.edit()** kan hantera), liknar mycket att skapa en förhandsgranskning av diagram. Följande avkortade exemplet bör hjälpa till att illustrera hur en dataförhandsgranskning skapas:

```
<!-- I instickspogrammets XML-fil -->>
        <dialog label="Import CSV data" >
                <browser id="file" type="file" label="File name"/>
                <!-- [...] -->>
                <preview id="preview" active="true" mode="data"/>
        </dialog>
>
```

Observera att elementet **<preview>** anger *mode="data"* den här gången. *active="true"* gör helt enkelt förhandsgranskningen normalt aktiv.

```
// In the plugin's JS file
        function preview () {
                // genererar koden använd för förhandsgranskning
                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 () {
                // [...]
        }
```

Återigen genererar funktionen **preview()** nästan samma R-kod som funktionen **calculate()**, så vi skapar hjälpfunktionen **doCalculate()** för att plocka ut de gemensamma delarna. Det viktigaste att observera är att importerad data måste tilldelas till objektet som heter *preview_data* (inne i den aktuella, lokala, omgivningen). *Allt annat sker automatiskt* (grovt sett anropar RKWard **rk.edit(preview_data)**, omgivet av ett anrop till **.rk.with.window.hints()**).

> NOT
> Medan förhandsgranskningar är en utmärkt funktion, kräver de resurser. I fallet med dataförhandsgranskningar, kan det finnas fall då förhandsgranskningar kan orsaka betydande prestandaproblem. Det kan vara för import av mycket stora datamängder (som helt enkelt är för stora att öppna för redigering i RKWards editorfönster), men också för ″normala″ datamängder som kan importeras felaktigt, vilket skapar ett mycket stort antal rader eller kolumner. *Det rekommenderas starkt att preview_data* begränsas till en dimension som ger en användbar förhandsgranskning, utan risk att skapa märkbara prestandaproblem (t.ex. 50 rader och 50 kolumner bör vara mer än tillräckligt i de flesta fall).

10.2.3 Anpassade förhandsgranskningar

Elementet **<preview>** kan användas för att skapa förhandsgranskningar av godtycklig typ av ″dokumentfönster″ som kan anslutas till RKWards arbetsrymd. Förutom diagram och datafönster, inkluderar det HTML-filer, R-skript och fönster med objektsammanfattningar. För de senare, måste **<preview mode=″custom″>** användas.

Om du har läst avsnitten som beskriver förhandsgranskningar av diagram och data, bör du ha en allmän idé om proceduren, men förhandsgranskningar av typen ″custom″ kräver något mer manuellt arbete bakom kulisserna. Den viktigaste R-funktionen att titta på här är **rk.assign.preview.data()**. Följande korta listning visar hur den genererade R-koden (för förhandsgranskning) skulle kunna se ut för ett insticksprogram som skapar textfilutmatning:

```
## Att genereras i kodsektionen preview() i ett insticksprogram
        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)
        })
```

Här ska värdet *SOMEID* hämtas från egenskapen *id* i elementet **<preview>**, dvs. genom att använda **getString (″preview.id″)** i insticksprogrammets .js-fil.

10.3 Sammanhangsberoende insticksprogram

Hittills har vi antagit att alla insticksprogram alltid är meningsfulla, och alla placerade i huvudmenyn. Dock är vissa insticksprogram bara (eller mer) meningsfulla i vissa sammanhang. Exempelvis är ett insticksprogram för att exportera innehållet i en R X11-grafikenhet uppenbarligen mest användbar när den är placerad i menyn för en X11-enhet, inte i huvudmenyraden. Dessutom bör ett sådant insticksprogram känna till enhetsnumret som den ska arbeta med, utan att behöva fråga användaren om det.

Vi kallar sådana insticksprogram sammanhangsberoende. På motsvarande sätt är de inte (eller inte bara) placerade i huvudhierarkin i `.pluginmap`-filen med **<hierarchy>**, utan istället i elementet **<context>**. Hittills stöds bara två olika sammanhang (fler kommer senare): x11 och filimport. Vi hanterar dem i tur och ordning. Även om du bara är intresserad av importsammanhanget, läs också gärna avsnittet om x11-sammanhanget, eftersom det är något mer utvecklat.

10.3.1 X11-enhetssammanhang

För att använda ett insticksprogram i sammanhang med en x11-enhet, det vill säga placera det i menyraden för fönstret som man får när `x11()` anropas i terminalen, deklarera det först som vanligt i filen `.pluginmap`:

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

Dock behövs det inte definieras i hierarkin (det går, om det också är meningsfullt som ett insticksprogram på toppnivå):

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

Lägg istället till en definition av sammanhanget ”x11”, och lägg till det i menyn där:

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

I insticksprogrammets XML-sektion logic, kan nu två egenskaper deklareras med **<external>**: *devnum* och *context* (om deklarerad) sätts till *”x11”* när insticksprogrammet startas i det sammanhanget. *devnum* sätts till grafikenhetens nummer som ska arbetas med. Och det är allt.

10.3.2 Importdatasammanhang

Innan du läser det här avsnittet, se till att du har läst avsnittet om X11-enhetssammanhang, eftersom det förklarar grundkoncepten.

Sammanhanget *”import”* används för att deklarera filterinsticksprogram för importfiler. De placeras helt enkelt i ett sammanhang med *id=”import”* i `.pluginmap`-filen. Dock finns det en extra knepighet när sådana insticksprogram deklareras: För att erbjuda en enhetlig filvalsdialogruta för alla filtyper som stöds, måste en del extra information deklareras för komponenten:

```
<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>
```

Egenskapslinjen talar helt enkelt om att de tillhörande filnamnsändelserna för XYZ-filer är `*.xyz` eller `*.zyx`, och att filtret ska namnges ‘ XYZ data files’ i filvalsdialogrutan.

Du kan deklarerar två egenskaper med **<external>** i insticksprogrammet. *`filename`* sätts till den valda filens namn, och *`context`* sätts till *`"import"`*.

10.4 Begära information från R

I vissa fall kan man vilja hämta ytterligare information från R, som ska presenteras i insticksprogrammets användargränssnitt. Man kanske exempelvis vill erbjuda ett urval av nivåer för en faktor som användaren har valt att analysera. Från version 0.6.2 av RKWard är det möjligt att göra det. Innan vi börjar är det viktigt att du är medveten om några förbehåll:

R-kod som körs inne i insticksprogrammets logik för användargränssnittet utvärderas i R:s händelsesnurra, vilket betyder att de kan köras *medan* andra beräkningar pågår. Det görs för att se till att insticksprogrammets användargränssnitt är användbart även när R är upptaget med att göra andra saker. Dock gör detta att det är mycket viktigt att koden inte har några sidoeffekter. I synnerhet:

- Gör *inte* några tilldelningar i .GlobalEnv eller några andra icke-lokala omgivningar.
- Skriv *inte* ut någonting i utdatafilen.
- Rita *inte* någonting på skärmen.
- I allmänhet, gör *ingenting* som får sidoeffekter. Koden kan *läsa in information*, inte *”göra”* någonting.

Med det i åtanke, här är det allmänna mönstret. Det används inne i en sektion med skriptbaserad logik för användargränssnittet:

```
<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;  ↩
                             // ett annat resultat är på väg ↩
                             att anlända
                        if (typeof (result) == "undefined") ↩
                             {
                                gui.setListValue ("selector ↩
                                    .available", Array (" ↩
                                    ERROR"));
                                return;
                        }
                        gui.setValue ("selector.enabled",  ↩
                            1);
                        gui.setListValue ("selector. ↩
                            available", result);
                }
        ]]></script>
```

Här är *variable* en egenskap som innehåller ett objektnamn (t.ex. inne i en **<varslot>**). Så snart den ändras, vill man uppdatera visningen av nivåer inne i **<valueselector>**, benämnd *selector*. Nyckelfunktionen här är **doRCommand()**, som har kommandosträngen att köra som första parameter, och namnet på en funktion att anropa när kommandot är klart som andra parameter. Observera att kommandot körs asynkront, och det gör saker och ting lite mer komplicerat. Man vill åtminstone försäkra sig om att **<valueselector>** förblir inaktiverad medan den inte innehåller aktuell information. En annan sak är att man potentiellt kan ha köat mer än ett kommando innan det första resultatet levereras. Det är därför varje kommando ges en ”id”, som vi lagrar i *ast_command_id* för senare referens.

När kommandot är klart, anropas det angivna återanropet (*commandFinished* i det här fallet) med två parametrar: Själva resultatet, och id för motsvarande kommando. Resultatet har en typ som liknar representationen i R, dvs. ett numeriskt fält, om resultatet är numeriskt, etc. Det kan till och med vara en **list()** i R, men i det här fallet representeras det som en **Array()** i JS utan namn.

Observera att till och med det här exemplet är något förenklat. I verkligheten bör man vidta ytterligare försiktighetsåtgärder, för att t.ex. undvika att lägga till ett extremt antal nivåer i väljaren. De goda nyheterna är att du troligtvis inte behöver göra allt själv. Exemplet ovan kommer från instickspogrammet **rkward::level_select**, som du helt enkelt kan inbädda i ditt eget instickspogram. Det låter dig till och med ange ett annat uttryck att köra istället för **levels()**.

10.5 Referera till det aktuella objektet eller aktuella filen

I många instickspogram är det önskvärt att arbeta med det aktuella objektet, ‘current’. Exempelvis skulle ett ‘sorteringsinstickspogram’ kunna välja data.frame som för närvarande redigeras för sortering i förväg. Namnet på det aktuella objektet är tillgängligt för instickspogram som en fördefinierad egenskap vid namn *current_object*. Det går att ansluta till egenskapen på vanligt sätt. Om inget objekt är aktuellt, utvärderas egenskapen till en tom sträng. På liknande sätt är webbadressen för den aktuella skriptfilen tillgänglig som en fördefinierad egenskap som kallas *current_filename*. Egenskapen är tom om ingen skriptfil för närvarande redigeras, eller om skriptfilen inte ännu har sparats.

För närvarande kan *current_object* bara ha klassen `data.frame`, men förlita dig inte på det, eftersom det kommer att utökas till andra datatyper i framtiden. Om du bara är intresserad av objekt av klassen `data.frame`, anslut istället till egenskapen *current_dataframe*. Som alternativ kan typkrav påtvingas genom att använda lämpliga begränsningar för de använda **<varslot>**, eller genom att använda skriptlogik för det grafiska användargränssnittet.

10.6 Repetera (ett antal) alternativ

Ibland vill man upprepa ett antal alternativ för att godtyckligt antal objekt. Antag t.ex. att du vill implementera ett instickspogram för att sortera en data.frame. Du vill tillåta sortering enligt ett godtyckligt antal kolumner (i händelse av liket i den första kolumnen eller de första kolumnerna). Det skulle helt enkelt kunna realiseras genom att tillåta att användare väljer flera variabler i en **<varslot>** med *multi="true"*. Men om du vill utöka det, t.ex. tillåta att användaren anger om varje variabel ska konverteras till tecken eller ett nummer, eller om sorteringen ska vara stigande eller fallande, behövs större flexibilitet. Andra exempel skulle kunna vara uppritning av flera linjer i ett diagram (med möjlighet att välja objekt, linjestil, linjefärg, etc. för varje linje), eller ange en avbildning för att koda om från en mängd gamla värden till nya.

Här kommer **<optionset>** in. Låt oss föst ta en titt på ett enkelt exempel:

```
<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>
```

Här skapade vi ett användargränssnitt för att ange ett antal personer (t.ex. upphovsmän). Användargränssnittet gräver minst en post (*min_rows="1"*). Inne i elementet **<optionset>** börjar vi med att ange innehållet med **<content>**, dvs, de element som hör till alternativmängden. Du bör redan vara bekant med de flesta elementen inne i **<content>**.

Därefter anger vi de intressanta variabler som vi vill läsa från alternativmängden i vår JS-fil. Eftersom vi hanterar ett godtyckligt antal objekt, kan vi inte bara läsa `getString ("firstname")` i JS. Istället anger vi en **<optioncolumn>** för varje intressant värde. För den första optioncolumn i exemplet, betyder **<connect="firstname.text">** att innehållet i elementet **<input>**, "firstname", läses för varje objekt. Alla **<optioncolumn>** där en *label* anges visas på skärmen i en kolumn med den beteckningen. Nu kan vi hämta förnamnen på alla upphovsmän genom att använda `getList("set.firstname")`, `getList("set.lastnames")` för efternamnen, och `getList("set.gender")` för ett strängfält med "m" eller "f".

Observera att det inte finns några begränsningar för vad som kan placeras inne i **<optionset>**. Det går till och med att använda inbäddade komponenter. Precis som med alla andra element, är det enda man måste göra att samla utmatningsvariablerna av intresse i en specifikation med **<optioncolumn>**. I fallet med inbäddade instickspogram, är det ofta en sektion med egenskapen "code". Till exempel:

```
<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>
```

Man kan förstås också använda gränssnittslogik inne i ett optionset. Det finns två sätt att göra det: Man kan skapa en anslutning (eller ett skript) i huvudsektionen **<logic>** av insticksprogrammet, som vanligt. Dock måste användargränssnittets element i innehållsregionen kommas åt som (t.ex.) ″set.contents.firstname.XYZ″. Observera prefixet ″set″ (det *id* som har tilldelats till mängden och ″contents″). Som alternativ kan en separat sektion läggas till som ett underliggande objekt till **<optionset>** med **<logic>**. I detta fall, adresseras alla *id* relativt till innehållsregionen, t.ex. ″firstname.XYZ″. Endast elementet **<script>** tillåts inte i logiksektionen för ett optionset. Om skript ska användas, måste insticksprogrammets huvudsektion **<logic>** utnyttjas.

> NOT
> När skriptlogik används i optionset, är allt man kan göra att komma åt *aktuell* innehållsregionen. Sålunda är den typiskt bara användbar för att ansluta element inne i innehållsregionen till varandra. Att ansluta en egenskap utanför **<optionset>** till en egenskap inne i innehållsregionen kan vara användbart för initiering. Dock gäller *inte* ändring av innehållsregionen efter initieringen objekten som användaren redan har definierat, bara objektet i mängden som för närvarande är markerat.

10.6.1 ″Drivna″ optionsets

Hittills har vi betraktat ett **<optionset>** som tillhandahåller knappar för att lägga till eller ta bort objekt. Dock är det i vissa fall mycket naturligare att välja objekt utanför **<optionset>**, och bara tillhandahålla alternativ för att anpassa vissa aspekter av varje objekt i ett **<optionset>**. Antag t.ex. att du vill låta användaren rita upp flera objekt inne i ett diagram. Användaren ska kunna ange linjefärg för varje objekt. Du *skulle* kunna lösa det genom att lägga till en **<varselector>** och **<varslot>** inne i området **<content>**, vilket låter användaren välja ett objekt i taget. Dock blir det mycket färre klick för användaren, om **<varslut multi=”true”>** används *utanför* **<optionset>** istället. Därefter ansluts de utvalda objekten till ett så kallat ″drivet″ optionset. Så här gör man:

```
<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>
```

Vi börjar med att titta på exemplet längst ner. Observera att två specifikationer av **<optioncolumn>** har *external="true"*. Det talar om för RKWard att de kontrolleras från utsidan av **<optionset>**. Här är det enda syftet med alternativkolumnen ″varnames″ att tillhandahålla lättlästa beteckningar vid visning av optionset (den är ansluten till värdet ″shortname″ i egenskapen som innehåller de valda objekten). Syftet med alternativkolumnen ″vars″ är att fungera som ″nyckelkolumn″, enligt specifikationen **<optionset keycolumn="vars"...>**. Det betyder att för varje post i listan, erbjuder mängden en uppsättning alternativ, och alternativen är logiskt kopplade till posterna. Kolumnen är ansluten till egenskapen som innehåller de valda objekten i **<varslot>**. Det vill säga för varje objekt som är valt där, tillåter **<optionset>** att ange linjefärg.

> NOT
> Externa kolumner kan också anslutas med *connect* till egenskaper inne i regionen **<content>**. Dock är det viktigt att observera att optioncolumns deklarerade med *external="true"* aldrig ska ändras inne i **<optionset>**, och optioncolumns deklarerade med *external="false"* (förval) aldrig ska ändras utanför **<optionset>**.

