# Introducció a l'escriptura de connectors per al RKWard

**Thomas Friedrichsmeier**
**Meik Michalke**
**Traductor: Josep M. Ferrer**

Introducció a l'escriptura de connectors per al RKWard

# Índex

# Índex de taules

**Resum**

Aquesta és una guia a l'escriptura de connectors per al RKWard.

# Capítol 1

# Introducció

Aquest document descriu com escriure els vostres propis connectors. La documentació ha crescut força amb el temps. No deixeu que això us espanti. Recomanem llegir els quatre passos bàsics (en termes generals, a continuació), per a obtenir una idea bàsica de com funcionen les coses. Després d'això, és possible que vulgueu fullejar la taula de continguts per a veure quins temes avançats podrien ser rellevants per a vós.

Per a preguntes i comentaris, escriviu a la llista de correu de desenvolupament del RKWard.

*No cal que ho llegiu per a utilitzar el RKWard.* Aquest document tracta sobre l'ampliació del RKWard. Està dirigit a usuaris avançats, o persones disposades a ajudar a millorar el RKWard.

Escriure un connector estàndard és bàsicament un procés de quatre passos:

- Col·locar una acció nova a la jerarquia del menú
- Descriure l'aparença i el comportament de la IGU del connector
- Definir com s'ha de generar el codi R a partir de la configuració que l'usuari fa a la IGU
- Afegir una pàgina d'ajuda al vostre connector

Aquests es tractaran per ordre.

Es poden utilitzar alguns conceptes avançats en aquests quatre passos, però es tracten en capítols separats, per a mantenir les coses senzilles:

- Lògica de la IGU
- Incrustar connectors en connectors
- Conceptes útils per a crear moltes sèries de connectors similars

A més, cap dels capítols mostra totes les opcions, sinó només els conceptes bàsics. Es proporciona per separat una referència completa d'opcions.

# Capítol 2

# Preliminars: Què són els connectors en el RKWard? Com funcionen?

Per descomptat, la primera pregunta que us podríeu fer és: quines parts de la funcionalitat del RKWard s'han realitzat utilitzant connectors? O: què poden fer els connectors?

Una manera de respondre-ho és: desseleccionar tots els fitxers del `.pluginmap` de l'**Arranjament** → **Configura el RKWard** → **Connectors**, i veure què falta. Una resposta una mica més útil: la majoria de les funcions estadístiques reals accessibles a través de la IGU es realitzen utilitzant connectors. També podeu crear IGU bastant flexibles per a tota mena d'operacions utilitzant connectors.

El paradigma bàsic darrere dels connectors del RKWard és el que us ensenyarem en aquest document: un fitxer XML descriu com es veu la IGU. S'utilitza un fitxer JavaScript addicional per a generar la sintaxi de l'R des de la configuració de la IGU. És a dir, els connectors no han de realitzar cap càlcul estadístic. Els connectors generen la sintaxi de l'R necessària per a executar aquests càlculs. La sintaxi de l'R s'envia al dorsal de l'R per a l'avaluació, i normalment es mostra un resultat a la finestra de sortida.

Llegiu en els capítols següents per a veure com es fa això.

# Capítol 3

# Creació d'entrades del menú

Quan creeu un connector nou, haureu de dir-li-ho al RKWard. Per tant, el primer que cal fer és escriure un fitxer `.pluginmap` (o modificar-ne un d'existent). El format del `.pluginmap` és XML. Us ensenyaré un exemple (també, per descomptat, assegureu-vos que el RKWard està configurat per a carregar el `.pluginmap`: **Arranjament** → **Configura el RKWard** → **Connectors**):

> SUGGERIMENT
> Després de llegir aquest capítol, mireu també el paquet rkwarddev. Proporciona algunes funcions de l'R per a crear la majoria de les etiquetes XML del RKWard.

```
<!DOCTYPE rkpluginmap>
```

El «doctype» no s'interpreta realment, però de totes maneres es defineix com a *"rkpluginmap"*.

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

Es pot utilitzar l'atribut *base_prefix*, si tots els connectors resideixen en un directori comú. Bàsicament, aleshores podeu ometre aquest directori dels noms de fitxers especificats a continuació. És segur deixar això com a *""*.

Com podeu veure a continuació, tots els connectors tenen un identificador únic, *id*. El *namespace* és una manera d'organitzar aquests ID, i fer que sigui menys probable crear accidentalment un identificador duplicat. Internament, s'anteposa l'espai de noms i després un «::» a tots els identificadors que especifiqueu en aquest `.pluginmap`. En general, si teniu la intenció de distribuir els connectors en un paquet R, és una bona idea utilitzar el nom del paquet com a paràmetre *namespace*. Els connectors enviats amb la distribució oficial del RKWard tenen el *namespace="rkward"*.

L'atribut *id* és opcional, però especificar un identificador per al vostre `.pluginmap` fa possible que altres persones facin que el seu `.pluginmap` carreguin automàticament el vostre `.pluginmap` (vegeu la secció sobre dependències).

```
<components>
```

Components? No estem parlant de connectors? Sí, però en el futur, els connectors no seran més que una classe especial de components. El que fem aquí, és registrar tots els components/connectors en el RKWard. Vegem una entrada d'exemple:

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

Primer l'atribut *type*: deixeu-ho com a *"standard"* per ara. Encara no s'han implementat altres tipus. L'*id* que ja hem indicat. A cada component se li ha de donar un identificador únic (en el seu espai de noms). Trieu-ne un que sigui fàcilment reconeixible. Eviteu espais i qualsevol caràcter especial. Fins ara no estan prohibits, però podrien tenir significats especials. Amb l'atribut *file*, especifiqueu on es troba la descripció del connector real. Això és relatiu al directori on es troba el fitxer `.pluginmap`, i el *base_prefix* anterior. Finalment, doneu una etiqueta al component. Aquesta etiqueta es mostrarà on es col·loqui el connector al menú (o en el futur potser també en altres llocs).

Normalment, un fitxer de `.pluginmap` contindrà diversos components, de manera que aquí en trobareu alguns més:

```
<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>
```

D'acord, aquest ha estat el primer pas. Ara el RKWard coneix que aquests connectors existeixen. Però com invocar-los? S'han de col·locar en una jerarquia de menús:

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

Just a sota de l'etiqueta **<hierarchy>**, comenceu a descriure, en quin **<menu>** haurien d'anar els vostres connectors. Amb la línia anterior, bàsicament dieu que el vostre connector hauria d'estar en el menú **Analysis** (no necessàriament directament allà, sinó en un submenú). El menú **Analysis** és estàndard en el RKWard, de manera que en realitat no s'ha de crear des de zero. Tanmateix, si encara no existís, utilitzant l'atribut *label* li donaríeu el seu nom. Finalment, l'*id* identifica de nou aquest **<menu>**. Això és necessari, de manera que diversos fitxers del `.pluginmap` poden col·locar els seus connectors en els mateixos menús. Ho fan buscant un **<menu>** amb l'*id* donat. Si l'ID encara no existeix, es crearà un menú nou. En cas contrari, les entrades s'afegiran al menú existent.

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

Bàsicament, aquí és el mateix: ara definim un submenú al menú **Analysis**. Es diu **Means**).

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

I un nivell final en la jerarquia del menú: un submenú del submenú **Means**.

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

Ara, finalment, aquest és el menú en què volem col·locar el connector. L'etiqueta **<entry>** indica que aquesta és realment la cosa real, en lloc d'un altre submenú. L'atribut *component* es refereix a l'*id* que heu donat al connector/component anterior.

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

En el cas que us hàgiu perdut: aquest és un altre submenú al menú **Analysis**. Vegeu la captura de pantalla següent. S'ometran algunes de les coses que no són visibles, marcades amb [...].

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

Aquestes són les entrades finals visibles en les captures de pantalla de sota.

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

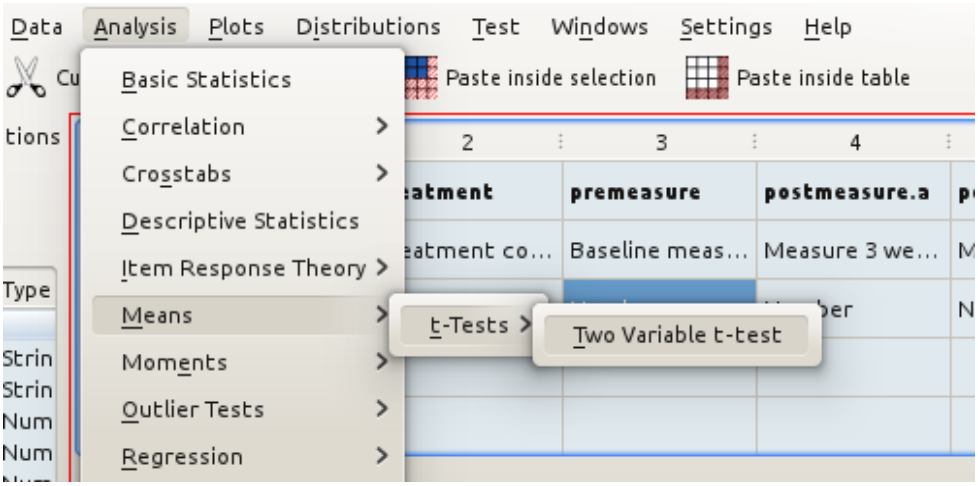
Per descomptat, també podeu col·locar els vostres connectors en menús diferents d'**Analysis**.

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

Fins i tot en menús estàndard com **Fitxer**. Tot el que necessiteu és l'`id` correcte.

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

Així és com es fa. I aquesta captura de pantalla mostra el resultat:




És confús? La manera més fàcil d'iniciar-se és probablement prenent alguns dels fitxers existents del `.pluginmap` enviats amb la distribució, i modificant-los segons les vostres necessitats. A més, si necessiteu ajuda, no dubteu a escriure a la llista de correu de desenvolupament.

## 3.1 Control de l'ordre de les entrades del menú

De manera predeterminada, tots els elements (entrades/submenús) dins d'un menú s'ordenaran alfabèticament, automàticament. En *alguns* casos, és possible que vulgueu més control. En aquest cas podeu agrupar elements de la manera següent:

- Podeu definir grups dins de qualsevol menú com aquest. Tots els elements que pertanyin al mateix grup s'agruparan junts:

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

- Si voleu que el grup estigui visualment separat d'altres entrades, utilitzeu:

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

- Les entrades, menús i grups es poden afegir a un grup especificat, utilitzant:

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

- De fet, també és possible definir grups (sense línies de separació) implícitament:

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

- Els noms dels grups són específics de cada menú. El grup «a» al menú «Data» no entra en conflicte amb el grup «a» del menú «Analysis», per exemple.
- El cas d'ús més comú és definir grups a la part superior, o a la part inferior d'un menú. Per això, hi ha grups predefinits ″top″ i ″bottom″ a cada menú.
- Les entrades dins de cada grup estan ordenades alfabèticament. Els grups apareixen en l'ordre de declaració (llevat que s'afegeixi a un altre grup, per descomptat).
- Els menús i les entrades sense especificació de grup formen lògicament un grup (″″).

# Capítol 4

# Definir la IGU

## 4.1 Definir un diàleg

En el capítol anterior heu vist com registrar un connector amb el RKWard. L'ingredient més important era especificar el camí a un fitxer XML amb una descripció de l'aspecte real del connector. En aquest capítol aprendreu a crear aquest fitxer XML.

> SUGGERIMENT
> Després de llegir aquest capítol, mireu també el paquet rkwarddev. Proporciona algunes funcions de l'R per a crear la majoria de les etiquetes XML del RKWard.

Una vegada més, us ensenyarem amb un exemple. L'exemple és una versió (lleugerament simplificada) de la «prova t» de dues variables.

```
<!DOCTYPE rkplugin>
```

El «doctype» realment encara no s'interpreta. Poseu-lo a *rkplugin*, de totes maneres.

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

Tots els connectors generen codi. Actualment, l'única manera de fer-ho és utilitzant JS tal com es detalla al capítol següent. Això defineix on cercar el codi JS. El nom del fitxer és relatiu al directori on es troba el connector XML.

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

Normalment, és una bona idea proporcionar també una pàgina d'ajuda per al vostre connector. El nom del fitxer d'aquesta pàgina d'ajuda es dona, aquí, en relació amb el directori, a on està l'XML del connector. L'escriptura de pàgines d'ajuda està documentada aquí. Si no proporcioneu un fitxer d'ajuda, ometeu aquesta línia.

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

Com ja sabeu, els connectors poden tenir un diàleg o una interfície assistent o ambdós. Aquí comencem a definir una interfície de diàleg. L'atribut *label* especifica la llegenda del diàleg.

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

Els elements de la IGU es poden organitzar utilitzant un «tabbook». Aquí definim un «tabbook» com el primer element del diàleg. Utilitzeu **<tabbook>**[...]**</tabbook>** per a definir el «tabbook» i després per a cada pàgina del «tabbook» utilitzeu **<tab>**[...]**</tab>**. L'atribut *label* a l'element **<tab>** us permet especificar una llegenda per a aquesta pàgina del «tabbook».

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

Les etiquetes **<row>** i **<column>** especifiquen la disposició dels elements de la IGU. Aquí diu que vol col·locar alguns elements un al costat de l'altre (d'esquerra a dreta). L'atribut *id* no és estrictament necessari, però l'utilitzarem més endavant, quan afegiu una interfície assistent al nostre connector. El primer element a col·locar a la fila és:

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

Amb aquesta etiqueta simple creareu una llista des de la qual l'usuari podrà seleccionar variables. Heu d'especificar un *id* per a aquest element, de manera que el RKWard sàpiga com trobar-lo.

> AVÍS
> NO podeu utilitzar un punt (.) a la cadena *id*.

```
<column>
```

A continuació, imbriquem un **<column>** dins la fila. Aquests són els elements següents que es col·locaran sobre dels altres (de dalt a baix), i tots estaran a la dreta del **<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"/>
```

Aquests elements són l'equivalent a **<varselector>**. Representen «ranures» en les quals l'usuari pot posar variables. Observeu que *source* s'estableix al mateix valor que la *id* de **<varselector>**. Això vol dir que els **<varslot>** prendran cadascuna de les seves variables del «varselector». També cal donar als **<varslot>** un *id*. Poden tenir una *label*, i es poden establir a *required*. Això vol dir que el botó **Submit** no estarà habilitat fins que el **<varslot>** tingui un valor vàlid. Finalment, l'atribut *type* encara no s'ha interpretat, però s'utilitzarà per a tenir en compte que només es permeten els tipus correctes de variables a **<varslot>**.

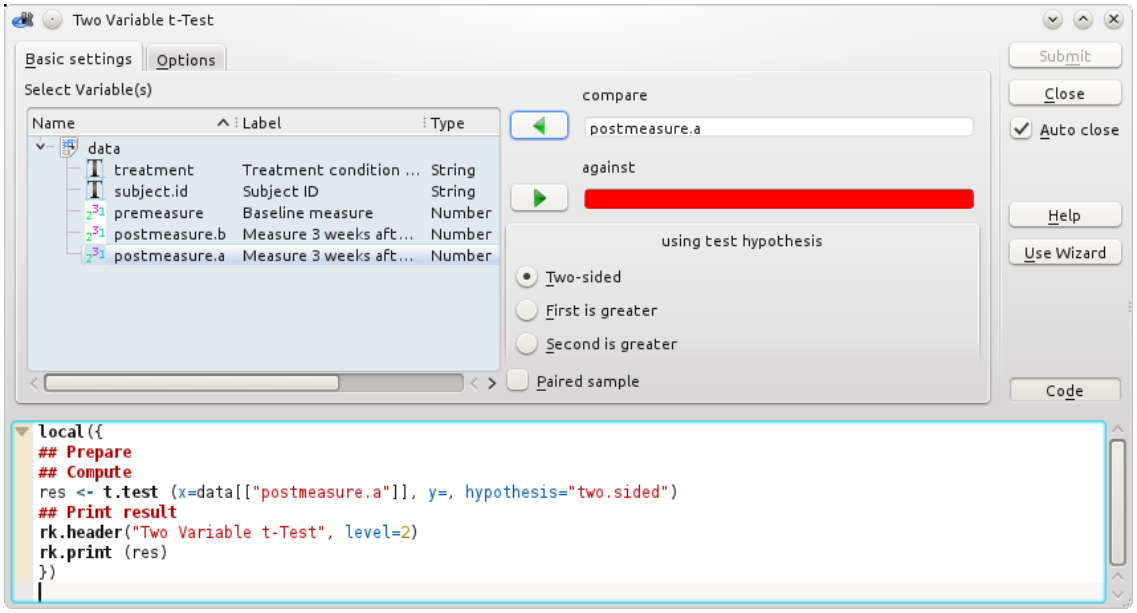
En cas que us pregunteu sobre l'atribut *i18n_context*: aquest és per a proporcionar context per a ajudar a la traducció correcta de la paraula "against", utilitzada com a etiqueta de **<varslot>**, però no afecta directament la funcionalitat del connector. Més sobre això en un capítol separat.