10.6.2 Alternativ: När optionsets inte ska användas

Att använda optionset är ett kraftfullt verktyg, men de kan ibland göra mer skada än nytta, eftersom de adderar betydande komplexitet, både från instickspogramutvecklarens och användarens perspektiv. Tänk dig alltså för en extra gång innan du använder dem. Här är några råd:

- I några enkla fall kan elementet **<matrix>** vara ett användbart lättviktigt alternativ.
- Låt inte instickspogrammet göra för mycket. Vi gav exemplet att använda optionset för ett instickspogram för att rita flera linjer i ett diagram. Men i allmänhet är det inte en god idé att skapa ett instickspogram som producerar enskilda diagram för varje objekt i ett optionset. Låt istället instickspogrammet skapa ett diagram, som användaren kan anropa flera gånger.
- Om du inte förväntar dig mer än två eller tre objekt i en mängd, överväg att upprepa alternativen manuellt istället.

Kapitel 11

Hantera beroenden och kompatibilitetsfrågor

11.1 RKWard versionskompatibilitet

Vi gör vårt bästa för att se till att insticksprogram utvecklade för en gammal version av RKWard fortsätter fungera i senare versioner av RKWard. Dock är det omvända inte alltid sant, eftersom nya funktioner läggs till. Eftersom inte alla användare kör senaste versionen av RKWard, kan det betyda att ditt insticksprogram inte fungerar för alla.

När du är medveten om sådana kompatibilitetsproblem, bör du se till att detta faktum dokumenteras i din `.pluginmap`-fil, genom att använda **<dependencies>**. Elementet **<dependencies>** kan antingen specificeras som ett direkt underliggande objekt till elementet <document> i en `.pluginmap`, eller som underliggande objekt till individuella **<component>**-definitioner. I det första fallet, gäller beroendena *alla* insticksprogram i avbildningen. I det senare fallet bara de individuella komponenterna, **<component>**. Det går också att blanda toppnivå ”globala” och ”specifika” beroenden. I det fallet läggs de ”globala” beroendena till de för individuella komponenter.

Låt oss titta på ett litet exempel:

```
<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>
```

I det här exemplet är det känt att alla insticksprogram kräver minst version 0.5.0c av RKWard. Ett insticksprogram, med *`id="myplugin"`* tillhandahålls i två olika varianter. Den första, förenklade, versionen används för RKWard versioner innan 0.6.1. Den senare utnyttjar funktioner som är nya i RKWard 0.6.1 och används bara i RKWard 0.6.1 och framåt.

Att tillhandahålla alternativa varianter på detta sätt är ett mycket användarvänligt sätt att utnyttja nya funktioner, och fortfarande behålla stöd för tidigare versioner av RKWard. Alternativa

versioner ska dela samma `id` (annars produceras varningar), och kan bara definieras *inne i samma* `.pluginmap`-fil.

Insticksprogram som inte är kompatibla med versionen av RKWard som kör, och som inte levereras med en alternativversion, ignoreras med en varning.

> NOT
> I själva verket är RKWard 0.6.1 den första versionen som tolkar beroenden, och rapporterar beroendefel, överhuvudtaget. Sålunda, i motsats till vad exemplet förespeglar, blir det ingen direkt effekt av att specificera några tidigare versioner i beroenden (men kan ändå vara en god idé i dokumentationssyfte).

Ibland kan det till och med vara möjligt att hantera inkompatibilitetsproblem mellan versioner *inne i* en enda `.pluginmap`-fil, genom att använda elementet **<dependency_check>**, som beskrivs i nästa avsnitt.

11.2 Kompatibilitet med R-version

I likhet med `rkward_min_version` och `rkward_max_version`, tillåter elementet **<dependencies>** att egenskaperna `R_min_version` och `R_max_version` anges. Dock finns följande skillnader:

- Insticksprogram som inte uppfyller kraven på R-version hoppas för närvarande *inte* över när en `.pluginmap`-fil läses. Användaren kan ändå anropa insticksprogrammet, och ser inte någon omedelbar varning (i framtida versioner kommer troligen ett varningsmeddelande visas).
- Som en konsekvens är det alltså *inte* möjligt att definiera alternativa versioner av ett insticksprogram beroende på vilken version av R som kör.
- Dock är det oftast enkelt att uppnå bakåtkompatibilitet som visas nedan. Om du är medveten om kompatibilitetsproblem med R, fundera på att använda den här metoden istället för att definiera ett beroende av en viss version av R.

I många fall är det lätt möjligt att tillhandahålla reducerad funktionalitet, om en viss funktion inte är tillgänglig i versionen av R som kör. Betrakta följande korta exempel på en `.xml`-fil för ett insticksprogram:

```
<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>
```

I exemplet inaktiveras helt enkelt komprimeringsalternativet ″xz″ när den körbara R-versionen är äldre än 2.10.0 (som inte stödde komprimering med xz). Elementet **<dependency_check>** stöder samma egenskaper som elementet **<dependencies>** i `.pluginmap`-filer. Det skapar en Boolesk egenskap, som är sann om de angivna beroendena är uppfyllda, och annars falsk.

11.3 Beroenden av R-paket

Beroenden på specifika R-paket kan definieras, men för RKWard 0.6.1 kontrolleras varken sådana beroenden eller installeras/läses in automatiskt. De visas dock i insticksprogrammets hjälpfiler. Här är ett exempel på en definition:

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

NOT
Försäkra dig alltid om att lägga till lämpliga anrop till `require()`, om insticksprogrammet kräver att vissa paket ska läsas in.

NOT
Om en `.pluginmap` distribueras som ett R-paket, och alla insticksprogram beror på ett visst paket, ska det beroendet definieras på R-paketnivå. Att definiera beroenden av R-paket på nivån för en RKWard `.pluginmap` är mest användbart om bara vissa av insticksprogrammen har beroendet, om paketet inte är tillgängligt från CRAN, eller för en `.pluginmap` som inte distribueras som ett R-paket.

11.4 Beroenden av andra RKWard `.pluginmap`

Om insticksprogram beror på insticksprogram definierade i en annan `.pluginmap` (som *inte* ingår i paketet) kan beroendet definieras så här:

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

För närvarande läses inte en saknad `.pluginmap` in, eller installeras, eller varnas för, men åtminstone visas information om beroenden (och var de kan erhållas) på insticksprogrammens hjälpsidor. Man behöver inte (och ska inte) deklarera beroenden på en `.pluginmap` som levereras med den officiella distributionen av RKWard, eller för en `.pluginmap` som är inne i det egna paketet. Dessutom, om en nödvändig `.pluginmap` distribueras som ett R-paket, ska ett beroende av paketet deklareras (som visas i föregående avsnitt), istället för avbildningen.

För att se till att en nödvändig `.pluginmap` verkligen läses in, använd taggen **<require>** (se referensen för detaljerad information).

11.5 Ett exempel

För att klargöra hur beroendedefinitioner kan blandas, följer här ett kombinerat exempel:

```
<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>
```

Kapitel 12

Översättning av insticksprogram

Hittills hara vi använd några få begrepp rörande översättningar, eller ″i18n″ (kort för ″internationalization″ som har 18 tecken mellan i och n), i förbigående. I det här kapitlet ger vi en mer djupgående redogörelse om i18n-funktionalitet för RKWard-insticksprogram. I de flesta fall behövs *inte* allt detta i dina insticksprogram. Dock kan det vara en god idé att läsa igenom hela kapitlet, eftersom förståelse av begreppen hjälper dig att skapa insticksprogram som är fullständigt översättningsbara, och tillåta översättningar med hög kvalitet.

12.1 Allmänna hänsynstaganden

En viktigt sak att förstå om översättning av programvara, i motsats till översättning av annat textmaterial, är att översättare ofta har svårighet att få en fullständig bild av *vad* de översätter. Översättningar av programvara är av nödvändighet ofta baserad på ganska korta textfragment: Varje beteckning som anges för en **<option>** i en **<radio>**, varje sträng som markeras för översättning med ett funktionsanrop till **i18n()**, utgör en separat ″översättningsenhet″. I allt väsentligt, presenteras varje sådant fragment isolerat för översättaren. Nå, inte fullständigt isolerat, eftersom vi försöker ge översättaren så mycket meningsfullt sammanhang som kan extraheras automatiskt. Men på vissa ställen behöver översättare ytterligare sammanhang för att förstå en sträng, i synnerhet när strängarna är korta.

12.2 i18n i RKWards XML-filer

För XML-filer i RKWard, kommer i18n oftast bara fungera. Om man skriver sin egen **.pluginmap** (t.ex. för ett externt insticksprogram), måste `po_id` anges intill `id` för pluginmap. Det definierar ″meddelandekatalog″ att använda. I allmänhet ska det vara identiskt med `id` för en **.pluginmap**, men om man tillhandahåller flera relaterade **.pluginmap** i ett enda paket, vill man troligen ange ett gemensamt `po_id` för alla. Ett `po_id` för en **.pluginmap**-fil ärvs av alla insticksprogram deklarerade i den, om de inte deklarerar annorlunda `po_id`.

För insticksprogram och hjälpsidor behöver man inte tala om för RKWard vilka strängar som ska översättas, eftersom det i allmänhet är uppenbart av deras användning. Dock bör man hålla utkik efter strängar som kan vara tvetydiga eller kräver viss förklaring för att kunna översättas korrekt, som förklarades ovan. Ange `i18n_context` för strängar som kan ha olika betydelse, på följande sätt:

```
<checkbox id="scale" label="Scale" i18n_context="Show the scale"/>
<checkbox id="scale" label="Scale" i18n_context="Scale the plot"/>
```

Att ange *i18n_context* gör att de två strängarna översätts separat. Annars skulle de dela en enda översättning. Dessutom visas sammanhanget för översättaren. Egenskapen *i18n_context* stöd för alla element som kan ha översättningsbara strängar någonstans, inklusive element som har text inne i sig (t.ex. element som **<text>**).

I andra fall har strängen en enda otvetydig betydelse, men kan ändå behöva en viss förklaring. I detta fall kan en kommetar läggas till som visas för översättare. Exempel kan omfatta:

```
<!-- 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"/>
```

Observera att sådana kommentarer måste föregå elementen de gäller, och måste antingen börja med ″i18n″ eller ″TRANSLATORS:″.

Till sist, i sällsynta fall kan man vilja undanta vissa strängar från översättning. Det kan vara vettigt om man till exempel erbjuder ett val mellan olika R-funktionsnamn via alternativknappar med **<radio>**. Då vill man inte att de ska översättas (men beroende på sammanhang, kanske man istället skulle fundera på att ge dem beskrivande beteckningar):

```
<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>
```

Observera att egenskapen *label* då ska utelämnas, och *noi18n_label* anges istället. Observera också, att i motsats till *i18n_context* och kommentarer, blir instickspogrammet inte kompatibelt med versioner av RKWard tidigare än 0.6.3 om *noi18n_label* används.

12.3 i18n i RKWards JS-filer och sektioner

I motsats till `.xml`-filerna, krävs mer eget arbete för att göra `.js`-filerna i ett instickspogram översättningsbara. Den stora skillnaden är att här finns det inget rimligt sätt att automatiskt avgöra om en sträng är avsedd att visas som en mänskligt läsbar sträng, eller är en bit kod. Man måste alltså markera det själv. Vi har redan visat exempel på det, hela tiden. Här är en fullständigare beskrivning av i18n-funktioner tillgängliga i js-kod, och några tips för komplexare fall:

i18n (msgid, [...])

Den viktigaste funktionen. Markerar strängen för översättning. Strängen (vare sig den översätts eller inte) returneras inom dubbla citationstecken (″″). Ett godtyckligt antal platsmarkörer kan användas i strängen som visas nedan. Att använda sådana platsmarkörer istället för att sammanfoga små delsträngar gör det mycket lättare för översättare:

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

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

Samma som **i18n()** men tillhandahåller dessutom ett meddelandesammanhang:

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

i18np (msgid_singularis, msgid_pluralis, n, [...])

Samma som **i18n()** men för meddelanden som kan skilja sig i singularis och pluralis (och vissa språk skiljer på ännu fler numerus). Observera att precis som med **i18n()** kan ett godtyckligt antal ersättningar användas, men den första (’%1’) krävs, och måste vara ett heltal.

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

i18ncp (msgctxt, msgid_singularis, msgid_pluralis, n, [...])

i18np() med tillagt meddelandesammanhang.

comment (kommentar, [indentering])

Ekar en kodkommentar markerad för översättning. I motsats till övriga i18n()-funktioner citeras den inte, men tecknet ’#’ läggs till på varje kommentarrad.

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

För att lägga till kommentarer för översättarna (se ovan för en diskussion om skillnaderna mellan kommentarer och sammanhang), lägg till en kommentar som börjar med ”i18n:” eller ”translators:” direkt ovanför **i18n()**-anropet som ska kommenteras, till exempel:

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

12.3.1 i18n och citationstecken

I de flesta fall behöver man inte bekymra sig om hur i18n() beter sig när det gäller citationstecken. Eftersom översättningsbara strängar typiskt är stränglitteraler, är det helt rätt att citera dem, och det sparar en del skrivarbete. Dessutom är i18n()-strängar skyddade från duplicerade citationstecken i funktioner såsom **makeHeaderCode()/Header()** som normalt citerar sina argument. Väsentligen fungerar det genom att först skicka den översatta strängen via **quote()** (för att citera den) och därefter genom **noquote()** (för att skydda den från ytterligare citering). Om en översättningsbar sträng som inte är citerad behövs, använd **i18n(noquote (”My message”))**. Om en översättningsbar sträng måste citeras en andra gång, skicka den genom **quote()** *två gånger*.

Trots det, är det i allmänhet inte en god idé att göra bitar som funktionsnamn eller variabelnamn översättningsbara. För det första är R, programspråket, inneboende på engelska, och det finns ingen internationalisering av själva språket. Kodkommentarer är en annan best, men man bör använda funktionen **comment()** för dem. För det andra, genom att göra syntaktiskt relevanta delar av den genererade koden översättningsbara, kan översättningar faktiskt orsaka fel i instickspogrammet, exempelvis om en intet ont anande översättare översätter en sträng som är avsedd som ett variabelnamn med två separata ord skilda åt med ett mellanslag.

12.4 Underhåll av översättningar

Nu när instickspogrammet har gjorts översättningsbart, hur får man det faktiskt översatt? I allmänhet behöver man bara bekymra sig om det när ett externt instickspogram utvecklas. För instickspogram i RKWards huvudarkiv, görs allt magiskt åt dig. Här är det grundläggande arbetsflödet för externa instickspogram. Observera att man måste ha ”gettext”-verktygen installerade:

- Markera alla strängar, tillhandahåll sammanhang och kommentarer där det behövs

- Kör **python3 scripts/update_plugin_messages.py --extract-only /sökväg/till/min.pluginmap**. Skriptet scripts/update_plugin_messages.py är för närvarande inte en del av källkodsutgåvorna, men finns när källkodsarkivet checkas ut.
- Distribuera den resulterande filen **rkward__*POID*.pot** till översättarna. För externa insticksprogram rekommenderas att placera den i underkatalogen ″po″ i inst/rkward.
- Översättaren öppnar filen i ett översättningsverktyg såsom **lokalize**. Även om du inte tänker göra någon översättning själv, bör du prova steget. Bläddra igenom de extraherade stängarna och titta efter problem eller tvetydigheter.
- Översättaren sparar översättningar som **rkward__*POID*.*xx*.po** (där *xx* är språkkoden), och skickar tillbaka den till dig.
- Kopiera **rkward__*POID*.*xx*.po** till din källkod, intill **rkward__*POID*.pot**. Kör **python3 scripts/update_plugin_messages.py /path/to/my.pluginmap** (observera: utan *--extract-only* den här gången). Det sammanfogar översättningen med eventuella senare ändringar av strängarna, kompilerar översättningen och installerar den i ***DIR_OF_PLUGINMAP*/po/*xx*/LC_MESSAGES/rkward__*POID*.mo** (där *xx* återigen är språkkoden).
- Du bör också inkludera den okompilerade översättningen (dvs. **rkward__*POID*.*xx*.po**) i distributionen, i underkatalogen ″po″.
- För alla uppdateringar av insticksprogrammet, kör **python3 scripts/update_plugin_messages.py /path/to/my.pluginmap** för att uppdatera .pot-filen, liksom också de befintliga .po-filerna, och de kompilerade meddelandekatalogerna.

12.5 Skriva översättningar för insticksprogram

Vi antar att du kan ditt hantverk som översättare, eller är villig att läsa på om det på annat håll. Dock några få ord särskilt om översättning av RKWard insticksprogram:

- RKWard insticksprogram gick inte att översätta förrän version 0.6.3, och var i de flesta fall inte skrivna med i18n i åtanke innan dess. Alltså kommer du att stöta på ganska många fler tvetydiga strängar, och andra problem med i18n, än i andra mogna projekt. Gå inte bara runt dem i tysthet, utan låt oss (eller underhållsansvariga för insticksprogrammet) få veta, så att vi kan fixa problemen.
- Många insticksprogram i RKWard hänvisar till mycket speciella termer från datahantering och statistik, men också från andra vetenskapliga fält. I många fall kräver en god översättning åtminstone grundkunskaper inom dessa fält. I vissa fall *finns* ingen god översättning för en teknisk term, och det bästa alternativet är att lämna termen oöversatt, eller inkludera den engelska termen inom parentes. Fokusera inte alltför mycket på att uppnå 100 % översatta strängar, utan fokusera på att erbjuda en bra översättning, även om det betyder att hoppa över vissa strängar (eller till och med hoppa över vissa meddelandekataloger i sin helhet). Andra användare kan ha möjlighet att fylla i eventuella luckor för tekniska termer.

Kapitel 13

Information om upphovsman, licens och version

Så du har skrivit ett antal insticksprogram och är klar att att dela med dig av ditt arbete. För att vara säker på att användare vet vad ditt arbete handlar om, med vilka villkor de kan använda och distribuera det, och vem de ska kontakta om problem eller idéer, bör du lägga till en del information *om* insticksprogrammen. Det kan göras med elementet **<about>**, som antingen kan användas i en `.pluginmap` eller i .xml-filer i enskilda insticksprogram (i båda fall som ett direkt underliggande objekt till dokumenttaggen). När det anges i en `.pluginmap` gäller det alla insticksprogram. Om **<about>** anges på båda ställen, överskrider informationen i **<about>** i insticksprogrammets `.xml`-fil den i `.pluginmap`-filen. Det går också att lägga till elementet **<about>** på .rkh-sidor, som inte är kopplade till ett insticksprogram, om det finns behov av det.

Nedan är ett exempel på en `.pluginmap`-fil med bara några få förklaringar. I tveksamma fall, kan mer information finnas tillgänglig i referensen.

```
<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>
```

Det mesta av det här bör vara självförklarande, så vi diskuterar inte varenda tagg-element. Men låt oss ta en titt på en del detaljer som troligen behöver några kommentarer för bättre förståelse.

Elementet *category* i **<about>** kan definieras ganska fritt, men bör vara meningsfullt, eftersom det är avsett att ordna instickspogram i grupper. Alla andra egenskaper i den inledande taggen krävs, och måste fyllas i med rimligt innehåll.

Åtminstone en **<author>** med en giltig e-postadress och rollen 'aut' ('author') måste också anges. Ifall instickspogrammet orsakar problem, eller någon skulle vilja dela sin tacksamhet med dig, bör det vara enkelt att kontakta någon som är inblandad. För ytterligare information om andra giltiga roller, såsom 'ctb' för bidragsgivare av kod eller 'cre' för paketunderhåll, se R-dokumentation om person().

> **NOT**
> Kom ihåg att det går att använda **<include>** och/eller **<insert>** för att upprepa information i flera `.xml`-filer (t.ex. information om en upphovsman som är inblandad i flera instickspogram). Mer information.

> **TIPS**
> Du måste inte skriva XML-koden för hand. Om du använder funktionen `rk.plugin.skeleton()` från paketet rkwarddev och tillhandahåller all nödvändig information via alternativet *about*, skapas automatiskt en `.pluginmap`-fil med en fungerande <about>-sektion åt dig.

Kapitel 14

Dela med dig av ditt arbete med andra

14.1 Externa insticksprogram

Från version 0.5.5 tillhandahåller RKWard ett bekvämt sätt att installera ytterligare insticksprogram från tredje part, som inte hör till själva paketet. Vi kallar dem 'externa insticksprogram'. De levereras i form av ett R-paket, och kan direkt hanteras via de vanliga pakethanteringsfunktionerna i R och/eller RKWard.

Det här avsnittet av dokumentationen beskriver hur externa insticksprogram ska paketeras så att RKWard kan använda dem. Hur själva insticksprogrammet skapas är förstås identiskt med föregående avsnitt. Det vill säga, det är troligen bäst att först skriva ett insticksprogram som fungerar, och därefter komma tillbaka hit för att ta reda på hur man distribuerar det.

Eftersom externa insticksprogram är en relativt sen funktion, kan detaljer av detta troligen komma att ändras i framtida utgåvor. Du är välkommen att bidra med dina idéer för att förbättra processen.

> TIPS
> Dokumentationen förklarar detaljerna hos externa insticksprogram så att du kan lära dig hur de fungerar. Förutom det, ta en titt på paketet rkwarddev, som konstruerats för att automatisera en stor del av skrivprocessen.

14.2 Varför externa insticksprogram?

Antal paket för att utöka funktionaliteten hos R är redan enormt, och växande. Å ena sidan vill vi uppmuntra dig att skriva insticksprogram även för de mest specialiserade uppgifter som behöver lösas. Å andra sidan ska inte den vanliga användaren behöva gå vilse i jättestora menyträd fulla av okända statistiska termer. Därför verkade det rimligt att låta hanteringen av insticksprogram i RKWard också vara riktigt modulär. RKWard-gruppen underhåller sitt eget öppna paketarkiv på http://files.kde.org/rkward/R, avsett att ta hand om externa insticksprogram.

Som en tumregel, bör insticksprogram som verkar tjäna ett syfte som används i stor utsträckning (t.ex. t-tester) ingå i det centrala paketet, medan de som betjänar en ganska begränsad grupp med särskilda intressen ska tillhandahållas som ett valfritt paket. För dig, som upphovsman till insticksprogram, är den bästa metoden att helt enkelt börja med ett externt insticksprogram.

14.3 Strukturen hos ett insticksprogrampaket

För att externa insticksprogram ska installeras och fungera riktigt, måste de följa några strukturella tumregler när det gäller deras filhierarki.

14.3.1 Filhierarki

Låt oss ta en titt på en prototypliknande filhierarki för ett komplicerat insticksprogramarkiv. Man behöver inte inkludera alla de här katalogerna och/eller filerna för att ett insticksprogram ska fungera (läs vidare för att ta reda på vad som är absolut nödvändigt). Betrakta det här som en 'bästa metodens' exempel:

```
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
```

> **NOT**
> I exemplet ska alla förekomster av plugin_name, testsuite_name och sometest_name ersättas med de riktiga namnen. Dessutom är ll en platsmarkör för en språkförkortning (t.ex. 'de', 'en' eller 'sv').

> **TIPS**
> Du behöver inte skapa filhierarkin för hand. Om funktionen `rk.plugin.skeleton()` från paketet rkwarddev, skapar den automatiskt alla nödvändiga filer och kataloger åt dig, utom katalogen `po` som skapas och hanteras av översättningsskriptet.

14.3.1.1 Grundläggande insticksprogramkomponenter

Det är nödvändig att inkludera minst tre filer: en `.pluginmap`, en .xml-beskrivning av ett insticksprogram, och en .js-fil för ett insticksprogram. Det vill säga, till och med katalogen ″plugins″ är valfri. Den kan dock hjälpa till att ge filerna en viss ordning, särskilt om mer än ett insticksprogram eller dialog inkluderas i arkivet, vilket förstås inte är något problem. Det går att ha så många kataloger för själva insticksprogramfilerna som anses lämpligt, de måste bara likna respektive `.pluginmap`. Det är också till och med möjligt att inkludera flera `.pluginmap`-filer, om det passar behoven, men då bör alla inkluderas i 'plugin_name.pluginmap'.

Varje R-paket måste ha en giltig beskrivningsfil, `DESCRIPTION`, som också är väsentlig för att RKWard ska känna igen att den tillhandahåller ett insticksprogram. Det mesta av informationen den bär med sig behövs också i insticksprogrammets meta-information och möjligen beroenden, men med ett annat format (R-dokumentationen förklarar beskrivningsfilen DESCRIPTION i detalj).

Förutom det allmänna innehållet i `DESCRIPTION`-filen, se till att också inkludera raden 'Enhances: rkward'. Det gör att RKWard automatiskt söker igenom paketet efter insticksprogram om det är installerat. Ett exempel på en `DESCRIPTION`-fil ser ut så här:

```
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"))
```

> **TIPS**
> Du måste inte skriva filen för hand. Om du använder funktionen `rk.plugin.skeleton()` från paketet rkwarddev och tillhandahåller all nödvändig information via alternativet *`about`*, skapas automatiskt en fungerande 'DESCRIPTION'-fil åt dig.

14.3.1.2 Ytterligare information (valfri)

`ChangeLog`, `README`, `AUTHORS`, `LICENSE` bör vara självförklarliga och är helt valfria. I själva verket tolkas de inte av RKWard, så de är istället avsedda att innehålla ytterligare information som kan vara relevant för t.ex. distributörer. Det mesta av deras relevanta innehåll (erkännanden av upphovsmän, licensvillkor, etc.)är dock ändå inkluderade i själva insticksprogrammets filer (se avsnittet om meta-information). Observera att alla filerna skulle också kunna placeras någonstans i katalogen ″Inst″, om man inte vill att de bara ska vara tillgängliga i källkodsarkivet utan också i det installerade paketet.

14.3.1.3 Automatiserad test av insticksprogram (valfri)

En annan valfri katalog är ″tests″, som är avsett att tillhandahålla filer som behövs för automatiserad test av insticksprogram. Testerna är användbara för att snabbt kontrollera om insticksprogrammet fortfarande fungerar med nya versioner av R eller RKWard. Om du vill inkludera tester, bör du verkligen begränsa dig till namnkonventionen och hierarkin som visas här. Det vill säga, tester ska finnas i en katalog som heter `tests`, som inkluderar filen `testsuite.R` och en katalog med teststandarder namngivna efter lämplig testsvit. Du kan dock tillhandahålla mer än en testsvit: I så fall, om du inte vill lägga till alla i en enda `testsuite.R`, kan de t.ex. delas upp i en fil för varje testsvit och en `testsuite.R` skapas som har anrop till varje svit med `source()`. I båda fall, skapa separata underkataloger med teststandarder för varje definierad svit.

Fördelen med att upprätthålla strukturen är att tester av insticksprogram kan helt enkelt köras genom att anropa `rktests.makplugintests()` från paketet rkwardtests utan ytterligare argument. Ta en titt på dokumentationen på nätet i Automated Plugin Testing för ytterligare information.

14.4 Bygga insticksprogrampaketet

Som tidigare förklarats, är externa RKWard-insticksprogram i själva verket R-paket, och därför är paketeringsprocessen identisk. I motsats till ″riktiga″ R-paket, innehåller ett rent insticksprogrampaket inte någon ytterligare R-kod (även om man förstås också kan lägga till RKWard-insticksprogram i vanliga R-paket, genom att använda samma metoder som förklaras här). Det bör göra det ännu enklare att skapa ett fungerade paket, under förutsättning att man har en giltig `DESCRIPTION`-fil och håller sig till filhierarkin som förklaras i tidigare avsnitt.

Det enklaste sättet att faktiskt bygga och prova insticksprogrammet är att använda R-kommandot på kommandoraden, till exempel:

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

> **TIPS**
> Paketet behöver inte byggas så här på kommandoraden. Om funktionen `rk.build.package()` i paketet rkwarddev används, bygger den och/eller kontrollerar insticksprogrammet åt dig.

Kapitel 15

Utveckling av instickspogram med paketet rkwarddev

15.1 Översikt

Att skriva externa instickspogram innefattar att skriva filer på tre språk (XML, JavaScript och R), och att skapa en standardiserad kataloghierarki. För att göra det mycket enklare för villiga utvecklare av instickspogram, tillhandahåller vi paketet rkwarddev. Det tillhandahåller ett antal enkla R-funktioner för att skapa XML-koden för alla dialogelement som flikböcker, kryssrutor, kombinationslistor eller filbläddrare, samt funktioner för att skapa JavaScript-kod och RKWard hjälpfiler att börja med. Funktionen `rk.plugin.skeleton()` skapar det förväntade katalogträdet och alla nödvändiga filer där det är meningen de ska finnas.

Paketet är inte normalt installerat, utan måste installeras för hand från RKWards eget arkiv. Du kan antingen göra det genom att använda det grafiska användargränssnittet (**Inställningar** → **Anpassa paket**), eller från en godtycklig R-session som kör:

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

Paketet rkwarddev beror på ett annat litet paket som kallas 'XiMpLe', vilket är en mycket enkel XML-tolk och generator som också finns i samma arkiv.

Hela dokumentationen på PDF-format hittas också där. En mer detaljerad introduktion till att arbeta med paketet hittas i rkwarddev vinjetten.

15.2 Praktiskt exempel

För att ge dig en idé om hur det ser ut att 'skapa ett instickspogram med skript', jämfört med den direkta metod som du har sett i tidigare kapitel, skapar vi hela t-test instickspogrammet igen, denna gång bara med R-funktionerna i paketet rkwarddev.

> TIPS
> Paketet lägger till en ny dialogruta i det grafiska användargränssnittet i RKWard under **Arkiv** → **Export** → **Skapa RKWard instickspogramskript**. Som namnet anger, kan instickspogrammallar för vidare redigering skapas med det. Dialogrutan själv skapades i sin tur av ett rkwarddev-skript, som finns i katalogen 'demo' i det installerade paketet och paketkällkoden, som ett ytterligare exempel. Det går också att köra det genom att anropa `demo("skeleton_dialog")`.

15.2.1 Beskrivning av det grafiska användargränssnittet

Man märker omedelbart att arbetsflödet är betydligt annorlunda: I motsats till att skriva XML-koden direkt, börjar man inte med definitionen av **<document>**, utan direkt med elementen i instickspogram som man vill ha i dialogrutan. Det går att tilldela alla gränssnittselement, vare sig de är kryssrutor, kombinationsmenyer, variabelplatser eller någonting annat, till individuella R-objekt och därefter kombinera dessa objekt till det verkliga grafiska användargränssnittet. Paketet har funktioner för varje XML-tagg som kan användas för att definiera instickspogrammets grafiska användargränssnitt och de flesta har till och med samma namn, förutom prefixet `rk.XML.*`. Att exempelvis definiera ett **<varselector>**- och två **<varslot>**-element för variablerna *"x"* och *"y"* i t-testen kan göras med:

```
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")
```

Den mest intressanta detaljen är troligen *source=variabler*: En framträdande funktion i paketet är att alla funktioner kan skapa automatiska id, så att man inte behöver vare sig bry sig om att tänka på *id*-värden eller komma ihåg dem för att referera till ett specifikt element i instickspogrammet. Man kan helt enkelt ange R-objekten som referens, eftersom alla funktioner som behöver en id från något annat element också kan läsa det från dessa objekt. `rk.XML.varselector()` är något speciell, eftersom den oftast inte har något särskilt innehåll att skapa en id från (den kan göra det, men bara om du anger en beteckning), så vi måste ange ett id-namn. Men `rk.XML.varslot()` skulle inte behöva argumenten *id.name* här, så följande skulle vara nog:

```
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)
```

För att återskapa exempelkoden exakt, skulle alla id värden behöva ställas in för hand. Men eftersom paketet är avsett att göra livet enklare, bryr vi oss inte om det längre.

> **TIPS**
> rkwarddev har möjlighet att automatisera en hel del för att hjälpa till att skapa instickspogram. Dock kanske det är att föredra att inte använda det fullt ut. Om målet är att skapa kod som inte bara fungerar, men är lättläst och jämförbart med genereringsskriptet av en person, bör man fundera på att alltid ange användbara id med *id.name*. Namngivning av R-objekt identiska med dessa id, hjälper också för att få skriptkod som är lätt att förstå.