```
<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>
```

Aquí, definiu un grup de botons exclusius **<radio>**. El grup té una *label* i un *id*. Cada **<option>** (botó) té una *label* i se li assigna un *value*. Aquest és el valor que l'element **<radio>** retornarà quan se seleccioni l'opció.

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

Cal tancar cada etiqueta. Hem posat tots els elements que volíem (els dos **<varslots>** i el **<radio>**) a la **<column>**). Posem tots els elements que volíem (el **<varselector>** i el **<column>** amb aquests elements) en la **<row>**. I hem posat tots els elements que volíem a la primera pàgina del **<tabbook>**. Encara no hem acabat de definir el **<tabbook>** (vindran més pàgines), i per descomptat hi ha més en el **<dialog>**, també. Però aquesta captura de pantalla és bàsicament el que hem fet fins ara:




Tingueu en compte que no s'han especificat els botons **Submit**, **Close**, etc. o la vista de codi. Aquests elements es generen automàticament. Però, per descomptat, encara hem de definir la segona pàgina del **<tabbook>**:

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

De manera predeterminada els elements es col·locaran de dalt a baix com en una **<column>**. Com que això és el que volem aquí, no hem d'indicar explícitament una disposició **<row>** ni **<column>**. El primer element que definim és una casella de selecció. Igual que **<radio><option>**, la casella de selecció té una *label* i un *value*. El *value* és el que es retorna, si la casella de selecció està marcada. Per descomptat, la casella de selecció també necessita un *id*.

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

Aquí hi ha un altre element de la disposició: per a indicar que els dos elements següents estan junts, dibuixem un **<frame>** (quadre). Aquest marc pot tenir una *label* (llegenda). Com que el marc només és un element de disposició passiva, no necessita un *id*, tot i que en definim un aquí, com que hi farem referència més tard, quan definim una interfície assistent addicional.

```
<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>
```

Dins del **<frame>** col·loquem un altre **<checkbox>** (usant *checked="true"*, senyalem que la casella de selecció s'ha de marcar de manera predeterminada), i un **<spinbox>**. El botó de selecció de valors permet a l'usuari seleccionar un valor entre *"min"* i *"max"* amb el valor per defecte/inicial *"0,95"*. Establir el *type* a *"real"* indica que s'accepten els nombres reals en lloc de *type="integer"* que només acceptaria enters.

> NOTA
> També és possible, i sovint preferible, fer que el **<frame>** es pugui marcar, en lloc d'afegir una **<checkbox>** a l'interior. Vegeu la referència per a més detalls. Això no es fa aquí, amb finalitats il·lustratives.

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

Això és tot en la segona pàgina del **<tabbook>**, totes les pàgines del **<tabbook>** i tots els elements en el **<dialog>**. Hem acabat de definir l'aspecte del diàleg.

```
</document>
```

Finalment tanquem l'etiqueta **<document>**, i ja està. La IGU està definida. Ara podeu desar el fitxer. Però com es genera la sintaxi de l'R a partir de la configuració de la IGU? Ho tractarem en el capítol següent. En primer lloc, però, mirarem d'afegir una interfície assistent i algunes consideracions generals.

## 4.2 Afegir una interfície assistent

En realitat no cal definir una interfície addicional **<wizard>**, però així és com es faria. Per a afegir una interfície assistent, afegiu una etiqueta **<wizard>** al mateix nivell que l'etiqueta **<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>
```

Algunes d'aquestes coses s'expliquen per si mateixes: afegim una etiqueta **<wizard>** amb una *label* per a l'assistent. Com que un assistent pot contenir diverses pàgines que es mostren una després de l'altra, a continuació definim la primera nota **<page>**, i hi posem una nota explicativa **<text>**. Llavors utilitzem una etiqueta **<copy>**. El que fa això, és que realment ens estalvia haver de definir de nou el que ja escrivim per al **<dialog>**: l'etiqueta copy cerca una altra etiqueta amb el mateix *id* abans en l'XML. Això es defineix en la secció **<dialog>**, i és un **<row>** en el qual hi ha el **<varselector>**, **<varslots>** i el control **<radio>** de la «hipòtesi». Tot això es copia 1:1 i s'insereix just a l'element **<copy>**.

Ara la segona pàgina:

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

Aquí ocorre el mateix. S'afegeixen alguns textos, i entre ells hi ha la **<copy>** d'altres seccions de la interfície de diàleg.

Per descomptat, podeu fer que la interfície de l'assistent sembli molt diferent del diàleg, i no utilitzar l'etiqueta **<copy>** en absolut. Assegureu-vos, però, d'assignar els elements corresponents el mateix *id* en ambdues interfícies. Això no només s'utilitza per a transferir la configuració des de la interfície de diàleg a la interfície assistent i al revés, quan l'usuari canvia d'interfície (això encara no succeeix a la versió actual del RKWard), sinó que també simplifica l'escriptura de la plantilla de codi (vegeu a sota).

## 4.3 Algunes consideracions sobre el disseny de la IGU

Aquesta secció conté algunes consideracions generals sobre quins elements de la IGU s'utilitzaran i a on. Si aquest és el primer intent de crear un connector, no dubteu a ometre aquesta secció, ja que no és rellevant per a aconseguir que una IGU bàsica funcioni. Torneu aquí, més tard, per a veure si podeu refinar la IGU del connector d'alguna manera o d'altra.

### 4.3.1 <radio>, <checkbox> i <dropdown>

Els tres elements **<radio>**, **<checkbox>**, **<dropdown>**, tots tenen una funció similar: per a seleccionar una de diverses opcions. Òbviament, una casella de selecció només permet triar entre dues opcions: marcada o no, de manera que no la podeu utilitzar si hi ha més de dues opcions per a triar. Però quan utilitzar quin dels elements? Algunes regles generals:

Si esteu creant un **<radio>** o **<dropdown>** amb només dues opcions, pregunteu-vos si la pregunta és essencialment un tipus de pregunta sí/no. Per exemple, una tria entre «ajusta els resultats» i «no ajusta els resultats», o entre «elimina els valors que manquen» i «mantén els valors que manquen». En aquest cas una **<checkbox>** és la millor opció: utilitza poc espai, tindrà menys paraules d'etiquetes i és més fàcil de llegir per l'usuari. Hi ha molt poques situacions en què hauríeu de triar un **<radio>** sobre una **<checkbox>** quan només hi ha dues opcions. Un exemple d'això podria ser: «Mètode de càlcul: 'pearson'/'spearman'». Aquí, podrien pensar-se més mètodes, i en realitat no formen un parell de contraris.

Triar entre **<radio>** i **<dropdown>** és principalment una qüestió d'espai. El **<dropdown>** té l'avantatge d'utilitzar poc espai, fins i tot si hi ha moltes opcions per a triar. D'altra banda, un **<radio>** té l'avantatge de fer visibles alhora totes les opcions possibles per a l'usuari, sense fer clic a la fletxa desplegable. Generalment, si hi ha sis o més opcions per a triar, és preferible un **<dropdown>**. Si hi ha cinc o menys opcions, un **<radio>** és la millor opció.

# Capítol 5

# Generació de codi R a partir de la configuració de la IGU

## 5.1 Ús del JavaScript en els connectors del RKWard

Ara tenim una IGU definida, però encara necessitem generar codi R a partir d'això. Per tant, necessitem un altre fitxer de text, `code.js`, ubicat al mateix directori que el `description.xml`. Podeu estar o no familiaritzats amb el JavaScript (o, per a ser tècnicament precís: ECMA-script). La documentació sobre JS es pot trobar en abundància, tant en forma impresa, com a Internet (p. ex.: https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide). Però per a la majoria de les finalitats no necessitareu saber gaire sobre JS, ja que només utilitzarem algunes característiques molt bàsiques.

> SUGGERIMENT
> Després de llegir aquest capítol, també doneu un cop d'ull al paquet rkwarddev. Proporciona algunes funcions de l'R per a crear codi JavaScript utilitzat habitualment en el RKWard. També pot detectar automàticament les variables utilitzades en un fitxer XML de connector i crear codi bàsic JavaScript a partir d'aquest per a començar.

> NOTA
> S'assumeix que els fitxers `.js` del connector estan codificats en UTF-8. Assegureu-vos de comprovar la codificació de l'editor, si utilitzeu qualsevol caràcter no ASCII.

Per a la prova t de dues variables, el fitxer `code.js` es mostra de la manera següent (amb comentaris entremig):

### 5.1.1 preprocess()

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

El fitxer JS està organitzat en tres funcions separades: `preprocess()`, `calculate()`, i `printout()`. Això es deu al fet que no es necessita tot el codi en totes les etapes. Actualment, la funció de preprocessament no s'utilitza realment en molts llocs (normalment l'ometreu).