Om man vill se hur XML-koden för det definierade elementet ser ut om det exporterades till en fil, kan objektet bara anropas enligt namn. Om man nu anropar 'var.x' i R-sessionen, bör man se någonting som liknar det här:

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

Vissa taggar är bara användbara i sammanhang med andra. Därför finns exempelvis inte någon funktion för taggen **<option>**. Istället definieras både alternativknappar och kombinationslistor så att deras alternativ inkluderas som en namngiven lista, där namnen representerar beteckningar som ska visas i dialogrutan, och deras värden är en namngiven vektor som kan ha två poster, *val* för alternativets värde och Boolean *chk* för att ange att alternativet normalt ska kontrolleras.

```
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")
        )
)
```

Resultatet ser ut så här:

```
<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>
```

Allt som saknas för elementen under fliken 'Basic settings' är kryssrutan för parade samplingar, och strukturering av alla elementen i rader och kolumner.

```
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()` är ett ovanligt undantag där funktionsnamnet inte innehåller hela taggnamnet, för att minska skrivbördan för det här ofta använda elementet. Det här är vad `basic.settings` nu innehåller:

```
<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>
```

På ett liknande sätt, skapar följande rader R-objekt för elementen under fliken 'Options', inklusive funktioner för nummerrutor, ramar och utsträckning:

```
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")
```

Nu är allt vi behöver göra att lägga ihop objekten i en flikbok, och placera den i en dialogsektion:

```
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)))
)
```

Vi kan också skapa guidesektionen med dess två sidor genom att använda samma objekt, så att deras id extraheras för taggarna **<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's  ↩
                            generally safe not to assume the
                                variables have equal variances. An  ↩
                                    appropriate correction will be applied  ↩
                                    then.
                                Choosing \"assume equal variances\" may  ↩
                                    increase test-strength, however."),
                        rk.XML.copy(check.eqvar),
                        rk.XML.text("Sometimes it's helpful to get an  ↩
                            estimate of the confidence interval of
                                the difference in means. Below you can  ↩
                                    specify whether one should be shown, and
                                which confidence-level should be applied  ↩
                                    (95% corresponds to a 5% level of
                                significance)."),
                        rk.XML.copy(conf.frame)))
```

Det är allt för det grafiska användargränssnittet. Det globala dokumentet kombineras till sist av `rk.plugin.skeleton()`.

15.2.2 JavaScript-kod

Hittills kanske det inte verkar som användning av paketet rkwarddev har hjälpt så mycket. Det kommer att ändras nu.

För det första, precis som vi inte behövde bry oss om id för elementen när layouten av det grafiska användargränssnittet definierades, behöver vi inte bry oss om namn på JavaScript-variabler i nästa steg. Om man vill ha större kontroll, kan man skriva vanlig JavaScript-kod och få den inklistrad i den genererade filen. Men det är troligen mycket effektivare att göra det på sättet som i rkwarddev.

Framför allt behöver man inte definiera några variabler själv, eftersom `rk.plugin.skeleton()` kan söka igenom XML-koden och automatiskt definiera alla variabler som troligen behövs. Man skulle exempelvis inte bry sig om att inkludera en kryssruta om inte dess värde eller tillstånd senare används. Så vi kan börja skriva den verkliga R-koden som skapar JS omedelbart.

TIPS
Funktionen `rk.JS.scan()` kan också söka igenom befintliga XML-filer efter variabler.

Paketet har några funktioner för JS-kodkonstruktioner som vanligtvis används i RKWard-insticksprogram, såsom funktionen `echo()` eller villkor med `if() {...} else {...}`. Det finns några skillnader mellan JS och R, t.ex. används kommatecken för att konkatenera teckensträngar för funktionen `paste()` i R, medan för `echo()` i JS används '+', och rader måste sluta med ett semikolon. Genom att använda R-funktionerna kan man nästan glömma bort skillnaderna och fortsätta skriva R-kod.

Funktionerna kan acceptera olika klasser av indataobjekt: Antingen vanlig text, R-objekt med XML-kod som ovan, eller i sin tur resultat från några andra JS-funktioner i paketet. I slutändan måste alltid `rk.paste.JS()` anropas, som beter sig på liknande sätt som `paste()`, men beroende på indataobjekt, ersätter dem med deras XML id, JavaScript variabelnamn eller till och med fullständiga JavaScript kodblock.

För t-test exemplet behöver vi två JS-objekt: Ett för att beräkna resultatet, och ett för att skriva ut dem i funktionen `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)
```

Som du kan se, tillhandahåller rkwarddev också en R-implementering av funktionen `echo()`. Den returnerar exakt en teckensträng med en giltig JS-version av sig själv. Du kanske också märker att alla R-objekten här är de vi skapade tidigare. De ersätts automatiskt av sina variabelnamn, så det bör vara riktigt intuitivt. Så snart bara den här ersättningen behövs, kan funktionen `id()` användas, som också returnerar exakt en teckensträng för alla objekt som anges (man kan säga att den beter sig som `paste()` med en mycket specifik objektsubstitution).

Funktionen `js()` är en omgivande funktion som låter dig använda R-villkor, **if(){...} else {...}** som du är van vid. De översätts direkt till JS-kod. Det bevarar också vissa operatorer som **<**, **>=** och **||**, så det går att jämföra R-objekten logiskt utan behov av citering för det mesta. Låt oss ta en titt på det resulterade objektet 'JS.calc', som nu har en teckensträng med följande innehåll:

```
echo("res <- t.test (x=" + vrslCompare + ", y=" + vrslAgainst + ",  ↩
    hypothesis=\"" + radUsngtstth + "\"");
        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");
```

> NOT
> Som alternativ till `if()`-villkor nästlade i `js()`, kan man använda funktionen `ite()`, som beter sig på liknande sätt som `ifelse()` i R. Dock är det oftast svårare att läsa villkorssatser skapade med `ite()`, och de bör ersättas med `js()` så fort det är möjligt.

15.2.3 Insticksavbildning

Det här avsnittet är mycket kort: Vi behöver inte skriva en `.pluginmap` alls, eftersom den kan skapas automatiskt av `rk.plugin.skeleton()`. Menyhierarkin kan anges via alternativet *`plugin map`*:

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

15.2.4 Hjälpsida

Det här avsnittet är också mycket kort: `rk.plugin.skeleton()` kan inte skriva en hel hjälpsida med den information den har. Men den kan söka igenom XML-dokumentet efter element som troligen förtjänar att omnämnas på hjälpsidan, och automatiskt skapa en mall för vårt insticksprogram. Allt vi måste göra efteråt är att skriva några rader för varje sektion som listas.

> TIPS
> Funktionen `rk.rkh.scan()` kan också söka igenom befintliga XML-filer för att skapa en mall för hjälpfilen.

15.2.5 Generera insticksprogrammets filer

Nu kommer det sista steget, då vi lämnar över alla genererade objekt till `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)
```

Filerna skapa normalt i en tillfällig katalog. De tre sista alternativen är inte nödvändiga, men mycket praktiska: *load=TRUE* lägger automatiskt till det nya insticksprogrammet i RKWards inställning (eftersom det finns i en tillfällig katalog, och därför slutar existera när RKWard stängs, tas det automatiskt bort igen av RKWard vid nästa start), *edit=TRUE* öppnar alla skapade filer för redigering i RKWards editorflikar, och *show=TRUE* försöker att direkt starta insticksprogrammet, så att du kan undersöka hur det ser ut utan något klick. Du kan överväga att lägga till *overwrite =TRUE* om du tänker köra skriptet upprepade gånger (t.ex. efter kodändringar), eftersom normalt skrivs inga filer över.

Resultatobjektet 'plugin.dir' innehåller sökvägen till katalogen där insticksprogrammet skapades. Det kan vara användbart i kombination med funktionen `rk.build.package()` för att bygga ett verkligt R-paket, för att dela insticksprogrammet med andra, t.ex. genom att skicka det till RKWard-utvecklingsgruppen för att läggas till i vårt insticksprogramarkiv.

15.2.6 Hela skriptet

För att rekapitulera allt det ovanstående, här är hela skriptet för att skapa det fungerande t-test exemplet. Som tillägg till koden som redan har förklarats, läser det också in paketet vid behov, och använder miljön `local()`, så att de skapade objekten inte hamnar i din nuvarande arbetsrymd (utom 'plugin.dir' förstås):

```
require(rkwarddev)

local({
        variables <- rk.XML.varselector(id.name="vars")
        var.x <- rk.XML.varslot("compare", source=variables, types="number ↩
            ", required=TRUE)
        var.y <- rk.XML.varslot("against", source=variables, types="number ↩
            ", required=TRUE)
        test.hypothesis <- rk.XML.radio("using test hypothesis",
                options=list(
                        "Two-sided"=c(val="two.sided"),
                        "First is greater"=c(val="greater"),
                        "Second is greater"=c(val="less")
                )
        )
        check.paired <- rk.XML.cbox("Paired sample", value="1", un.value ↩
            ="0")
        basic.settings <- rk.XML.row(variables, rk.XML.col(var.x, var.y,  ↩
            test.hypothesis, check.paired))

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

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

        full.wizard <- rk.XML.wizard(
                        label="Two Variable t-Test",
```

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

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

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

plugin.dir <<- rk.plugin.skeleton("t-Test",
        xml=list(
                dialog=full.dialog,
                wizard=full.wizard),
        js=list(
                results.header="Two Variable t-Test",
```

```
                        calculate=JS.calc,
                        printout=JS.print),
                pluginmap=list(
                        name="Two Variable t-Test",
                        hierarchy=list("analysis", "means", "t-Test")),
                load=TRUE,
                edit=TRUE,
                show=TRUE,
                overwrite=TRUE)
})
```

15.3 Lägga till hjälpsidor

Om du vill skriva en hjälpsida för instickspogrammet, är det mest rättframma sättet att göra det att direkt lägga till de specifika instruktionerna i definitionen av XML-elementen de hör till:

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

Texten som ges till parametern *help* kan då hämtas av `rk.rkh.scan()` och skrivas till hjälpsidan för den komponenten i instickspogrammet. För att det ska fungera tekniskt, måste dock `rk.rkh.scan()` veta vilka R-objekt som hör till en komponenten i instickspogrammet. Det är orsaken till att du också måste tillhandahålla parametern *component*, och se till att den är identisk för alla objekt som hör ihop.

Eftersom du oftast kombinerar många objekt i en dialogruta, och också kan vilja återanvända objekt som **<varslot>** för flera komponenter i dina instickspogram, är det möjligt att definiera en komponent globalt med `rk.set.comp()`. Om det är gjort, antas att alla följande objekt som används i skriptet hör till den specifika komponent, tills `rk.set.comp()` anropas igen med ett annat komponentnamn. Då kan parametern *component* utelämnas:

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

För att lägga till allmänna sektioner som **<summary>** eller **<usage>** på hjälpsidan, används funktioner som `rk.rkh.summary()` eller `rk.rkh.usage()`. Deras resultat används sedan för att ange listelement som *summary* eller *usage* i parametern *rkh* för `rk.plugin.component()` och `rk.plugin.skeleton()`.

15.4 Översätta instickspogram

Paketet rkwarddev klarar av att skapa externa instickspogram med fullständigt stöd för i18n. Alla relevanta funktioner som genererar XML-objekt erbjuder en valfri parameter för att ange *i18n_context* eller *noi18n_label*:

```
varComment <- rk.XML.varselector(id.name="vars", i18n=list(comment="Main ↩
    variable selector"))
```

```
varContext <- rk.XML.varselector(id.name="vars", i18n=list(context="Main ↩
    variable selector"))
cboxNoi18n <- rk.XML.cbox(label="Power", id.name="power", i18n=FALSE)
```

Exemplen ovan ger utmatning som ser ut så här:

```
# 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" />
```

Det finns också stöd för översättningsbar JS-kod. I själva verket försöker paketet lägga till anrop till `i18n()` direkt på platser där det oftast är till hjälp. Funktionen `rk.JS.header()` är ett bra exempel:

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

Det producerar följande JS-kod:

```
new Header(i18n("Test results")).print();
```

Men det går också att markera strängar i JS-koden som översättningsbara för hand, genom att använda funktionen `i18n()` precis som man skulle göra om JS-filen skrevs direkt.

Bilaga A

Referens

A.1 Typer av egenskaper och modifierare

På några ställen i den här introduktionen har vi talat om 'egenskaper' hos element i det grafiska användargränssnittet eller annars. I själva verket finns det flera olika typer av egenskaper. Oftast behöver man inte bekymra sig om det, eftersom man kan använda sunt förnuft för att ansluta vilken egenskap som helst till vilken annan egenskap som helst. Dock finns det internt olika typer av egenskaper. När det har betydelse är när vissa särskilda värden hämtas i JS-mallen. I satser som getString (˝id˝)/getBoolean (˝id˝)/getList (˝id˝) kan man också ange vissa så kallade 'modifierare' på följande sätt: `getString (˜id.modifierare˜)`. En modifierare påverkar på vilket sätt värdet skrivs ut. Läs vidare för listan över egenskaper, och de modifierare som var och en tillhandahåller.

Strängegenskaper

Den enklaste typen av egenskap, som helt enkelt används för att innehålla ett textstycke. Modifierare:

Ingen modifierare (˝˝)
: Strängen som den definierades eller tilldelades.

quoted
: Strängen i citerad form (lämplig att skicka till R som tecken).

Booleska egenskaper

Egenskaperna kan antingen vara på eller av, sanna eller falska. Exempelvis egenskaper skapade av <convert>-taggar, samt egenskapen som följer en <checkbox> (se nedan). Följande värden returneras enligt angiven modifierare:

Ingen modifierare (˝˝)
: Normalt returnerar egenskapen 1 om den är sann, och 0 annars. Det rekommenderade sättet att hämta Booleska värden är att använda `getBoolean()`. Observera att för `getString()` returneras strängen ˝0˝ när egenskapen är falsk. Strängen kan utvärderas som sann, inte falsk, i JS.

˝labeled˝
: Returnerar strängen ˝true˝ när sann, ˝false˝ när falsk, eller de egna strängarna som har angivits (typiskt i en <checkbox>).

˝true˝
: Returnerar strängen som om egenskapen var sann, även om den är falsk

˝false˝
: Returnerar strängen som om egenskapen var falsk, även om den är sann

"not"
Returnerar i själva verket en annan Boolesk egenskap, som är det omvända av den nuvarande (dvs. falsk om sann, sann om falsk)

"numeric"
Föråldrad, tillhandahålls för bakåtkompatibilitet. Samma som ingen modifierare "". Returnera "1" om egenskapen är sann, eller "0" om den är falsk.

Heltalsegenskaper

En egenskap konstruerad för att innehålla ett heltalsvärde (men returnerar förstås ändå en sträng med numeriska tecken i JS-mallen). Den accepterar inga modifierare. Används i <spinbox> (se nedan)

Egenskaper för reella tal

En egenskap konstruerad för att innehålla ett reellt numeriskt värde (men returnerar förstås ändå en sträng med numeriska tecken i JS-mallen). Används i <spinbox> (se nedan)

Ingen modifierare ("")
För `getValue() / getString()` returneras samman som "formatted". I framtida versioner kommer det att vara möjligt att erhålla en numerisk representation istället.

"formatted"
Returnerar det formaterade talet (som en sträng).

RObject-egenskaper

En egenskap konstruerad för urval av ett eller flera R-objekt. Används mest framträdande i varselector och varslot. Följande värden returneras enligt angiven modifierare:

Ingen modifierare ("")
Normalt returnerar egenskapen det valda objektets fullständiga namn. Om mer än ett objekt är valt, skiljs objektnamnen åt av radbrytningar ("\n").

"shortname"
Som ovan, men returnerar bara korta namn på objekten. Exempelvis skulle ett objekt inne i en lista bara ges namnet det har inne i listan, utan listans namn.

"label"
Som ovan, men returnerar objektens RKWard beteckningar (om ingen beteckning är tillgänglig, är det samma sak som "shortname").

Egenskaper för stränglistor

Egenskapen innehåller en lista med strängar.

Ingen modifierare ("")
För `getValue()/getString()`, returneras alla strängar åtskilda av "\n". Eventuella "\n"-tecken i varje objekt hanteras som litteralen "
n". Dock är den rekommenderade användningen att hämta värdet med `getList()` istället, som returnerar ett fält av strängar.

"joined"
Returnerar listan som en enda sträng, med objekten sammanfogade av "\n". I motsats till ingen modifierare (""), ändras *inte* de individuella strängarna.

Kodegenskaper

En egenskap som hålls av instickspogram som genererade kod. Den är viktig för inbäddning av instickspogram, för att kunna inbädda koden som genereras av det inbäddade instickspogrammet i koden som genereras av det inbäddande (toppnivå) instickspogrammet. Följande värden returneras enligt angiven modifierare:

Ingen modifierare ("")
Returnerar den fullständiga koden, dvs. sektionerna "preprocess", "calculate", "printout" (men inte "preview") konkatenerade i en sträng.

"preprocess"
Returnerar bara kodens förbehandlingssektion preprocess

"calculate"
Returnerar bara kodens beräkningssektion calculate

"printout"
Returnerar bara kodens utskriftssektion printout

"preview"
Returnerar kodens förhandsgranskningssektion preview

A.2 Element för allmänna syften att använda i vilken XML-fil som helst (`.xml`, `.rkh`, `.pluginmap`)

<snippets>
Tillåten som ett direkt underliggande objekt till noden <document> och bara där. Bör placeras nära filens början. Se avsnittet om att använda snippet. Det får bara finnas ett <snippet>-element. Valfri, inga egenskaper.

<snippet>
Definierar en enskild snippet. Bara tillåten som ett direkt underliggande objekt till elementet <snippets>. Egenskaper:

<id>
En identifierarsträng för denna snippet. Krävs.

<insert>
Infoga innehållet i en <snippet>. Tillåten var som helst. Egenskaper:

<snippet>
Identifierarsträngen för den snippet som ska infogas. Krävs.

<include>
Inkluderar innehållet i en annan XML-fil (allting inne i elementet <document> i den filen). Tillåten var som helst. Egenskaper:

<file>
Filnamnet, relativt till katalogen som den aktuella filen är i. Krävs.

A.3 Element att använda i instickspogrammets XML-beskrivning

Egenskaper som elementen innehåller listas i ett separat avsnitt.

A.3.1 Allmänna element

<document>
Måste finnas i alla beskrivning.xml-filer som rotnoden. Ingen specialfunktion. Inga egenskaper

<about>

Information om insticksprogrammet (upphovsman, licens, etc.). Elementet tillåts i både ett individuellt insticksprograms .xml-fil, och i `.pluginmap`-filer. Se `.pluginmap` filreferens för detaljerad referensinformation, och kapitlet om 'om' information för en introduktion.

<code>

Definierar var JS-mallen för insticksprogrammet kan hittas. Använd bara en gång per fil, som ett direkt underliggande objekt till taggen document. Egenskaper:

file

JS-mallens filnamn, relativt till katalogen som insticksprogrammets XML finns i

<help>

Definierar var hjälpfilen för insticksprogrammet kan hittas. Använd bara en gång per fil, som ett direkt underliggande objekt till taggen document. Egenskaper:

file

Hjälpfilens filnamn, relativt till katalogen som insticksprogrammets XML finns i

<copy>

Kan användas som underliggande objekt (direkt eller indirekt) till huvudelementen för layout, dvs. <dialog> och <wizard>. Används för att kopiera ett helt block av XML-element ett-till-ett. Egenskaper:

id

Id att söka efter. Taggen <copy> söker efter ett tidigare XML-element som har givits samma id, och kopierar det inklusive alla underliggande element.

copy_element_tag_name

I några få fall, vill man ha en nästan exakt kopia, men ändra taggnamnet på elementet som kopieras. Det viktigaste exemplet på det är när man vill kopiera en hel flik, <tab>, från ett dialoggränssnitt till en sida, <page>, i ett guidegränssnitt. I detta fall, skulle man ange copy_element_tag_name=″page″ för att automatiskt utföra konverteringen.

A.3.2 Gränssnittsdefinitioner

<dialog>

Definierar ett gränssnitt av dialogtyp. Placera definitionen av det grafiska användargränssnittet inne i taggen. Använd bara en gång per fil, som ett direkt underliggande objekt till taggen document. Åtminstone en av taggarna ″dialog″ eller ″wizard″ krävs för ett insticksprogram. Egenskaper:

label

Dialogrutans rubrik

recommended

Ska dialogrutan användas som det ″rekommenderade″ gränssnittet (dvs. gränssnittet som normalt visas, om användaren inte har ställt in RKWard att använda ett specifikt förvalt gränssnitt)? Egenskapen har inte någon effekt för närvarande, eftersom den implicit är ″sann″, om inte guiden rekommenderas.

<wizard>

Definierar ett gränssnitt av guidetyp. Placera definitionen av det grafiska användargränssnittet inne i taggen. Använd bara en gång per fil, som ett direkt underliggande objekt till taggen document. Åtminstone en av taggarna ″dialog″ eller ″wizard″ krävs för ett insticksprogram. Accepterar bara taggarna <page> eller <embed> som direkt underliggande objekt. Egenskaper:

label
: Guidens rubrik

recommended
: Ska guiden användas som det ″rekommenderade″ gränssnittet (dvs. gränssnittet som normalt visas, om användaren inte har ställt in RKWard att använda ett specifikt förvalt gränssnitt)? Valfri, förvalt värde är ″false″.

A.3.3 Layoutelement

Alla element i den här sektionen accepterar egenskapen id=″identifierarsträng″. Egenskapen är valfri för alla element. Den kan exempelvis användas för att dölja eller inaktivera hela layoutelementet och alla element som finns i det (se kapitlet om logik i det grafiska användargränssnittet). Id-strängen får inte innehålla ″.″ (punkt) eller ″;″ (semikolon), och bör i allmänhet begränsas till alfanumeriska tecken och understreck (″_″). Bara ytterligare egenskaper listas.

<page>
: Definierar en ny sida i en guide. Tillåts bara som ett direkt underliggande objekt av elementet <wizard>.

<row>
: Alla direkt underliggande objekt till taggen ″row″ placeras från vänster till höger.

<column>
: Alla direkt underliggande objekt till taggen ″column″ placeras uppifrån och ner.

<stretch>
: Normalt upptar elementen i det grafiska användargränssnittet allt tillgängligt utrymme. Om man exempelvis har två kolumner sida vid sida, där den vänstra är fylld med element, medan den högra bara innehåller en ensam alternativknapp, **<radio>**, expanderas **<radio>**-knappen vertikalt även om den inte egentligen behöver det tillgängliga utrymmet, och det ser fult ut. I det här fallet vill man egentligen lägga till ett ″tomrum″ under **<radio>**-knappen. Använd elementet <stretch> för det. Det upptar helt enkelt ett visst utrymme. Använd inte elementet i onödan, oftast är det en god idé att låta elementen i det grafiska användargränssnittet få allt tillgängligt utrymme. Det är bara ibland som layouten blir utsträckt. Elementet <stretch> har inga argument, inte ens ″id″. Det går inte heller att placera några underliggande objekt inne i elementet <stretch> (med andra ord, används det bara som ″<stretch/>″).

<frame>
: Ritar en ram eller ruta omkring sina direkt underliggande objekt. Kan användas för att visuellt gruppera relaterade alternativ. Layouten inne i en ram är uppifrån och ner, om man inte placerar en <row> inne i den. Egenskaper:

  label
  : Ramens rubrik (valfri)

  checkable
  : Ramar kan göras markeringsbara. I detta fall, inaktiveras alla ingående element när ramen avmarkeras, och aktiveras när den är markerad (valfri, förvalt värde ″false″)

  checked
  : Bara för markeringsbara ramar: Ska ramen normalt vara markerad? Förvalt värde är ″true″. Tolkas inte för ramar som inte är markeringsbara.

<tabbook>
: Organiserar elementen i en flikbok. Accepterar bara <tab>-taggar som direkt underliggande objekt.

<tab>

Definierar en sida i en flikbok. Placera flikens definition i det grafiska användargränssnittet inne i taggen. Kan bara användas som ett direkt underliggande objekt till taggen <tabbook>. En <tabbook> måste ha minst två flikar definierade. Egenskaper:

label
Fliksidans rubrik (krävs)

<text>

Visar texten innesluten i taggen i det grafiska användargränssnittet. Viss enkel HTML-liknande markering stöds (i synnerhet <b>, <i>, <p> och
). Håll dock formatering till ett minimum. Att infoga en helt tom rad lägger till en hård radbrytning. Egenskaper:

type
Textens typ. Antingen ″normal″, ″warning″ (varning) eller ″error″ (fel). Det påverkar textens utseende (valfri, förvalt värde är normal)

A.3.4 Aktiva element

Alla element i den här sektionen accepterar egenskapen id=″identifierarsträng″. Egenskapen krävs för alla element. Bara ytterligare egenskaper listas. Id-strängen får inte innehålla ″.″ (punkt).

<varselector>

Tillhandahåller en lista över tillgängliga objekt, där användaren kan välja ett eller flera. Kräver en eller flera <varslot> som motpart för att vara användbar. Egenskaper:

label
Beteckning för en varselector (valfri, förvalt värde är ″Select variable(s)″)

<varslot>

Används tillsammans med en ″varselector″ för att låta användare välja en eller flera variabler. Egenskaper:

label
Beteckning för en varslot (rekommenderas, förvalt värde är ″Variable:″)

source
Den varselector som valet hämtas från (krävs, om man inte ansluter manuellt eller använder source_property)

source_property
En godtycklig egenskap att kopiera värden från, när valknappen klickas. Om den anges, överskrider den egenskapen ″source″.

required
Om det krävs att en varslot innehåller ett giltigt värde för att verkställa koden. Se required-property (valfri, förvalt värde false)

multi
Om en varslot innehåller bara ett (förval, ″false″), eller flera objekt

allow_duplicates
Om en varslot bara kan acceptera unika objekt (förval, ″false″), eller om samma objekt kan läggas till flera gånger.

min_vars
Bara meningsfull om multi=″true″: Minimalt antal variabler som kan markeras för att markeringen ska anses giltig (valfri, förvalt värde ″1″)

min_vars_if_any
Bara meningsfull om multi=″true″: En varslot kan anses giltig om den exempelvis antingen är tom eller innehåller minst två värden. Det anger hur många variabler som måste väljas, om några överhuvudtaget (2 i exemplet). (valfri, förvalt värde ″1″)

max_vars
: Bara meningsfull om multi=″true″: Minimalt antal variabler som kan markeras (valfri, förvalt värde ″0″, vilket betyder inget maximum)

classes
: Om ett eller flera R klassnamn anges (åtskilda av mellanslag (″ ″)) här, accepterar denna varslot bara objekt som hör till klasserna (valfri, *använd med stor försiktighet*, användaren ska inte förhindras att göra giltiga val, och R har *många* olika klasser).

types
: Om en eller flera variabeltyper anges (åtskilda med mellanslag (″ ″)) här, accepterar bara denna varslot objekt av typerna. Giltiga typer är ″unknown″, ″number″, ″string″, ″factor″, ″invalid″. (valfri, *använd med stor försiktighet*, användaren ska inte förhindras att göra giltiga val, och RKWard känner inte alltid till en variabels typ)

num_dimensions
: Antal dimensioner som ett objekt måste ha. ″0″ (förvalt värde) betyder att godtyckligt antal dimensioner är acceptabla. (valfritt, förvalt värde ″0″)

min_length
: Den minimala längden som ett objekt måste ha för att vara acceptabelt. (valfri, förvalt värde ″0″)

max_length
: Den maximala längden som ett objekt måste ha för att vara acceptabelt. (valfri, förvalt värde är det största heltal som kan representeras i systemet)

<valueselector>
: Tillhandahåller en lista över tillgängliga strängar (inte R-objekt) att väljas i en eller flera medföljande <valueslot>. Strängalternativ kan definieras genom att använda taggarna <option> som direkt underliggande objekt (se nedan), eller anges med dynamiska egenskaper. Egenskaper:

label
: Beteckning för en valueselector (valfri, förvalt värde är ingen beteckning)

<valueslot>
: Används tillsammans med en <valueselector> för att låta användaren välja ett eller flera strängobjekt. Elementet är i huvudsak identiskt med <varslot> och delar samma egenskaper, utom de som refererar till egenskaper för acceptabla objekt (dvs. klasser, typer, antal dimensioner, minimal längd, maximal längd).

<radio>
: Definierar en grupp av exkluderande alternativknappar (bara en kan väljas åt gången). Kräver minst två <option>-taggar som direkt underliggande objekt. Inga andra taggar tillåts som underliggande objekt. Egenskaper:

label
: Alternativknapparnas beteckning (rekommenderas, förvalt värde ″Select one:″)

<dropdown>
: Definierar en grupp alternativ där ett och endast ett kan väljas åt gången, med en kombinationslista. Den är funktionellt ekvivalent med en <radio>, men ser annorlunda ut. Kräver minst två <option>-taggar som direkt underliggande objekt. Inga andra taggar tillåts som underliggande objekt. Egenskaper:

label
: Kombinationslistans beteckning (rekommenderas, förvalt värde ″Select one:″)

<select>
: Tillhandahåller en lista över tillgängliga strängar där användaren kan välja ett godtyckligt antal. Strängalternativ kan definieras genom att använda <option>-taggar som direkt underliggande objekt (se nedan), eller anges med dynamiska egenskaper. Egenskaper:

label
Beteckning för <select> (valfri, förval är ingen beteckning)

enkel
Om satt till sant, går det bara att välja ett enda värde, istället för flera värden på en gång (Booleskt, förvalt värde falskt)

<option>

Kan bara användas som ett direkt underliggande objekt till elementen <radio>, <dropdown>, <valueselector> eller <select>. Representerar ett valbart alternativ i en alternativknapp eller kombinationslista. Eftersom elementen <option> alltid ingår som en del av ett av urvalselementen, har de normalt ingen egen ”id”, men se nedan. Egenskaper:

label
Alternativets beteckning (krävs)

value
Strängvärdet det överliggande objektet returnerar om alternativet är markerat eller valt (krävs)

checked
Om alternativet normalt ska vara markerat/valt, ”true” eller ”false”. I en <radio> eller <dropdown>, kan bara ett alternativ anges som `checked="true"`, och om inget alternativ är angett som markerat, blir det första alternativet i det överliggande objektet automatiskt markerat/valt. I en <select> kan godtyckligt antal alternativ anges som markerade. (valfritt, förvalt värde ”false”)

id
Att ange ”id” parametrar för <option>-element är valfritt (och det rekommenderas i själva verket att inte ange ”id”, om det inte verkligen behövs). Dock blir det möjligt att aktivera/inaktivera en <option> dynamiskt om ”id” anges, genom att ansluta till den Booleska egenskapen `id_of_radio.id_of_optionX.enabled`. För närvarande fungerar det bara för egenskaper inne i elementen <radio> och <dropdown>. Alternativen <valueselector> och <select> stöder för närvarande inte ”id”.

<checkbox>

Definierar en kryssruta, dvs. ett enstaka alternativ som antingen kan vara av eller på. Egenskaper:

label
Kryssrutans beteckning (krävs)

value
Värdet som kryssrutan returnerar om den är markerad (krävs)

value_unchecked
Värdet som returneras om kryssrutan inte är markerad (valfritt, förvalt värde är ””, dvs. en tom sträng)

checked
Om alternativet normalt ska vara markerat, ”true” eller ”false” (valfritt, förvalt värde ”false”)

<frame>

Ramelementet används i allmänhet som ett rent layoutelement och det listas i avsnittet om layoutelement. Det kan dock också göras markeringsbart, och sålunda samtidigt fungera som en enkel kryssruta.

<input>

Definierar ett fritt inmatningsfält. Egenskaper:

label
Inmatningsfältets beteckning (krävs)

initial
Textfältets ursprungliga text (valfri, förvalt värde ””, dvs. en tom sträng)

size
Ett av ″small″, ″medium″, or ″large″. ″large″ definierar ett inmatningsfält med flera rader, medan ″small″ och ″medium″ är fält med en rad (valfri, förvalt värde ″medium″)

required
Om det krävs att indata inte är tomt för att verkställa koden. Se required-property (valfri, förvalt värde false)

<matrix>
En tabell för att mata in matrisdata (eller vektorer) i det grafiska användargränssnittet.

> NOT
> Det här inmatningselementet är *inte* optimerat för att mata in eller redigera stora mängder data. Även om det inte finns någon fast gräns för storleken hos matrisen <matrix>, bör den i allmänhet inte överstiga omkring tio rader eller kolumner. Om du förväntar dig mer data, låt användaren välja den som ett R-objekt (vilket kan vara en god idé som ett alternativ vid nästan *alla* tillfällen då ett matriselement används).

Egenskaper:

label
Tabellens beteckning (krävs)

mode
Ett av ″integer″, ″real″, or ″string″. Typ av data som accepteras i tabellen (krävs)

min
Minimalt acceptabelt värde (för matriser av typ ″integer″ eller ″real″) (valfritt, förvalt värde är det minsta värde som kan representeras)

max
Maximalt acceptabelt värde (för matriser av typ ″integer″ eller ″real″) (valfritt, förvalt värde är det största värde som kan representeras)

allow_missings
Om saknade (tomma) värden tillåts i matrisen. Det är underförstått för matriser av typen ″string″ (valfritt, förvalt värde false).

allow_user_resize_columns
När satt till sant, kan användaren lägga till kolumner genom att skriva i (inaktiva) cellerna längst till höger (valfritt, förvalt värde är true).

allow_user_resize_rows
När satt till sant, kan användaren lägga till rader genom att skriva i (inaktiva) cellerna längst ner (valfritt, förvalt värde är true).

rows
Antal rader i matrisen. Har ingen effekt med allow_user_resize_rows=″true″.

> NOT
> Kan också bestämmas genom att ange egenskapen ″rows″

(valfritt, förvalt värde 2).

columns
Antal kolumner i matrisen. Har ingen effekt med allow_user_resize_columns=″true″.

> NOT
> Kan också bestämmas genom att ange egenskapen ″columns″

(valfritt, förvalt värde 2).

min_rows
: Minimalt antal rader i matrisen. Matrisen vägrar att krympa under den här storleken (valfritt, förvalt värde 0, se också: `allow_missings`).

min_columns
: Minimalt antal kolumner i matrisen. Matrisen vägrar att krympa under den här storleken (valfritt, förvalt värde 0, se också: `allow_missings`).

fixed_height
: Tvinga elementet i det grafiska användargränssnittet att behålla sin ursprungliga höjd. Använd det inte i kombination med matriser, där antal rader kan ändras på vilket sätt som helst. Användbar i synnerhet när ett inmatningselement för en vektor skapas (columns="1"). När alternativet är satt till true, visas ingen horisontell rullningslist, även om matrisen överskrider tillgänglig bredd (eftersom det skulle påverka höjden). (valfritt, förvalt värde false).

fixed_width
: Något felaktigt benämnt: Antag att kolumnantalet inte kommer att ändras. Den sista (eller oftast enda) kolumnen sträcks ut för att uppta hela tillgängliga bredden. Använd det inte i kombination med matriser, där antal kolumner kan ändras på vilket sätt som helst. Användbar i synnerhet när ett inmatningselement för en vektor skapas (rows="1"). (valfritt, förvalt värde false).

horiz_headers
: Strängar att använda för den horisontella rubriken, åtskilda av ";". Rubriken döljes om satt till "". (valfri, förvalt värde är kolumnens nummer).

vert_headers
: Strängar att använda för den vertikala rubriken, åtskilda av ";". Rubriken döljes om satt till "". (valfri, förvalt värde är radens nummer).

<optionset>

Ett användargränssnitt för att upprepa en mängd alternativ för ett godtyckligt antal objekt (Introduktion av optionsets). Egenskaper:

min_rows
: Om angiven, markeras mängden som ogiltig, om den inte har åtminstone så här många rader (valfri, heltal).

min_rows_if_any
: Som min_rows, men testas bara om det finns minst en rad (valfri, heltal).

max_rows
: Om angiven kommer mängden att markeras som ogiltig, om den inte har som mest det här antalet rader (valfri, heltal).

keycolumn
: Id för kolumnen som ska fungera som keycolumn. Ett optionset med en (giltig) keycolumn, fungerar som ett "drivet" optionset. Ett optionset utan keycolumn tillåter att manuellt infoga eller ta bort objekt. En keycolumn måste vara markerad som external. (valfri, förvalt värde är ingen keycolumn).

Delelement:

<optioncolumn>
: Deklarerar en optioncolumn i mängden. För varje värde som ska hämtas från ett optionset, måste en separat <optioncolumn> deklareras. Egenskaper:

  id
  : Id för en optioncolumn (krävs, sträng)

  external
  : Sätt till sann, om en optioncolumn kontrolleras utifrån optionset (valfri, Boolean, förvalt värde är "false").

  label
  : Om angiven visas en optioncolumn i en kolumn enligt beteckningen (valfri, sträng, förvalt värde är att inte visa den).

connect
Egenskap att ansluta till optioncolumn to, angiven som id inne i området <content>. För en extern <optioncolumn> ändras motsvarande värde till det externt inställda värdet. För en vanlig (inte extern) <optioncolumn> ändras motsvarande rad i <optioncolumn>-egenskapen när egenskapen ändras inne i innehållsområdet (valfri, sträng, förvalt värde är inte ansluten).

default
Bara för externa kolumner: Värdet att anta för kolumnen, om inget värde är känt för en post. Sällan användbar (valfri, förvalt värde är en tom sträng).

<content>
Deklarera innehållet i användargränssnittet/mängden. Inga egenskaper. Alla vanliga aktiva, passiva och layoutelement tillåts som underliggande namnelement. Dessutom tilläts det särskilda underliggande elementet **<optiondisplay>** i tidigare versioner av RKWard (till och med 0.6.3). Det är föråldrat i RKWard 0.6.4, och ska helt enkelt tas bort från befintliga instickspogram.

<logic>
Valfri specifikation av logik i användargränssnitt som gäller *inne i* innehållsregionen för ett optionset. Se referensen om <logic>.

<browser>
Ett element konstruerat att välja ett enda filnamn (eller katalognamn). Observera att fältet accepterar vilken sträng som helst, även om det är avsett att bara användas för filer:

label
Bläddrarens beteckning (valfri, förvalt värde ″Enter filename″)

initial
Initialvärdet för texten i bläddraren (valfritt, förvalt värde ″″, dvs. en tom sträng)

type
Ett av ″file″, ″dir″ eller ″savefile″. För att välja respektive en befintlig fil, befintlig katalog eller icke-befintlig fil (valfri, förvalt värde ″file″)

allow_urls
Om (icke-lokala) webbadresser kan väljas (valfri, förvalt värde ″false″)

filter
Filtypsfilter, t.ex. (″*.txt *.csv″ för .txt- och .csv-filer). En separat post för ″Alla filer″ läggs till automatiskt (valfritt, förvalt värde är ″″, dvs. Alla filer)

required
Om det krävs att fältet inte är tomt för att verkställa koden. Observera att det inte nödvändigtvis betyder att det valda filnamnet är giltigt. Se required-property (valfri, förvalt värde true)

<saveobject>
Ett element konstruerat för att välja namnet på ett R-objekt att spara i (dvs. i allmänhet inte redan befintligt, i motsats till en varslot):

label
Inmatningsrutans beteckning (valfri, förvalt värde ″Save to:″)

initial
Initialvärdet för texten i inmatningsrutan (valfritt, förvalt värde ″my.data″)

required
Om det krävs att fältet innehåller ett tillåtet objektnamn för att verkställa koden. Se required-property (valfri, förvalt värde true)

checkable
I många användarfall är det valfritt att spara i ett R-objekt. I dessa fall kan en kryssruta integreras i saveobject-elementet genom att använda egenskapen. När den är satt till true, aktiveras/inaktiveras saveobject-elementet av kryssrutan. Se egenskapen active för saveobject (valfri, förvalt värde false)

checked
Bara för markeringsbara saveobject-element: Om objektet normalt är markerat eller aktiverad (valfritt, förvalt värde är ″false″)

<spinbox>

En nummerruta där användaren kan välja ett numeriskt värde, antingen genom att använda direkt tangentbordsinmatning eller små uppåt- och neråtpilar. Egenskaper:

label
Nummerrutans beteckning (rekommenderas, förvalt värde ″Enter value:″)

min
Det minsta värde som användaren får mata in i nummerrutan (valfritt, förval är det minsta värdet som tekniskt kan representeras i nummerrutan)

max
Det största värde som användaren får mata in i nummerrutan (valfritt, förval är det största värdet som tekniskt kan representeras i nummerrutan)

initial
Initialvärdet som visas i nummerrutan (valfritt, förvalt värde ″0″)

type
Antingen ″real″ eller ″integer″. Om nummerrutan accepterar reella tal eller bara heltal (valfritt, förvalt värde är ″real″)

default_precision
Bara meningsfull om nummerrutan har type=″real″. Anger förvalt antal decimalsiffror som visas i nummerrutan (bara så här många avslutande nollor visas). När användaren klickar på uppåt- eller neråtpilen, ändras decimalsiffrorna. Användaren kan dock ändå mata in värden med större precision (se nedan) (valfri, förvalt värde ″2″)

max_precision
Det maximala antalet siffror som kan representeras på ett meningsfullt sätt (valfritt, förvalt värde är ″8″)

<formula>

Det här avancerade elementet tillåter användaren att välja en formel eller interaktionsmängd från valda variabler. För en GLM kan elementet exempelvis användas för att tillåta användaren att ange modellens interaktionstermer. Egenskaper:

fixed_factors
Id för den varslot som innehåller de markerade förbestämda faktorerna (krävs)

dependent
Id för den varslot som innehåller den valda beroende variabeln (krävs)

<embed>

Inbädda ett annat instickspogram i det här (se kapitlet om inbäddning). Egenskaper:

component
Det registrerade namnet på komponenten att inbädda (se kapitlet om att registrera komponenter) (krävs)

as_button
Om satt till ″true″, läggs bara en tryckknapp till i det inbäddande grafiska användargränssnittet, det inbäddade grafiska användargränssnittet visas bara (i ett separat fönster) när tryckknappen klickas (valfri, förvalt värde är ″false″)

label
Bara meningsfull om as_button=″true″: Knappens beteckning (rekommenderas, förvalt värde är ″Options″)

<preview>

Kryssruta för att byta förhandsgranskningsfunktionalitet. Från version 0.6.5 av RKWard hanteras förhandsgranskningselement med **<preview>** speciellt i instickspogrammens dialogrutor (inte guider). De placeras i knappkolumnen, oberoende av exakt var de definieras i användargränssnittet. Det är ändå en bra idé att definiera dem på ett vettigt ställe i layouten, för bakåtkompatibilitet.

label
Rutans beteckning (valfri, förvalt värde är ″Preview″)

mode
Typ av förhandsgranskning. Typer som stöds är ″plot″ (se kapitlet om förhandsgranskning av diagram), ″output″ (se kapitlet om förhandsgranskning av HTML-)utmatning), ″data″ (se förhandsgranskning av data) och ″custom″ (se anpassade förhandsgranskningar). (valfri, förvalt värde är ″plot″)

placement
Förhandsgranskningens placering: ″attached″ (ansluten till huvudarbetsplatsen), ″detached″ (fristående fönster), ″docked″ (ansluten till instickspogrammets dialogruta) och ″default″ (för närvarande samma som ″docked″, men kan komma att bli möjligt att ställa in av användaren vid något tillfälle). I allmänhet rekommenderas att låta den förbli default, för bästa möjliga likformighet i användargränssnittet (valfri, förvalt värde är ″default″)

active
Om förhandsgranskningen normalt är aktiv. I allmänhet bör bara dockade förhandsgranskningar göras normalt aktiva, och till och med för dem finns en orsak att det normala värdet är inaktiva förhandsgranskningar (valfritt, förvalt värde är ″false″)

A.3.5 Logiksektion

<logic>
Det omgivande elementet för logiksektionen. Alla element nedan tillåts bara inne i elementet <logic>. Elementet <logic> tillåts bara som ett direkt underliggande objekt till <document>-elementet (som mest en gång per document), eller till <optionset>-element (som mest en gång per optionset). Dokumentets logiksektion gäller för både de grafiska användargränssnitten i <dialog> och <wizard> på samma sätt.

<external>
Skapar en ny (sträng)egenskap som är avsedd att anslutas till en egenskap utanför om instickspogrammet inbäddas. Se avsnittet om ofullständiga instickspogram. Egenskaper:

id
Den nya egenskapens id (krävs)

default
Den nya egenskapens förvalda strängvärde, dvs. värdet som används om egenskapen inte är ansluten till en egenskap utanför (valfri, förvalt värde är en tom sträng)

<i18n>
Skapar en ny (sträng)egenskap som är avsedd att tillhandahålla en beteckning som använder i18n. Egenskaper:

id
Den nya egenskapens id (krävs)

label
Beteckningen. Den kommer att översättas (krävs).

<set>
Ställ in en egenskap till ett konstant värde (om egenskapen dessutom ansluts till någon annan egenskap förblir naturligtvis inte värdet konstant). Om ett instickspogram exempelvis inbäddas, men man vill dölja vissa av dess element, kan synlighetsegenskapen för elementen ställas in till false. Användbar i synnerhet för inbäddade och inbäddande instickspogram. Observera: Om det finns flera <set>-element för en enda `id`, gäller den som definieras sist. Det kan ibland vara användbart att förlita sig på när inkluderade delar med <include> används. Egenskaper:

id
Egenskapens id som ska tilldelas (krävs)

to
Strängvärdet att tilldela egenskapen (krävs). Observera: För Booleska egenskaper som synlighet och aktivering, anges egenskapen antingen som to="true" eller to="false".

<convert>

Skapa en ny Boolesk egenskap som beror på tillståndet hos en eller flera egenskaper. Egenskaper:

id
Den nya egenskapens id (krävs)

sources
Id för egenskaperna som den här egenskapen kommer att bero på. En eller flera egenskaper kan anges, åtskilda med ";" (krävs)

mode
Metoden för konverteringen/operationen. Ett av "equals", "notequals", "range", "and", "or". Med metoden equals, är egenskapen bara sann om värdet på alla dess källor är lika med egenskapen standard (se nedan). Med metoden notequals, är egenskapen bara sann om värdet på alla dess källor skiljer sig från egenskapen standard (se nedan). Med metoden range, måste källorna vara numeriska (heltal eller reella tal). Egenskapen är bara sann om alla källor är inom intervallet angivet av egenskaperna min och max (se nedan). Med metoden and, måste källorna vara Booleska egenskaper. Egenskapen är bara sann om alla källor samtidigt är sanna. Med metoden or, måste källorna vara Booleska egenskaper. Egenskapen är bara sann om minst en av källorna är sann. (krävs)