### 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');
}
```

Aquesta funció genera la sintaxi real de l'R que s'executarà des de la configuració de la IGU. Mirem això en detall: el codi que s'utilitza es genera utilitzant la sentència `echo()`. Mirant la sentència `echo()` pas a pas, la primera part d'aquesta és

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

com a text net. A continuació cal omplir el valor que l'usuari ha seleccionat com a primera variable. Ho obtenim utilitzant `getString ("x")`, i l'afegeix a la cadena per a ser «reproduït». Això mostra el valor de l'element de la IGU amb *`id="x"`*: la nostra primera **<checkbox>**. A continuació, afegim una «, » i fem el mateix per a obtenir el valor de l'element *`"y"`*, la segona **<checkbox>**. Per a la hipòtesi (el grup **<radio>**), i les variàncies iguals **<checkbox>**, el procediment és molt similar.

Tingueu en compte que en lloc de concatenar els fragments de sortida amb «+», també podeu utilitzar diverses sentències `echo()`. Tot s'imprimeix en una sola línia. Per a produir un salt de línia en el codi generat, inseriu una *`"\n"`* en la cadena reproduïda. En teoria, fins i tot podeu produir moltes línies amb una única sentència d'eco, però manteniu-la en una (o menys) línia de codi generat per `echo()`.

> NOTA
> A més de `getString()`, també hi ha funcions `getBoolean()`, que intentaran retornar el valor com un element binari (apropiat per a utilitzar en una sentència `if()`), i `getList()`, que intentarà retornar dades semblants a una llista en un JS `Array()`. Més tard mostrarem exemples d'això.
> En mirar els connectors existents, també trobareu molts connectors que utilitzen `getValue()`, en lloc de `getString()`, i de fet els dos són *gairebé* idèntics. No obstant això, utilitzar `getString()`, `getBoolean()` i `getList()` és la pràctica recomanada des de la versió 0.6.1.

Es torna una mica més complicat per al nivell de confiança. Per raons d'estètica, no volem especificar explícitament el nivell de confiança a utilitzar, si correspon al valor predeterminat. Per tant, en lloc d'imprimir el valor incondicionalment, primer el recuperarem en una variable. Després comprovem si aquesta variable difereix de *`"0,95"`* i, si és així, imprimeix un argument addicional. Finalment, reproduïm un parèntesi de tancament i un salt de línia: *`")\n"`*. Això és tot per a la funció de càlcul.

### 5.1.3 printout()

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

I això era tot el que hi ha a la funció d'impressió en la majoria dels casos. `rk.header()` imprimeix un titular estàndard del resultat. Tingueu en compte que en els fitxers `.js` haureu de marcar totes les cadenes traduïbles a mà, utilitzant **i18n()**, o altres ordres alternatives. Més informació en el capítol sobre internacionalització. També podeu afegir més informació a això, si voleu, p. ex.:

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

`rk.print()` utilitza el paquet R2HTML per a proporcionar una sortida amb format HTML. Una altra funció útil és `rk.results()`, que també pot generar tipus diferents de taules de resultats. No obstant això, si hi ha dubtes, només cal utilitzar `rk.print()`, i amb això és suficient. La classe JS `Header` és un ajudant de nivell JS per a generar una crida a `rk.header()` (només cal donar un cop d'ull al codi R generat). En alguns casos, és possible que vulgueu cridar `echo («rk.header (...)»)` directament per a imprimir una capçalera per la sortida.

Tingueu en compte que internament la sortida és només un document HTML normal en aquest moment. Per tant, podeu estar temptats d'afegir HTML personalitzat utilitzant `rk.cat.output()`. Encara que això funcionarà, no ho feu. El format de sortida pot canviar (p. ex., a ODF) en el futur, de manera que és millor no introduir codi HTML específic. Més aviat, manteniu-ho senzill amb `rk.header()`, `rk.print()`, `rk.results()`, i si cal, `rk.print.literal()`. Si això no sembla satisfer les vostres necessitats de format, contacteu amb nosaltres a la llista de correu per a obtenir ajuda.

Felicitats! Heu creat el vostre primer connector. Llegiu en els capítols següents quant a conceptes més avançats.

## 5.2 Convencions, polítiques i coneixement general

Hi ha moltes maneres d'escriure codi R per a una determinada tasca, i hi ha encara més maneres de generar aquest codi R a partir de JS. Com ho feu exactament, és cosa vostra. Encara hi ha una sèrie de consideracions que hauríeu de seguir, i la informació de base que hauríeu d'entendre.

### 5.2.1 Entendre l'entorn `local()`

Amb freqüència haureu de crear un o més objectes R temporals en el codi generat pel connector. Normalment, no voleu que es col·loquin a l'espai de treball de l'usuari, fins i tot sobreescrivint les variables d'usuari. Per tant, tot el codi generat pel connector s'executa en un entorn `local()` (vegeu la pàgina d'ajuda de la funció `local()` de l'R). Això vol dir que totes les variables que creeu són temporals i no es desaran permanentment.

Si l'usuari demana explícitament que es desi una variable, haureu d'assignar a aquest objecte utilitzant `.GlobalEnv$objectname <- value`. En general, no utilitzeu l'operador `<<`. No s'assignarà necessàriament a .GlobalEnv.

Un escull important és utilitzar `eval()`. Aquí, haureu de tenir en compte que «eval» utilitzarà de manera predeterminada l'entorn actual per a l'avaluació, és a dir, el local. Això funcionarà bé la majoria de les vegades, però no sempre. Per tant, si necessiteu utilitzar `eval()`, probablement voldreu especificar el paràmetre *`envir`*: `eval(..., envir=globalenv())`.

### 5.2.2 Format del codi

El més important és que el codi R generat funcioni, però també hauria de ser fàcil de llegir. Per tant, vigileu també el format. Algunes consideracions:

S'han d'alinear a l'esquerra les sentències R normals de nivell superior.

Les sentències d'un bloc inferior s'han de sagnar amb una tabulació (vegeu l'exemple a continuació).

Si feu càlculs molt complexos, afegiu un comentari aquí i allà, especialment per a marcar les seccions lògiques. Tingueu en compte que hi ha una funció dedicada **comment()** per a inserir comentaris traduïbles en el codi generat.

Per exemple, el codi generat podria tenir aquest aspecte. El mateix codi sense sagnat o comentaris seria bastant difícil de llegir, malgrat la seva modesta complexitat:

```
# primer determina el balanceig i la rotació
my.wobble <- wobble (x, y)
my.rotation <- wobble.rotation (my.wobble, z)

# cal triar el mètode de balanceig segons la rotació
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 Tractament amb opcions complexes

Molts connectors poden fer més d'una cosa. Per exemple, el connector «Estadístiques descriptives» pot calcular la mitjana, l'interval, la suma, el producte, la mediana, la longitud, etc. No obstant això, normalment l'usuari només triarà fer alguns d'aquests càlculs. En aquest cas, intenteu mantenir el codi generat el més senzill possible. Només hauria de contenir porcions rellevants per a les opcions realment seleccionades. Per a aconseguir-ho, aquí hi ha un exemple d'un patró de disseny comú tal com l'utilitzaríeu (en JS; aquí, ″domean″, ″domedian″ i ″dosd″ serien elements de <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 Consells i trucs

Aquí hi ha alguns trucs diversos que poden fer que l'escriptura de connectors sigui menys tediosa:

Si necessiteu el valor d'un paràmetre de la IGU en diversos llocs en el codi del connector, considereu assignar-la a una variable en JS, i utilitzant-la en lloc de recuperar-la una vegada i una altra amb `getString()/getBoolean()/getList()`. Això és més ràpid, més llegible i amb menys tecleig, tot alhora:

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

La simple funció d'ajuda `makeOption()` pot facilitar l'omissió dels paràmetres que tenen el seu valor predeterminat, en molts casos:

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

# Capítol 6

# Escriure una pàgina d'ajuda

Quan el vostre connector funciona bàsicament, ha arribat el moment de proporcionar una pàgina d'ajuda. Encara que normalment no voldreu explicar tots els conceptes subjacents en profunditat, és possible que vulgueu afegir alguna explicació més per a alguna de les opcions, i enllaçar amb connectors relacionats i funcions de l'R.

> SUGGERIMENT
> Després de llegir aquest capítol, doneu també un cop d'ull al paquet rkwarddev. Proporciona algunes funcions de l'R per a crear la majoria de les etiquetes XML del RKWard. També és capaç de crear esquelets bàsics de fitxers d'ajuda a partir dels fitxers XML existents per a començar.