standard
Bara meningsfull för lägena equals eller notequals: Strängvärdet att jämföra med (krävs för något av dessa lägen)

min
Bara meningsfull för läget "range": Det minimala värdet att jämföra med (valfritt, förvalt värde är det minsta flyttalet som datorn kan representera)

max
Bara meningsfull för läget "range": Det maximala värdet att jämföra med (valfritt, förvalt värde är det största flyttalet som datorn kan representera)

require_true
Om satt till "true", kommer egenskapen att krävas, och anses bara giltig om dess tillstånd är "true" eller "on". Sålunda blockerar egenskapen knappen **Verkställ** om den är falsk (valfri, förvalt värde "false").

> OBSERVERA
> Om det används, se till att användaren enkelt kan detektera vad som är fel, genom att exempelvis visa en förklarande <text>.

<switch>

Skapa en ny egenskap som vidarebefordrar till olika målegenskaper (eller förbestämda strängar) baserat på villkorsegenskapens värde. Det gör det möjligt att skapa logik som liknar konstruktionerna `if()` eller `switch()`. Egenskaper:

id
Den nya egenskapens id (krävs)

villkor
Villkorsegenskapens id (krävs)

Underliggande element:

<true>
Om villkorsegenskapen är Boolesk, kan två de underliggande elementen <true> och <false> anges (och bara dessa) (Krävs om <false> också anges).

<false>
Om villkorsegenskapen är Boolesk, kan två de underliggande elementen <true> och <false> anges (och bara dessa) (Krävs om <true> också anges).

<case>
Om villkorsegenskapen inte är Boolesk, kan ett godtyckligt antal element av typen <case> anges, ett för varje värde på villkorsegenskapen som ska matchas (åtminstone ett sådant element krävs om villkorsegenskapen inte är Boolesk)

<default>
Om villkorsegenskapen inte är Boolesk, gör det valfria elementet <default> det möjligt att ange beteendet om inget av elementen av typen <case> matchar villkorsegenskapens värde (valfri, tillåts bara en gång i kombination med ett eller flera element av typen <case>).

Underliggande elementen <true>, <false>, <case>, och <default> har följande egenskaper:

standard
Bara för element av typen <case>. Värdet som villkorsegenskapen ska matchas mot (krävs, sträng).

fixed_value
En bestämd sträng som ska anges som värdet på egenskapen <switch>, om om det aktuella villkoret matchar (krävs om dynamic_value inte anges).

dynamic_value
Målegenskapens *id* som anges som värdet på egenskapen <switch>, om det aktuella villkoret matchar (krävs, om fixed_value inte anges).

<connect>
Ansluter två egenskaper. Klientegenskapen ändras så fort den styrande egenskapen ändras (men inte det omvända). Egenskaper:

client
Klientegenskapens id, dvs. egenskapen som justeras (krävs)

governor
Den styrande egenskapens id, dvs. egenskapen som justerar klientegenskapen. Det kan inkludera en modifierare (krävs)

reconcile
Om ″true″, justerar klientegenskapen den styrande egenskapen i anslutningen på ett sådant sätt att den styrande egenskapen bara accepterar värden som också är acceptabla av klienten (antag t.ex. att den styrande egenskapen är en numerisk egenskap med minimalt värde ″0″, och klienten är en numerisk egenskap med minimalt värde ″100″. Minimum för båda egenskaperna justeras till 100 om reconcile=″true″). I allmänhet fungerar det bara för egenskaper med samma grundtyp (valfri, förvalt värde ″false″)

<dependency_check>
Skapar en Boolesk egenskap som är sann om angivna beroenden uppfylls, och annars falsk. Elementets XML-syntax är likadan som för elementet **<dependencies>**, beskriven i `.pluginmap`-referensen. Från RKWard 0.6.1 tas bara hänsyn till versionerna för RKWard och R, inte beroenden av paket eller pluginmaps.

<script>
Definiera skriptkoden för att kontrollera användargränssnittets logik. Se avsnittet om skriptbaserad logik för det grafiskt användargränssnittet för detaljerad information. Skriptkoden att köra kan antingen anges med egenskapen *″file″* eller som en (kommenterad) text i elementet. Elementet **<script>** tillåts inte i sektionen **<logic>** i ett optionset. Egenskaper:

file
Skriptfilens filnamn (krävs).

A.4 Egenskaper för element i instickspogram

Alla layoutelement och alla aktiva element innehåller följande egenskaper, som kan kommas åt via"elementnamnets_id.egenskapens_namn":

visible
Om elementet i det grafiska användargränssnittet är synligt eller inte (Boolean)

enabled
Om elementet i det grafiska användargränssnittet är aktiverat eller inte (Boolean)

required
Om elementet i det grafiska användargränssnittet krävs (innehålla en giltig inställning) eller inte. Observera att alla element som är inaktiverade eller dolda också implicit inte krävs (Boolean).

Förutom det här, har vissa element ytterligare egenskaper som man kan ansluta till. De flesta aktiva element har också en "förvald" egenskap vars värde returneras av anrop till `getBoolean`, `getString`, `getList("...")`, om ingen specifik egenskap, som beskrivs nedan, namnges.

<text>
Förvald egenskap är text

text
Texten som visas (text)

<varselector>
Ingen förvald egenskap

selected
Objekten som för närvarande är markerade. Troligen vill man inte använda det. Används internt (RObject)

root
Rotobjekt eller överliggande objekt för de objekt som erbjuds för urval (RObject)

<varslot>
Förvald egenskap är "available"

available
Alla objekt som finns i denna varslot (RObject)

selected
De objekt av alla i en varslot som för närvarande är markerade. Troligen vill man inte använda det. Används internt (RObject)

source
En kopia av objekten som är markerade i motsvarande varselector. Troligen vill man inte använda det. Används internt (RObject)

<valueselector>
Förvald egenskap är "selected"

selected
Strängarna som för närvarande är markerade. Modifierare "labeled" för att hämta motsvarande beteckningar. I en <valueselector> vill man troligtvis inte använda det direkt (bara i en <select>) (läs och skriv, stränglista).

available
Listan med strängvärden att välja mellan (läs/skriv StringList)

labels
Beteckningar att visa för strängvärdena (läs/skriv StringList)

<valueslot>
Samma som <varslot>, men egenskaperna är stränglistor istället för RObjects.

<radio>
Förvald egenskap är ″string″

string
Värdet på alternativet som för närvarande är valt (sträng)

number
Nummer för alternativet som för närvarande är markerat (alternativ numreras uppifrån och ner med början på 0) (heltal)

<dropdown>
Samma som <radio>

<select>
Samma som <valueselector>

<option>
Ingen förvald egenskap. Den *enda* egenskapen är ″enabled″, och den är för närvarande inte tillgänglig för alternativ inne i <select> eller <valueselector>. Egenskaperna ″visible″ eller ″required″ finns inte för <option>.

enabled
Om det här enskilda alternativet ska aktiveras eller inaktiveras. I de flesta fall aktiverar eller inaktiverar man alla alternativknapparna, <radio<, eller kombinationsrutan, <dropdown<, istället. Men det kan användas för att dynamiskt ställa in aktiveringstillståndet för ett enskilt alternativ inne i en <radio< eller <dropdown< (Boolean).

<checkbox>
Förvald egenskap är ″state.labeled″, vilket betyder att värdena angivna av egenskaperna `value` och `value_unchecked` returneras, *inte* kryssrutans visade beteckning.

state
Kryssrutans tillstånd (på eller av). Observera att användbara modifierare av egenskapen (som för alla Booleska egenskaper) är ″not″ och ″numeric″ (se egenskapers typer). Dock är det ofta mest användbart att ansluta till egenskapen utan modifierare, dvs. ″*checkbox_id*.state″, vilket returnerar kryssrutans tillstånd på ett format användbart för användning i en villkorssats (0 eller 1) (Boolean).

<frame>
Förvald egenskap är ″checked″, om och endast om ramen är markeringsbar. För ramar som inte är markeringsbara finns ingen förvald egenskap.

checked
Bara tillgänglig för markeringsbara ramar: kryssrutans tillstånd (på eller av). Observera att användbara modifierare av egenskapen (som för alla Booleska egenskaper) är ″not″ och ″numeric″ (se egenskapers typer) (Boolean).

<input>
Förvald egenskap är ″text″

text
Aktuell text i inmatningsfältet (sträng)

<matrix>
Förvald egenskap är ″cbind″.

rows
Antal rader i matrisen (heltal). Om matrisen tillåter användaren att lägga till och ta bort rader, ska egenskapen behandlas som skrivskyddad. Annars ändras matrisens storlek om den ändras.

columns
Antal kolumner i matrisen (heltal). Om matrisen tillåter användaren att lägga till och ta bort kolumner, ska egenskapen behandlas som skrivskyddad. Annars ändras matrisens storlek om den ändras.

tsv
Data i matrisen på tsv-format (sträng, läs-skriv). Observera att jämfört med den vanliga tsv-layouten, är *kolumner*, inte rader, åtskilda med nyradstecken, och celler inom en kolumn är åtskilda av tabulatortecken.

0,1,2...
Data från en enskild kolumn (0 för vänstra kolumnen). `getValue()` eller `getString()` returnerar den som en enda sträng, åtskild av ″\n″. Dock är det rekommenderade sättet att hämta den att använda `getList()`, som returnerar kolumnen som ett strängfält.

row.0,row.1,row.2...
Data från en enskild rad (0 för översta raden). `getValue()` eller `getString()` returnerar den som en enda sträng, åtskild av ″\n″. Dock är det rekommenderade sättet att hämta den att använda `getList()`, som returnerar raden som ett strängfält.

cbind
Data på ett format lämpligt för att klistra in i R, omgivet av en cbind-sats (sträng, skrivskyddad).

<optionset>

Ingen förvald egenskap.

row_count
Antal objekt i optionset (heltal). Skrivskyddad.

current_row
Objekt som för närvarande är aktivt i ett optionset (heltal). -1 för inget aktivt objekt. Läs- och skrivbar.

optioncolumn_ids
För varje <optioncolumn> som definieras, skapas en ″string list″-egenskap med angivet id.

<browser>

Förvald egenskap är ″selection″

selection
Aktuell text (markerat filnamn) i bläddraren (sträng)

<saveobject>

Förvald egenskap är ″selection″

selection
Det markerade objektets fullständiga namn (sträng, skrivskyddad: för att tilldela det programmatiskt, använd ″parent″ och ″objectname″).

parent
Det markerade objektets överliggande objekt. Det är alltid ett befintligt R-objekt av en typ som kan innehålla andra objekt (t.ex. en lista eller data.frame). När det är satt till en tom sträng eller ett ogiltigt objekt, antas ″.GlobalEnv″ (RObject)

objectname
Det markerade objektets grundnamn, dvs. strängen som matas in av användaren (ändrat till ett giltigt R-namn, vid behov) (sträng)

active
Endast för markeringsbara saveobjekts: Om objektet är markerad eller aktiverat. Alltid sant för ej markeringsbara saveobjects (Boolean)

<spinbox>

Förvald egenskap är antingen ″int″ eller ″real.formatted″ beroende på nummerrutans läge

int
Heltalsvärde som nummerrutan innehåller, eller närmaste heltal för reellt läge (heltal)

real
Reellt värde som nummerrutan innehåller (och heltal, om i heltalsläge) (reell)

<formula>

Förvald egenskap är ″model″

model
Den aktuella modellsträngen (sträng)

table
Den data.frame som innehåller nödvändiga variabler. Om bara variabler från en data.frame används, returneras namnet på denna data.frame. Annars konstrueras en ny data.frame efter behov (sträng)

labels
Om variabler från flera data.frames är inblandade, kan deras namn bli omvandlade (exempelvis om båda data.frames innehåller en variabel som heter ″x″). Returnerar en lista med de omvandlade namnen som index och den beskrivande beteckningen som värde (sträng)

fixed_factors
De förbestämda faktorerna. Troligen vill man inte använda det. Används internt (RObject)

dependent
Den eller de beroende variablerna. Troligen vill man inte använda det. Används internt (RObject)

<embed>

Ingen förvald egenskap

code
Koden som skapas av det inbäddade instickspogrammet (code)

<preview>

Förvald egenskap är ″state″

state
Om förhandsgranskningsrutan är markerad (inte nödvändigtvis om förhandsgranskningen redan har visats) (Boolean)

<convert>

Elementet (som används i sektionen <logic>) är speciell på det sättet att den tekniskt sett *är* en egenskap, istället för att bara innehålla en eller flera egenskaper. Den är av typen Boolean. Observera att användbara modifierare av egenskapen (som för alla Booleska egenskaper) är ″not″ och ″numeric″ (se egenskapers typer).

<switch>

Elementet (som används i sektionen <logic>) är speciell på det sättet att den tekniskt sett *är* en (sträng-) egenskap, istället för att bara innehålla en eller flera egenskaper. Den gör det möjligt att byta mellan olika målegenskaper beroende på värdet av en villkorsegenskap, eller för att ändra avbildning av villkorsegenskapens värden. Alla modifierare som anges skickas till målegenskaperna. Sålunda kan t.ex. modifieraren ″shortname″ också användas

i omkopplaren, om alla målegenskaper är RObject-egenskaper. Om målegenskaperna har olika typer kan dock användning av modifierare orsaka fel. För förbestämda värden, `fixed_value`, bortses tyst från eventuella modifierare. Observera att målegenskaper, vid åtkomst via en omkopplare, alltid är skrivskyddade.

A.5 Inbäddningsbara insticksprogram som levereras med den officiella utgåvan av RKWard

Ett antal inbäddningsbara insticksprogram levereras med RKWard, och kan användas i dina egna insticksprogram. Detaljerad dokumentation är för närvarande bara tillgänglig i insticksprogrammens källkod eller hjälpfiler. Här är dock en lista för att ge en snabb översikt av vad som är tillgängligt:

ID	**Pluginmap**	**Beskrivning**	**Exempel på användning**
rkward::plot_options	embedded.pluginmap	Tillhandahåller ett stort antal alternativ för diagram. De flesta insticksprogram för diagramritning utnyttjar det.	Plots->Barplot, de flesta andra insticksprogram för diagramritning
rkward::color_chooser	embedded.pluginmap	Mycket enkelt insticksprogram för att ange en färg. Nuvarande implementering tillhandahåller en lista med färgnamn. Framtida implementeringar kan komma att tillhandahålla mer utarbetad färghämtning.	Plots->Histogram
rkward::plot_stepfun_options	embedded.pluginmap	Diagramalternativ för stegfunktion	Plots->ECDF plot
rkward::histogram_options	embedded.pluginmap	Histogramalternativ (diagram)	Plots->Histogram
rkward::barplot_embed	embedded.pluginmap	Alternativ för stapeldiagram	Plots->Barplot
rkward::one_var_tabulation	embedded.pluginmap	Tillhandahåller tabelluppställning för en ensam variabel.	Plots->Barplot
rkward::limit_vector_length	embedded.pluginmap	Begränsa en vektors längd (till de n största eller minsta elementen).	Plots->Barplot

rkward::level_select	embedded.pluginmap	Tillhandahåller en <valueselector> ifylld med nivåerna (eller unika värdena) i en vektor.	Data->Recode Categorical data
rkward::multi_input	embedded.pluginmap	Kombinerar nummerruta, inmatningsruta och alternativknapp för att tillhandahålla inmatning av tecken, numerisk och logisk data.	Data->Recode Categorical data

Tabell A.1: Inbäddningsbara standardinstickspogram

A.6 Element att använda i `.pluginmap`-filer

<document>

Måste finnas i varje `.pluginmap`-fil som rotnod (exakt en gång). Egenskaper:

base_prefix

Filnamn angivna i `.pluginmap`-filen antas vara relative till katalogen där `.pluginmap`-filen finns plus det prefix som anges här. Särskilt användbar om alla komponenter är placerade i en enda underkatalog.

namespace

En namnrymd för komponent-identifierare. När komponenter slås upp för inbäddning, går det att hämta komponenterna via strängen ″namnrymd::komponent_id″. För närvarande satt till ″rkward″.

id

En valfri identifierarsträng för din `.pluginmap`. Att ange den låter en tredjepartsutvecklare hänvisa till den och läsa in din `.pluginmap` från sin egen (se kapitlet om att hantera beroenden).

priority

Ett av *˜hidden˜*, *˜low˜*, *˜medium˜* eller *˜high˜*. En `.pluginmap` med prioritet ″medium″ eller ″high″ aktiveras automatiskt när RKWard först hittar dem. Använd *priority=˜hidden˜* för en `.pluginmap` som inte är avsedd att aktiveras, katalog (bara avsedd för inkludering). I den nuvarande implementeringen döljer det dock inte i själva verket en `.pluginmap`. (Valfri, med förvalt värde ″medium″).

<dependencies>

Elementet, som anger beroenden, tillåts som ett direkt underliggande objekt till elementet <document> (en gång), och som ett underliggande objekt till elementen <component> (en gång för varje <component>). Anger de beroenden som måste uppfyllas för att kunna använda instickspogrammen. Se kapitlet om beroenden för en översikt. Egenskaper:

rkward_min_version, rkward_max_version

Minimal och maximal tillåten version av RKWard. Versionsspecifikationer får innehålla icke-numeriska suffix, såsom ″0.5.7z-devel1″. Om ett angivet beroende inte uppfylls, kommer instickspogrammen det gäller att *ignoreras*. Mer information. Valfri. Om ej specificerad krävs ingen minimal eller maximal version av RKWard.

R_min_version, R_max_version