Potser recordareu posar això dins del connector XML (si no l'heu posat, feu-ho ara):

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

On, òbviament, substituireu `filename` per un nom més apropiat. Ara és el moment de crear aquest fitxer `.rkh`. Aquest és un exemple autodescriptiu:

```
<!DOCTYPE rkhelp>
<document>
        <summary>
En aquesta secció, posareu informació breu i molt bàsica sobre què fa el  ↩
    connector.
Aquesta secció sempre es mostrarà a la part superior de la pàgina d'ajuda.
        </summary>

        <usage>
La secció d'ús pot contenir una mica més d'informació pràctica. Però
no explica tots els paràmetres en detall (això es fa a la secció «settings» ↩
    ).

Per a iniciar un paràgraf nou, inseriu una línia buida, com es mostra a  ↩
    dalt.
Aquesta línia, en canvi, serà al mateix paràgraf.

En totes les seccions podeu inserir codi HTML senzill, com ara el text <b> ↩
    bold</b> o <i>italic</i>. No obstant això, manteniu el format al mínim  ↩
    necessari.
```

```
La secció d'ús és sempre la segona secció que es mostra en una pàgina d' ↩
    ajuda.
        </usage>

        <section id="sectionid" title="Secció genèrica" short_title=" ↩
            Genèrica">
Si cal, podeu afegir seccions addicionals entre les seccions d'ús i de  ↩
    configuració.
No obstant això, normalment no ho necessitareu mentre documenteu els  ↩
    connectors.
L'atribut «id» proporciona un punt d'ancoratge per a saltar a aquesta  ↩
    secció
des del menú de navegació. L'atribut «short_title» proporciona un títol  ↩
    curt per a utilitzar a la barra de navegació.
Això és opcional, per defecte el "títol" principal s'utilitzarà tant com a  ↩
    encapçalament de la secció, com a nom de l'enllaç a la barra de  ↩
    navegació.

En qualsevol secció podeu inserir enllaços per a més informació. Ho feu  ↩
    afegint

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

A on URL pot ser un enllaç extern com https://rkward.kde.org .
Es permeten diversos URL especials a les pàgines d'ajuda:

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

Això enllaça a una pàgina d'ajuda de nivell superior del RKWard (no per a  ↩
    un connector).

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

Això enllaça a la pàgina d'ajuda d'un altre connector. La part [namespace/] ↩
     es pot ometre
(en aquest cas, s'assumeix el RKWard com a espai de noms estàndard, p. ex.:
<link href="rkward://component/import_spss"/> o
<link href="rkward://component/rkward/import_spss"/> són equivalents).
El «component_id» és el mateix que heu especificat en el .pluginmap.

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

Enllaça a la pàgina d'ajuda "rfunction" de l'R .

Tingueu en compte que els noms dels enllaços es generaran automàticament  ↩
    per a aquests tipus d'enllaços.
        </section>

        <settings>
                <caption id="id_of_tab_or_frame"/>
                <setting id="id_of_element">
Descripció de l'element de la IGU identificat per l'id indicat
                </setting>
                <setting id="id_of_elementb" title="descripció">
Normalment el títol de l'element de la IGU s'extraurà a partir de la
definició XML del connector, automàticament. Tanmateix,
per a alguns elements de la IGU aquesta descripció pot no ser suficient per ↩
```

```
    a identificar-los amb fiabilitat.
En aquest cas podeu afegir un títol explícit usant l'atribut "title".
                </setting>
                <setting id="id_of_elementc">
Descripció de l'element de la IGU identificat per "id_of_elementc"
                </setting>
                [...]
        </settings>

        <related>
La secció de relacions normalment conté diversos enllaços, com ara:

<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>
La secció tècnica (opcional, sempre l'última) pot contenir alguns detalls ↩
    tècnics de la implementació del connector,
que només són d'interès per als desenvolupadors del RKWard. Això és ↩
    particularment rellevant
per als connectors que estan dissenyats per a ser incrustats en molts ↩
    altres connectors,
i podria detallar quines opcions estan disponibles per a personalitzar el ↩
    connector incrustat,
i quines seccions de codi contenen quin codi R.
        </technical>
</document>
```

# Capítol 7

# Interaccions lògiques entre elements de la IGU

## 7.1 Lògica de la IGU

Tots els conceptes bàsics de crear un connector per al RKWard s'han descrit en els capítols anteriors. Aquests conceptes bàsics haurien de ser suficients per a molts casos, si no la majoria. No obstant això, de vegades voleu més control sobre com es comporta la IGU del connector.

Per exemple, suposem que voleu ampliar l'exemple de la prova t utilitzat en aquesta documentació per a permetre: comparar una variable amb una altra variable (com es mostra), així com comparar una variable amb un valor constant. Ara, una manera de fer-ho seria afegir un control d'opcions que canviï entre els dos modes, i afegir un botó de selecció de valors per a introduir el valor constant amb el qual comparar. Considereu aquest exemple simplificat:

```
<!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>
```

Fins ara està bé, però hi ha una sèrie de problemes amb aquesta IGU. En primer lloc, sempre es mostren tant el «varslot» com el «spinbox», mentre que realment només s'utilitza un dels dos. Pitjor encara, el «varslot» sempre requereix una selecció vàlida, fins i tot si es compara amb una constant. Òbviament, si creem una IGU multiús com aquesta, volem més flexibilitat. Introduïu: la secció **<logic>** (inserida al mateix nivell que **<code>**, **<dialog>**, o **<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">
        [...]
```

La primera línia dins de la secció «logic» és una etiqueta **<convert>**. Bàsicament, això proporciona una propietat booleana nova (activada o desactivada, certa o falsa), que es pot utilitzar més endavant. Aquesta propietat (*˜varmode˜*) és certa, sempre que se seleccioni el botó d'opció superior, i fals quan se seleccioni el botó d'opció inferior. Com es fa això?

En primer lloc, sota *sources*, s'enumeren les propietats de les fonts («sources») en les quals treballar (en aquest cas només una cadascuna; es podrien llistar com a *sources=˜mode.string;somethingelse˜*, llavors *˜varmode˜* només seria cert, si tant *˜mode.string˜* com *˜somethingelse˜* són iguals a la cadena *˜variable˜*). Tingueu en compte que en aquest cas no només escrivim *˜mode˜* (com ho faríem a `getString(˜mode˜)`), sinó *˜mode.string˜*. En realitat, aquesta és la manera interna en què funciona un control d'opcions: té una propietat 'string', que té el seu valor de cadena. `getString(˜mode˜)` és només una abreviatura, i equival a `getString(˜mode.string˜)`. Vegeu la referència per a totes les propietats dels diferents elements de la IGU.

En segon lloc, hem establert el mode de conversió a *mode=˜equals˜*. Això vol dir que volem comprovar si la/es font/s és/són igual/s que un valor determinat. Finalment «standard» és el valor contra el qual comparar, de manera que amb *standard=˜variable˜*, es comprova si la propietat *˜mode.string˜* és igual que la cadena *˜variable˜* (el valor de l'opció del botó superior). Si és igual, llavors la propietat «varmode» és certa, altrament és falsa.

Passem a les coses reals: **<connect>** la propietat *˜varmode˜* a «y.visible», que controla si es mostra o no la gràfica *˜y˜*. Tingueu en compte que qualsevol element que es fa invisible és implícitament no obligatori. Per tant, si se selecciona l'opció del botó superior, es requereix el «varslot» *˜y˜*, i és visible. Si no, no és necessari ni ocult.

Per al botó de selecció de valors, volem exactament el contrari. Afortunadament, no necessitem un altre **<convert>** per a això: les propietats booleanes es poden negar molt fàcilment afegint el modificador *˜no˜*, així que **<connect>** *˜varmode.not˜* a la propietat de visibilitat del botó de selecció de valors. En efecte, es mostra i es requereix el «varslot», *o* es mostra el botó de selecció de valors i es requereix (depenent de quina opció està seleccionada en el control d'opcions). La IGU està canviant segons l'opció del botó. Proveu l'exemple, si voleu.

Per a obtenir una llista completa de propietats, consulteu la referència. Una propietat més, però, és especial perquè tots els elements de la IGU la tenen: 'enabled'. Això és una mica menys dràstic que 'visible'. No mostra/oculta l'element de la IGU, però només l'activa/desactiva. Els elements desactivats es mostren normalment en gris i no reaccionen a l'entrada de l'usuari.

NOTA
A més de **<convert>** i **<connect>**, hi ha diversos elements més per a utilitzar a la secció **<logic>**. Per exemple, les construccions condicionals també es poden implementar utilitzant l'element **<switch>**. Consulteu la referència sobre elements lògics per a més detalls.

## 7.2 Lògica de la IGU amb scripts

Sovint és suficient connectar les propietats tal com es descriu anteriorment, però de vegades és més flexible o més pràctic utilitzar JS per a crear scripts en la lògica de la IGU. D'aquesta manera, l'exemple anterior es podria reescriure com:

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

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

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

La primera línia de codi li diu al RKWard que cridi a la funció `modeChanged()` sempre que canviï el valor del quadre de botons d'opció *`id="mode"`*. Dins d'aquesta funció, definim una variable d'ajuda *`"varmode"`* que és certa quan el mode és *`"variable"`*, falsa quan és *`"constant"`*. Després utilitzem `gui.setValue()` per a establir les propietats 'enabled' de *`"y"`* i *`"constant"`*, de la mateixa manera que abans vam fer servir sentències **<connect>**.

L'enfocament amb scripts a la lògica de la IGU esdevé particularment útil quan voleu canviar l'opció disponible segons el tipus d'objecte que l'usuari ha seleccionat. Vegeu la referència per a les funcions disponibles.

Tingueu en compte que l'enfocament amb scripts a la lògica de la IGU es pot barrejar amb sentències **<connect>** i **<convert>** si voleu. Tingueu en compte també que l'etiqueta **<script>** permet especificar un nom de fitxer de script a més o com a alternativa a la inclusió del codi de script. Normalment la inclusió del codi de script tal com es mostra a dalt és més pràctic.

# Capítol 8

# Incrustar connectors en connectors

## 8.1 Casos d'ús per a incrustar

En escriure connectors, sovint trobareu que esteu creant una sèrie de connectors que només difereixen en alguns aspectes, però tenen molt més en comú. Per exemple, per al traçat, hi ha una sèrie d'opcions R genèriques que es poden utilitzar amb la majoria de tipus de diagrames. Hauríeu de crear una IGU i una plantilla JS per a aquests una vegada i una altra?

Evidentment, això seria una molèstia. Afortunadament, no cal que ho feu. Més aviat creareu la funcionalitat comuna una vegada, i més tard podreu incrustar-la en diversos connectors. De fet, és possible incrustar qualsevol connector en qualsevol altre connector, fins i tot si l'autor original del connector incrustat mai ho va pensar, algú voldria incrustar el seu connector en un altre.

## 8.2 Incrustació dins d'un diàleg

D'acord, ja hem parlat prou. Com funciona? És senzill: utilitzeu l'etiqueta **<embed>**. Aquest és un exemple retallat:

```
<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>
```

El que passa aquí, és que tota la IGU o el connector d'opcions del diagrama (excepte per descomptat per als elements estàndard com el botó **Submit**, etc.) s'incrusta directament en el vostre connector (proveu-ho!).

Com podeu veure la sintaxi de l'etiqueta **<embed>** és força senzilla. Pren un *id* com la majoria d'elements. El component del paràmetre especifica quin connector incrustar, com es defineix al fitxer .pluginmap (*~rkward::plot_options~* és el resultat de concatenar l'espai de noms 'rkward', un separador '::', i el nom del component 'plot_options').

## 8.3 Generació de codi en incrustar

Fins ara tot bé, però què passa amb el codi generat? Com es fusiona el codi del connector incrustador i l'incrustat? En el codi JS del connector incrustador escriviu quelcom com això:

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

Bàsicament, estem recuperant el codi generat pel connector incrustat igual que estem recuperant qualsevol altra opció de la IGU. Aquí la cadena *~plotoptions.code.printout~* es pot desenvolupar com «La secció d'impressió del codi generat de l'element amb plotoptions d'identificador *id*» («plotoptions» és l'ID que hem donat a l'etiqueta **<embed>** anterior). I sí, si voleu un control avançat, fins i tot podreu recuperar els valors dels elements individuals de la IGU dins del connector incrustat (però no a l'inrevés, ja que el connector incrustat no coneix res sobre el seu entorn).

## 8.4 Incrustació dins d'un assistent

Si el vostre connector proporciona una IGU d'assistent, la incrustació funciona bàsicament de la mateixa manera. Generalment utilitzareu:

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

Si el connector incrustat proporciona una interfície assistent, les seves pàgines s'inseriran entre *~page12~* i *~page13~* del vostre connector. Si el connector incrustat només proporciona una interfície de diàleg, s'afegirà una única pàgina nova entre les pàgines *~page12~* i *~page13~*. L'usuari no se n'adonarà mai.

## 8.5 Incrustació menys incrustada: botó d'opcions addicionals

Encara que la incrustació és genial, cal anar amb compte de no excedir-se. Massa funcions dins d'una IGU només fa que sigui difícil trobar les opcions rellevants. Per descomptat, a vegades és possible que vulgueu incrustar una gran quantitat d'opcions (com totes les opcions a `plot()`), però com que són realment opcionals, no les voldreu de manera prominent a la IGU.

Una alternativa és incrustar aquestes opcions «com a un botó»:

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

En aquest cas, s'afegirà un únic botó de prémer al connector, etiquetat **Especifica les opcions de traçat**. Quan premeu aquest botó, apareixerà un diàleg separat, amb totes les opcions del connector incrustat. Fins i tot si aquesta IGU incrustada no és visible la majoria de les vegades, podeu obtenir la seva configuració tal com es descriu a dalt.

> ATENCIÓ
> Probablement el «botó» d'aproximació només s'hauria d'utilitzar per als connectors que mai poden no ser vàlids (per configuració que manca/no vàlida). En cas contrari, l'usuari no podria enviar el codi, però podria tenir dificultats per a esbrinar-ho, el motiu està amagat darrere d'algun botó.

## 8.6 Incrustació/definició de connectors incomplets

Alguns connectors, i de fet, el «plot_options» utilitzat a l'exemple anterior és un d'ells, no es completen per si sols. Simplement no tenen els elements de la IGU per a seleccionar alguns valors importants. Estan destinats només a ser incrustat en altres connectors.

Fins a quin punt està incomplet el connector «plot_options»? Bé, per algunes opcions de configuració, necessita conèixer el nom dels objectes/expressions dels eixos x i y (de fet, funcionarà bé si només en té un, però necessita almenys un per a funcionar correctament). No obstant això, no té un mecanisme per a seleccionar aquests objectes, ni introduir-los d'una altra manera. Com sap d'ells?

A la secció lògica del connector «plot_options» hi ha dues línies addicionals, encara no explicades:

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

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

Això defineix dues propietats addicionals al connector «plot_options», l'únic propòsit de les quals és connectar-se a algunes propietats (encara desconegudes) del connector incrustat. Al connector «plot_options» aquestes dues propietats s'utilitzen simplement com qualsevol altra, i per exemple hi ha crides a `getString(˜xvar˜)` en la plantilla JS «plot_options».

Ara, per al connector incomplet no hi ha manera de saber on s'incrustarà, i quina serà la configuració rellevant en el connector incrustant. Per tant, també cal afegir dues línies addicionals a la secció lògica del connector incrustant:

```
<logic>
                [...]

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

Això no és res nou en principi, hem explicat les sentències **<connect>** en el capítol de lògica de la IGU. Simplement connecteu els valors en dos «varlots» (anomenats *"xvarslot"* i *"yvarslot"* en aquest exemple) a les propietats 'external' rebudes del connector incrustat. Això és tot. Tota la resta es prepara automàticament.

# Capítol 9

# Tractament amb molts connectors similars

## 9.1 Vista general de diferents enfocaments

De vegades, és possible que vulgueu desenvolupar connectors per a una sèrie de funcions similars. Per exemple, considereu els diagrames de distribució. Aquests generen codi força similar, i per descomptat és desitjable fer que les interfícies gràfiques s'assemblin entre si. Finalment, grans seccions dels fitxers d'ajuda poden ser idèntiques. Només uns quants paràmetres són diferents en cada connector.

L'enfocament ingenu d'això és desenvolupar un connector, després bàsicament copiar i enganxar tot el contingut dels fitxers `.js`, `.xml`, i `.rkh`, després canviar les poques porcions que són diferents. No obstant això, i si algun temps després trobeu un error ortogràfic que s'ha copiat i enganxat a tots els connectors? I si voleu afegir suport per a una característica nova? Hauríeu de tornar a visitar tots els connectors i canviar-ho a cadascun. Un procés pesat i tediós.

Un segon enfocament seria utilitzar incrustacions. Tanmateix, en alguns casos això no es presta bé al problema que tenim entre mans, principalment perquè els «fragments» que podeu incrustar són de vegades massa grans per a ser útils, i posa algunes restriccions en la disposició. Per a aquests casos, els conceptes incloent fitxers `.js`, incloent fitxers `.xml` i fragments poden ser molt útils (però vegeu els pensaments sobre quan és preferible utilitzar la incrustació).

Algunes paraules a tenir en compte abans de començar a llegir: aquests conceptes poden ajudar a simplificar la gestió de molts connectors similars, i poden millorar el manteniment i la llegibilitat d'aquests connectors. No obstant això, l'excés pot conduir fàcilment a l'efecte invers. Utilitzeu-ho amb cura.

## 9.2 Ús de la sentència «include» del JS

Podeu incloure fàcilment un fitxer de script en un altre en els connectors del RKWard. El valor d'això esdevé immediatament obvi si algunes seccions del codi JS són similars entre els connectors. Podeu definir aquestes seccions en un fitxer `.js` separat, i incloure'l en tots els fitxers `.js` del connector. Per exemple, com a:

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

function doCommonStuff () {
        // perhaps fetch some options, etc.
        // ...
```

```
        comment ("This is R code you want in several different plugins\n");
        // ...
}
```

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

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

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

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

Tingueu en compte que de vegades és encara més útil invertir això, i definir l'«esquelet» de les funcions `preprocess()`, `calculate()`, i `printout()` en un fitxer comú, i fer que aquestes crides tornin per a aquelles parts que són diferents entre els connectors. P. ex.:

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

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

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

        // ...
}
```

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

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

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

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

Un problema que hauríeu de tenir en compte quan utilitzeu aquesta tècnica és l'àmbit de les variables. Vegeu el manual del JS sobre els àmbits de les variables.

Aquesta tècnica s'utilitza molt en els connectors de traçat de distribució i de traçat de TLC, de manera que és possible que vulgueu cercar-hi exemples.

## 9.3 Incloure els fitxers `.xml`

Bàsicament, la mateixa característica d'incloure fitxers també està disponible per al seu ús en els fitxers `.xml`, `.pluginmap` i `.rkh`. En qualsevol lloc d'aquests fitxers podeu posar una etiqueta **<include>** com es mostra a continuació. L'efecte és que tot el contingut d'aquest fitxer XML (per

a ser precisos: tot dins de l'etiqueta **<document>** d'aquest fitxer) s'inclou literalment en aquest punt en el fitxer. Recordeu que només podeu incloure un altre fitxer XML.

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

L'atribut *file* és el nom del fitxer relatiu al directori on es troba el fitxer actual.

## 9.4 Ús de <snippets>

Si bé incloure fitxers com es mostra a la secció anterior és bastant potent, es torna més útil quan s'utilitza en combinació amb **<snippets>**. Els «snippets» (fragments) són seccions més petites que podeu inserir en un altre punt del fitxer. Un exemple il·lustra millor això:

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

Per tant, definiu el fragment en un lloc a la part superior del fitxer XML, i després feu una inserció amb **<insert>** en qualsevol lloc/s que desitgeu.

Si bé aquest exemple no és massa útil en si mateix, penseu a combinar-lo amb un fitxer **<include>** `.xml`. Tingueu en compte que també podeu col·locar fragments per al fitxer `.rkh` al mateix fitxer. Simplement hauríeu de fer-hi **<include>** també del fitxer, i **<insert>** el fragment rellevant:

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

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

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

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

Similar a la inclusió en JS, l'enfocament invers és sovint encara més útil:

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

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

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

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

Finalment, també és possible **<insert>** fragments en altres fragments, sempre que: a) només hi hagi un nivell d'imbricació, i b) la secció **<snippets>** es col·loca a la part superior del fitxer (abans que s'insereixi un fragment niat); això és perquè les sentències **<insert>** es resolen de dalt a baix.

## 9.5 <include> i <snippets> vs. <embed>

A primera vista, **<include>** i **<snippets>** proporcionen una funcionalitat bastant similar a incrustar: permet reutilitzar algunes porcions de codi entre els connectors. Llavors, quina és la diferència entre aquests enfocaments i quan s'ha d'utilitzar quin?

La diferència clau entre aquests conceptes és que els connectors incrustables són paquets més petits. Combinen una IGU completa, codi per a generar codi R a partir d'això, i una pàgina d'ajuda. Per contra, incloure i inserir permet un control molt més fi de la granularitat, però a costa de menys modularitat.

És a dir, un connector que incrusta un altre connector no necessitarà saber gaire sobre els detalls interns del connector incrustat. Un exemple principal és el connector «plot_options». Els connectors que vulguin incrustar això no necessàriament necessiten conèixer totes les opcions proporcionades, o com es proporcionen. Això és una cosa bona, ja que en cas contrari un canvi en el connector «plot_options» podria fer necessari ajustar tots els connectors que incrusten això (molts). Per contra, incloure i inserir realment exposa tots els detalls interns, i els connectors que utilitzen això, per exemple, necessitaran conèixer els ID exactes i potser fins i tot el tipus dels elements utilitzats.

Per tant, la regla general és la següent: incloure i inserir són adequats si les opcions rellevants només són necessàries per a un grup clarament limitat de connectors. Els connectors incrustats són millors si el grup de connectors als quals poden ser útils no està clarament definit, i si la funcionalitat es pot fer modular fàcilment. Una altra regla general: si podeu posar les porcions comunes en un sol «fragment», llavors feu-ho i utilitzeu la incrustació. Si necessiteu molts fragments petits per a definir les porcions comunes, llavors utilitzeu **<snippets>**. Una manera final de veure-ho: si tots els connectors proporcionen funcionalitats *molt* similars, les inclusions i les insercions són probablement una bona idea. Si simplement comparteixen un o dos «mòduls», probablement la incrustació és millor.

# Capítol 10

# Conceptes per a utilitzar en connectors especialitzats

Aquest capítol conté informació sobre alguns temes que només són útils per a certes classes de connectors.

## 10.1 Connectors que produeixen un diagrama

Crear un diagrama des d'un connector és fàcil de fer. No obstant això, hi ha alguns paranys subtils que cal evitar, i també algunes funcionalitats genèriques que cal tenir en compte. Aquesta secció mostra els conceptes bàsics i conclou amb un exemple canònic que haureu de seguir sempre que creeu connectors de diagrama.

### 10.1.1 Dibuixar un diagrama a la finestra de sortida

Per a dibuixar un diagrama a la finestra de sortida, utilitzeu `rk.graph.on()` directament abans de crear el diagrama, i `rk.graph.off()`, directament després. Això és similar, p. ex., cridar `post script()` i `dev.off()` en una sessió normal de l'R.

Tanmateix, és important que *sempre* es cridi `rk.graph.off()` després de cridar `rk.graph.on()`. En cas contrari, el fitxer de sortida es deixarà en un estat trencat. Per a assegurar-vos que `rk.graph.off()` realment sigui cridat, haureu d'ajustar *totes les ordres* R entre les dues crides en l'expressió `try()`. Mai ho havíeu escoltat? No us preocupeu, és fàcil. Tot el que heu de fer és seguir el patró mostrat a l'exemple a continuació.

### 10.1.2 Afegir la funcionalitat de vista prèvia

> NOTA
> Aquesta secció analitza l'addició de funcionalitats de vista prèvia als connectors que produeixen diagrames. Hi ha seccions separades sobre previsualitzacions de sortida (HTML), previsualitzacions de dades (importades), i previsualitzacions personalitzades. No obstant això, es recomana llegir primer aquesta secció, ja que l'enfocament és similar en cada cas.

Una característica molt útil per a tots els connectors que generen un diagrama/gràfic és proporcionar una previsualització d'actualització automàtica. Per a fer-ho, necessitareu dues coses: Afegir una casella de selecció **<preview>** a la definició de la IGU, i ajustar el codi generat per a la vista prèvia.

Afegir una casella de selecció de **<preview>** és senzill. Col·loqueu el següent en algun lloc de la IGU. S'encarregarà de tota la màgia entre bastidors de crear un dispositiu de vista prèvia, actualitzar la vista prèvia sempre que la configuració hagi canviat, etc. Exemple:

> NOTA
> Des de la versió 0.6.5 del RKWard els elements de vista prèvia **<preview>** són casos especials en els diàlegs dels connectors (no assistents): es col·locaran a la columna de botons, independentment d'on estiguin exactament definits a la interfície d'usuari. Continua sent una bona idea definir-les en un lloc assenyat de la disposició, per a la compatibilitat cap endarrere.

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

I això és per la definició de la IGU.

Ajustar la plantilla JS és només una mica més de feina, aquí haureu d'assegurar-vos que només es genera el diagrama en si, i es mostra en un dispositiu en pantalla, en lloc d'anar dirigit a la sortida. És a dir, sense impressió de capçaleres, `rk.graphics.on()`, o crides similars. Per a ajudar-vos en això, el RKWard cridarà les funcions `preprocess()`, `calculate()` i `printout()` amb un paràmetre addicional que s'estableix a *`true`* en generar codi per a una vista prèvia. (El paràmetre s'omet en generar el codi final. En el javascript això avaluarà *`false`* quan s'utilitzi dins d'una sentència `if`.) Vegeu l'exemple a continuació per al patró típic que utilitzareu.

Alternativament, si necessiteu més control que aquest, podeu afegir una funció nova anomenada `preview()` a la plantilla JS, i generar el codi requerit per a una vista prèvia, allà (probablement, almenys en part, de nou cridant `calculate()`, etc.).

### 10.1.3 Opcions genèriques de diagrama

Us haureu adonat que la majoria dels connectors de traçat al RKWard proporcionen una àmplia varietat d'opcions genèriques, p. ex., per a personalitzar els títols dels eixos o els marges de les xifres. Afegir aquestes opcions al vostre connector és fàcil. Són proporcionats per un connector incrustable anomenat **rkward::plot_options**. Incrusteu això a la interfície d'usuari del connector com aquí:

```
<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>
```

Això afegirà un botó a la interfície d'usuari per a obrir una finestra amb les opcions del diagrama. La secció lògica és només un exemple. Permet un cert control sobre el connector d'opcions del diagrama. Llegiu-ne més a la pàgina d'ajuda del connector «plot_options» (enllaçada des de la pàgina d'ajuda de qualsevol connector que proporcioni les opcions genèriques).