Minimal och maximal tillåten version av R. Versionsspecifikationer får *inte* innehålla icke-numeriska suffix, såsom ″0.5.7z-devel1″. Beroendet av R-version visas på insticksprogrammets hjälpsidor, men har ingen direkt effekt till och med RKWard 0.6.1. Mer information. Valfri, om ej specificerad krävs ingen minimal eller maximal version av R.

Underliggande element:

<package>
Lägger till ett beroende på ett specifikt R-paket. Egenskaper:
name
Paketnamn (krävs).
min_version, max_version
Minimal eller maximal tillåten version (valfri)
repository
Arkiv där paketet kan hittas. Valfritt, men rekommenderas starkt. Om paketet inte är tillgängligt på CRAN.

<pluginmap>
Lägger till ett beroende på en specifik RKWard `.pluginmap`. Egenskaper:
name
Id-sträng för den `.pluginmap` som krävs (krävs).
min_version, max_version
Minimal eller maximal tillåten version (valfri)
url
Webbadress där `.pluginmap` kan hittas. Krävs.

<about>
Får finnas exakt en gång som ett direkt underliggande objekt till elementet <document>. Innehåller metainformation om `.pluginmap` (eller insticksprogram). Se kapitlet om 'om'-information för en översikt. Egenskaper:

name
Synligt användarnamn. Valfritt. Måste inte vara samma som ″id″.
version
Versionsnummer. Valfritt. Formatet är inte begränsat, men för att vara på den säkra sidan, följ vanlig versionsnumrering såsom ″x.y.z″.
releasedate
Specifikation av utgivningsdatum. Valfri på formatet ″ÅÅÅÅ-MM-DD″.
shortinfo
En *kort* beskrivning av insticksprogrammet eller `.pluginmap`. Valfri.
url
Webbadress där mer information kan finnas. Valfri, men rekommenderad.
copyright
Specifikation av copyright, t.ex. ″2012-2013 av Anna Svensson″. Valfri, men rekommenderad.
licence
Specifikation av licens, t.ex. ″GPL″ eller ″BSD″. Försäkra att filerna åtföljs av en fullständig kopia av relevant licens. Valfri, men rekommenderad.
category
Insticksprogrammens kategori, t.ex. ″Item response theory″. Från RKWard 0.6.1 är inga kategorier fördefinierade. Valfri.

Underliggande element:

<author>
Lägger till information om en upphovsman. Egenskaper:

name, given, family
Ange antingen hela namnet som *name*, eller ange både *given* (förnamn) och *family* (efternamn) separat.

role
Upphovsmannens rollbeskrivning (valfri).

email
E-postadress där upphovsmannen kan kontaktas. Krävs. Kan sättas till e-postlistan rkward-devel, om du prenumererar, och ditt insticksprogram är avsett att inkluderas i den officiella utgåvan av RKWard.

url
Webbadress med mer information om upphovsmannen, t.ex. hemsida (valfri).

<components>
Måste finnas exakt en gång som ett direkt underliggande objekt till elementet <document>. Innehåller de individuella elementen <component> som beskrivs nedan. Inga egenskaper.

<component>
En eller flera element av typen <component> ska anges som direkt underliggande objekt till elementen <components> (och bara där). Registrerar en komponent eller ett insticksprogram i rkward. Egenskaper:

type
För framtida utökningar: Typ av komponent eller insticksprogram. För tillfället alltid satt till ″standard″ (den enda typ som för närvarande stöds).

id
Det id som kan användas för att hämta komponenten (för att placera den i menyn, se nedan, eller för inbäddning). Se namnrymden <document> ovan.

file
Krävs åtminstone för komponenter av type=″standard″: Filnamnet på XML-filen som beskriver det grafiska användargränssnittet.

label
Komponentens rubrik när den placeras i menyhierarkin.

<attribute>
Definierar en egenskap för en komponent. Hittills bara meningsfull för importinsticksprogram. Bara tillåten som ett direkt underliggande objekt till <component>. Egenskaper:

id
Egenskapens id

value
Egenskapens värde

labels
Rubrik som hör ihop med egenskapen

<hierarchy>
Måste finnas exakt en gång som ett direkt underliggande objekt till elementet <document>. Beskriver var komponenten som deklareras ovan ska placeras i menyhierarkin. Accepterar bara <menu>-element som direkt underliggande objekt. Inga egenskaper.

<menu>
Ett eller flera menyelement, <menu>, ska anges som direkt underliggande objekt till elementet <hierarchy>. Deklarerar en ny (under-)meny. Om en meny med angivet id (se nedan) redan finns, sammanfogas de två menyerna. Elementet <menu> tillåts antingen som ett direkt underliggande objekt till elementet <hierarchy> (toppnivåmeny) eller som ett direkt underliggande objekt till någon annan <menu> (undermeny). Omvänt, accepterar elementet <menu> andra <menu> eller <entry> som underliggande objekt. Egenskaper:

id
: En identifierarsträng för menyn. Användbar när menydefinitioner läses från flera `.pluginmap`-filer, för att försäkra att instickspogram kan placeras i samma meny eller menyer. Vissa menyidentifierare såsom ″file″ refererar till fördefinierade menyer (i detta fall, menyn ″Arkiv″). Var noga med att kontrollera befintliga `.pluginmap`-filer för att använda överensstämmande identifierare.

label
: En rubrik för menyn.

group
: Gör det möjligt att bestämma menyalternativens ordning. Se gruppering av menyalternativ. Valfri.

<entry>

En menypost, dvs. ett menyalternativ för att starta ett instickspogram. Kan bara användas som ett direkt underliggande objekt till elementet <menu>, accepterar inga underliggande element. Egenskaper:

component
: Id för komponenten som ska anropas när menyalternativet aktiveras.

group
: Gör det möjligt att bestämma menyalternativens ordning. Se gruppering av menyalternativ. Valfri.

<group>

Deklarerar en grupp av objekt i menyn. Se gruppering av menyalternativ. Egenskaper:

id
: Namnet på den här gruppen.

separated
: Valfritt. Om det sätts till ″true″ är objektet i gruppen visuellt avskilt från omgivande objekt.

group
: Namnet på gruppen som den här gruppen ska läggas till sist i (valfri).

<context>

Deklarerar posterna i ett context. tillåts bara som direkt underliggande objekt till noden <document>. Accepterar bara <menu>-taggar som direkt underliggande objekt. Egenskaper:

id
: Sammanhangets id. Hittills är bara två sammanhang implementerade: ″xll″ och ″import″.

<require>

Inkludera en annan `.pluginmap`-fil. Denna `.pluginmap`-fil läses bara in en gång, även om den krävs från flera andra filer med <require>. Det viktigaste användarfallet är att inkludera en pluginmap-fil som deklarerar några komponenter, som inbäddas av komponenter i denna `.pluginmap`. Elementen <require> tillåts bara som direkt underliggande objekt till noden <document>. Egenskaper:

file
: Filnamnet för den `.pluginmap` som ska inkluderas. Det betraktas som relativt till katalogen för den aktuella `.pluginmap`-filen plus base_prefix (se ovan för elementet <document>). Om den relativa sökvägen till den `.pluginmap` som ska inkluderas inte är känd, använd istället egenskapen *`map`* för att referera till den enligt id.

map

För att inkludera en `.pluginmap`-fil från ett annat paket (eller en RKWard `.pluginmap` från din externa `.pluginmap`), som kan refereras till med *`namespacename::id`*, som angivet i elementet <document> som krävs för denna `.pluginmap`. Inkluderingen misslyckas om ingen `.pluginmap` med detta id är känd, t.ex. inte är installerad på användarens system). Metoden bör bara användas för att inkludera en `.pluginmap` utanför ditt paket. För de som är inne i ditt paket, är en relativ sökväg snabbare och tillförlitligare (egenskapen *`file`*).

A.7 Element att använda i .rkh-filer (hjälp)

<document>

Måste finnas i varje `.xml`-fil som rotnod (exakt en gång). Inga egenskaper.

<title>

Hjälpsidans titel. Det tolkas *inte* för ett insticksprograms hjälpsidor (de tar titeln från insticksprogrammet självt), bara för fristående sidor. Inga egenskaper. Texten som finns i taggen <title> blir hjälpsidans rubrik. Kan bara definieras en gång, som ett direkt underliggande objekt till noden <document>.

<summary>

En kort sammanfattning av hjälpsidan (eller vad insticksprogrammet används för). Den visas alltid överst på hjälpsidan. Inga egenskaper. Texten som är innehållet i taggen <summary> visas. Rekommenderas, men krävs inte. Kan bara definieras en gång, som ett direkt underliggande objekt till noden <document>.

<usage>

En något mer detaljerad summering av användningen. Den visas alltid direkt efter <summary>. Inga egenskaper. Texten som är innehållet i taggen <usage> visas. Rekommenderas på hjälpsidor för insticksprogram, men krävs inte. Kan bara definieras en gång, som ett direkt underliggande objekt till noden <document>.

<section>

En sektion för allmänna ändamål. Kan användas hur många gånger som helst som ett direkt underliggande objekt till noden <document>. Sektionerna visas i den ordning de definieras, men alla *efter* sektionen <usage> och *innan* sektionen <settings>. Texten som är innehållet i taggen <section> visas.

id

En identifierare som behövs för att gå till avsnittet från navigeringsraden (eller en länk). Måste vara unik i filen. Krävs, inga förval.

title

Avsnittets titel (rubrik). Krävs, inget förvalt värde.

short_title

En kort titel lämplig att visa i navigeringsraden. Valfri, förvalt värde är den fullständiga titeln.

<settings>

Definierar sektionen som innehåller referenser till de olika inställningarna i det grafiska användargränssnittet. Bara meningsfull och använd för hjälpsidor relaterade till insticksprogram. Använd som ett direkt underliggande objekt till <document>. Får bara innehålla elementen <setting> och <caption> som direkt underliggande objekt. Inga egenskaper.

<setting>

Förklarar en enskild inställning i det grafiska användargränssnittet. Tillåts bara som ett direkt underliggande objekt till elementet <settings>. Texten som elementet innehåller elementet visas.

id

Inställningens id i instickspogrammets `.xml`. Krävs, saknar förvalt värde.

title

En valfri titel för inställningen. Om den utelämnas (att utelämna den rekommenderas i de flesta fall), tas titeln från instickspogrammets `.xml`.

<caption>

En rubrik för att gruppera flera inställningar visuellt. Får bara användas som ett direkt underliggande objekt till elementet <settings>.

id

Det motsvarande elementets id (oftast en <frame>, <page> eller <tab>) i instickspogrammets `.xml`.

title

En valfri titel för rubriken. Om den utelämnas (att utelämna den rekommenderas i de flesta fall), tas titeln från instickspogrammets `.xml`.

<related>

Definierar ett avsnitt som innehåller länkar till ytterligare relaterad information. Visas alltid efter avsnittet <settings<. Inga egenskaper. Texten som finns inom taggen <related> visas. Ofta innehåller det en lista med HTML-stil. Rekommenderas för instickspogrograms hjälpsidor, men krävs inte. Kan bara definieras en gång direkt under noden <document>.

<technical>

Definierar ett avsnitt som innehåller teknisk information som inte är relevant för slutanvändare (såsom instickspogrammets interna struktur). Visas alltid sist på en hjälpsida. Inga egenskaper. Texten som finns inom taggen <related> visas. Krävs inte, och rekommenderas inte för de flesta instickspogrograms hjälpsidor. Kan bara definieras en gång direkt under noden <document>.

<länk>

En länk. Kan användas i vilken som helst av sektionerna som beskrivs ovan.

href

Målwebbadressen. Observera att flera RKWard-specifika webbadresser är tillgängliga. Se avsnittet om att skriva hjälpsidor för detaljerad information.

<label>

Infogar värdet av en rubrik i användargränssnittet. Kan användas i vilken som helst av sektionerna som användas ovan.

id

Elementets id i instickspogrammet, som egenskapen *`label`* ska kopieras från.

<olika HTML-taggar>

De flesta HTML-grundtaggar tillåts inne i sektionerna. Håll dock manuell formatering så liten som möjlig.

A.8 Funktioner tillgängliga för att skriva skriptlogik för grafiska användargränssnitt

Klass ″Component″

Klass som representerar en enskild komponent eller komponentegenskap. Den viktigaste instansen av klassen är variabeln ″gui″ som är fördefinierad som den aktuella komponentens rotegenskapen. Följande metoder är tillgängliga för instanser av klassen ″Component″:

absoluteId(bas_id)

Returnerar absolut id för *bas_id*, eller om bas_id utelämnas, komponentens identifierare.

getValue(id)

Avråds från. Använd `getString()`, `getBoolean()` eller `getList()` istället. Returnerar värdet av det givna underliggande objektets egenskap. Returnerar värdet av egenskapen om ID utelämnas.

getString(id)

Returnerar värdet av det givna underliggande objektets egenskap som en sträng. Returnerar värdet av egenskapen om ID utelämnas.

getBoolean(id)

Returnerar värdet av det givna underliggande objektets egenskap som en Boolean (om möjligt). Returnerar värdet av egenskapen om ID utelämnas.

getList(id)

Returnerar värdet av det givna underliggande objektets egenskap som ett fält av strängar (om möjligt). Returnerar värdet av egenskapen om ID utelämnas.

setValue(id, värde)

Tilldela värdet *värde*till det givna underliggande objektet.

getChild(id)

Returnerar en instans av det underliggande egenskapen med givet *id*.

addChangeCommand(id, kommando)

Kör *kommando* så snart det underliggande objektets egenskap angiven av *id* ändras.

Klass ″RObject″

Klass som representerar ett enskilt R-objekt. En instans av klassen kan erhållas genom att använda **makeRObject(objektnamn)**. Följande metoder är tillgängliga för instanser av klassen ″RObject″.

> VARNING
> Om några kommandon fortfarande väntar i bakgrundsprogrammet, kan informationen som levereras av de här metoderna vara inaktuellt när instickspogrammets kod väl körs. Lita *inte* på det för kritiska operationer (och riskera dataförlust).

getName()

Returnerar objektets absoluta namn.

exists()

Returnera om objektet finns. Du bör i allmänhet kontrollera det innan någon av metoderna som listas nedan används.

dimensions()

Returnerar ett fält med dimensioner (liknar **dim()** i R).

classes()

Returnerar ett fält med klasser (liknar **class()** i R).

isClass(klass)

Returnerar true, om objektet är av klassen *klass*.

isDataFrame()
Returnerar true, om objektet är en dataram.

isMatrix()
Returnerar true, om objektet är en matris.

isList()
Returnerar true, om objektet är en lista.

isFunction()
Returnerar true, om objektet är en funktion.

isEnvironment()
Returnerar true, om objektet är en omgivning.

isDataNumeric()
Returnerar true, om objektet är en vector med numerisk data.

isDataFactor()
Returnerar true, om objektet är en vektor med faktordata.

isDataCharacter()
Returnerar true, om objektet är en vektor med teckendata.

isDataLogical()
Returnerar true, om objektet är en vektor med logisk data.

parent()
Returnerar en instans av ″RObject″ som representerar det överliggande objektet till det här objektet.

child(namn)
Returnerar en instans av ″RObject″ som representerar det underliggande objektet *namn* till det här objektet.

Klass ″RObjectArray″

Ett fält av RObject-instanser. En instans av klassen kan erhållas genom att använda **makeRObjectArray(objektnamn)**. Det är särskilt användbart när det handlar om varslots, vilka möjliggör att flera objekt markeras.

Funktionen include()

include(filnamn) kan användas för att inkludera en separat JS-fil.

Funktionen doRCommand()

doRCommand(kommando, återanrop) kan användas för att begära information från R. Läs avsnittet om att begära information från R inne i ett instickspogram för detaljerad information och förbehåll.

Bilaga B

Felsökning under utveckling av insticksprogram

Så du har läst all dokumentation, har gjort allt rätt, och kan ändå inte få det att fungera? Var inte orolig, vi löser det. Det första att göra är att aktivera fönstret **RKWard avlusningsmeddelanden** (tillgängligt i menyn **Fönster**, eller genom att klicka på en av verktygsraderna), och därefter starta ditt insticksprogram igen. Som en allmän tumregel ska du inte se någon utmatning i meddelandefönstret när ditt insticksprogram, eller när som helst annars. Om det visas något, har det troligen att göra med ditt insticksprogram. Se om det hjälper dig.

Om allting verkar bra i terminalen, försök att öka avlusningsnivån (från kommandoraden genom att använda **rkward --debug-level 3** eller genom att ställa in avlusningsnivån till 3 i **Inställningar** → **Anpassa RKWard** → **Avlusning**. Alla meddelanden som visas vid en högre avlusningsnivå anger inte ett problem, men chansen finns att ditt problem dyker upp någonstans i meddelandena.

Om du ändå inte kan räkna ut vad som är fel, misströsta inte. Vi är medvetna om att det är komplicerat, och trots allt är det möjligt att du har stött på ett fel i RKWard, och RKWard behöver fixas. Skriv bara till utvecklarnas e-postlista och berätta för oss om problemet. Vi hjälper gärna till.

Till sist, även om du har tagit reda på hur att göra det själv, men märkt att dokumentationen inte är till stor hjälp, eller till och med felaktig i något avseende, tala också om det för oss på e-postlistan så att vi kan rätta eller förbättra dokumentationen.

Bilaga C

Licens

Översättning Stefan Asserhäll stefan.asserhall@bredband.net

Den här dokumentationen licensieras under villkoren i GNU Free Documentation License.