A continuació, haureu d'assegurar-vos que el codi corresponent a les opcions del diagrama s'afegeix al codi generat per al diagrama. Per a fer-ho, obteniu les propietats **code.preprocess**, **code.printout**, i **code.calculate** des del connector incrustat d'opcions del diagrama i inseriu-les al codi tal com es mostra a l'exemple a continuació.

### 10.1.4 Un exemple canònic

Aquest és un exemple de fitxer .JS que hauríeu d'utilitzar com a plantilla, sempre que creeu un connector de traçat:

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

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

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

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

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

```
    // insert any option-setting code that should be run after the actual  ↩
        plot.
    printIndentedUnlessEmpty ('\t', getString ("plotoptions.code.calculate ↩
        "), '\n');
    echo ('})'\n);  // the closure of the try() statement

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

## 10.2 Vistes prèvies de dades, sortida i altres resultats

### 10.2.1 Vistes prèvies de sortida (HTML)

> NOTA
> Aquesta secció tracta d'afegir funcionalitats de vista prèvia als connectors que creen impressions de sortida/HTML. Es recomana que llegiu la secció separada de les previsualitzacions de diagrama abans d'aquesta secció.

Crear una vista prèvia de la sortida HTML és gairebé el mateix procediment que crear una vista prèvia del gràfic. En aquest cas, simplement assegureu-vos que **preview()** genera les ordres **rk.print()/rk.results()** pertinents. No obstant això, en general és una bona idea ometre les sentències de capçalera en la previsualització. Aquí hi ha un exemple reduït:

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

Tingueu en compte l'especificació del *mode="output"* en l'element **<preview>**.

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

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

### 10.2.2 Vistes prèvies de dades (importades)

> NOTA
> Aquesta secció analitza l'addició de funcionalitats de vista prèvia als connectors que creen (importen) dades. Es recomana que llegiu la secció separada a les previsualitzacions de diagrama, abans d'aquesta secció.

Crear una vista prèvia de les dades importades (qualsevol tipus de dades que **rk.edit()** pot gestionar), és molt similar a crear una vista prèvia de diagrama. L'exemple reduït següent hauria d'ajudar a il·lustrar com crear una vista prèvia de dades:

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

Tingueu en compte que l'element **<preview>** especifica *mode="data"* aquesta vegada. *active="true"* simplement activa la vista prèvia predeterminada.

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

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

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

De nou, la funció **preview()** genera gairebé el mateix codi R que la funció **calculate()**, de manera que creem una funció auxiliar **doCalcuate()** per a factoritzar les parts comunes. El més important a tenir en compte és que haureu d'assignar les dades importades a un objecte anomenat *preview_data* (dins de l'entorn actual: local). *Tota la resta passarà automàticament* (aproximadament, el RKWard cridarà **rk.edit(preview_data)**, embolcallat dins d'una crida a **.rk.with.window.hints()**).

NOTA
Mentre que les vistes prèvies són una característica bona, consumeixen recursos. En el cas de les previsualitzacions de dades pot haver-hi casos on les previsualitzacions poden causar problemes de rendiment significatius. Això podria ser per a la importació de conjunts de dades enormes (que són massa grans per a ser oberts per a l'edició a la finestra de l'editor del RKWard), però també es podrien importar conjunts de dades "normals", creant un gran nombre de files o columnes. *És molt recomanable que limiteu les preview_data* a una dimensió que proporciona una vista prèvia útil, sense el perill de crear problemes de rendiment notables (p. ex., 50 files per 50 columnes haurien de ser més que suficients en la majoria dels casos).

### 10.2.3 Vistes prèvies personalitzades

L'element **<preview>** es pot utilitzar per a crear vistes prèvies per a qualsevol tipus de finestra de "document" que es pot adjuntar al lloc de treball del RKWard. A més de diagrames i finestres de dades, això inclou fitxers HTML, scripts R i finestres de resum d'objectes. Per a aquests últims, haureu d'utilitzar **<preview mode="custom">**.

Si heu llegit les seccions que descriuen la vista prèvia del diagrama i les vistes prèvies de les dades, hauríeu de tenir una idea general sobre el procediment, però les vistes prèvies «personalitzades» requereixen una mica més de treball manual entre bastidors. La funció R més important que cal mirar és **rk.assign.preview.data()**, aquí. El llistat curt següent mostra com podria ser el codi R generat (previsualització) per a un connector que creï una sortida de fitxer de text:

```
## Per a ser generat en la secció de codi preview() d'un connector
        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)
        })
```

Aquí hauríeu d'obtenir el valor *SOMEID* a partir de la propietat *id* de l'element **<preview>**. P. ex., s'utilitza **getString ("preview.id")** al fitxer .js del connector.

## 10.3 Connectors dependents de context

Fins ara hem assumit que tots els connectors sempre tenen sentit, i tots es col·loquen al menú principal. No obstant això, alguns connectors només tenen sentit (o addicionalment) en un context determinat. P. ex., un connector per a exportar el contingut d'un dispositiu gràfic X11 de l'R és òbviament el més útil quan es col·loca en el menú d'un dispositiu X11, no a la barra de menús principal. A més, aquest connector hauria de conèixer el número de dispositiu en què hauria d'operar, sense haver de preguntar-ho a l'usuari.

Anomenem aquests connectors dependents del context. Per tant, en el fitxer `.pluginmap` no es col·loquen (o no només) en el **<hierarchy>** principal, sinó en un element **<context>**. Fins ara només s'admeten dos contextos diferents (després en vindran més): x11 i importació de fitxers. Ens ocuparem d'ells. Fins i tot si només esteu interessat en el context d'importació, llegiu també la secció sobre el context x11, ja que aquest és una mica més elaborat.

### 10.3.1 Context de dispositiu X11

Per a utilitzar un connector en el context d'un dispositiu x11, que se situï a la barra de menús de la finestra que obteniu quan crideu `x11()` a la consola, primer declareu-lo com de costum al fitxer `.pluginmap`:

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

No obstant això, no cal que el definiu a la jerarquia (podeu, si també té sentit com a connector de nivell superior):

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

En lloc d'això, afegiu una definició del context «x11», i afegiu-la als menús:

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

A la secció lògica de l'XML del connector, ara podeu declarar dues propietats **<external>**: *devnum* i *context*. El *context* (si es declara) s'establirà a *"x11"* quan s'invoqui el connector en aquest context. *devnum* s'establirà al número del dispositiu gràfic on operar. I això és tot.

### 10.3.2 Importar el context de les dades

Abans de llegir aquesta secció, assegureu-vos de llegir la secció context del dispositiu X11, ja que explica els conceptes bàsics.

El context *"import"* s'utilitza per a declarar els connectors del filtre de fitxers d'importació. Simplement col·loqueu-los en un context amb *id="import"* al fitxer `.pluginmap`. No obstant això, hi ha un aspecte addicional en declarar aquests connectors: per tal d'oferir un diàleg de selecció de fitxers unificat per a tots els tipus de fitxers admesos, cal declarar un bit addicional d'informació sobre el component:

```
<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>
```

La línia d'atribut simplement diu que les extensions associades del nom de fitxer per als fitxers XYZ són `*.xyz` o `*.zyx`, i que el filtre s'ha d'etiquetar amb «fitxers de dades XYZ» en el diàleg de selecció de fitxers.

Podeu declarar dues propietats **<external>** al connector. *`filename`* establirà al nom de fitxer seleccionat, i *`context`* s'establirà com *`"import"`*.

## 10.4 Consultar l'R per a obtenir informació

En alguns casos, és possible que vulgueu obtenir més informació de l'R, que es presentarà a la interfície d'usuari del vostre connector. Per exemple, podeu oferir una selecció dels nivells d'un factor que l'usuari ha seleccionat per a l'anàlisi. Des de la versió 0.6.2 del RKWard és possible fer-ho. Abans de començar, és important que tingueu present algunes advertències:

El codi R que s'executa des de dins de la lògica de la interfície d'usuari del connector s'avalua en el bucle d'esdeveniments de l'R, el que significa que es poden executar *mentre* s'estan executant altres càlculs. Això és per a assegurar-vos que la interfície d'usuari del vostre connector es pugui utilitzar, fins i tot mentre l'R estigui ocupat fent altres coses. Tanmateix, això fa que sigui molt important que el seu codi no tingui efectes secundaris. En particular:

- *No* fa cap assignació a «.GlobalEnv» o qualsevol altre entorn no local.
- *No* imprimeix res al fitxer de sortida.
- *No* traça res en la pantalla.
- En general, *no* faci res que tingui efectes secundaris. El vostre codi pot *llegir informació*, no «*fer*» altres coses.

Amb això en ment, aquí està el patró general. Ho utilitzareu dins d'una secció lògica IU amb scripts:

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

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

```
                                }

                                commandFinished = function (result, id) {
                                        if (id != last_command_id) return;  ↩
                                             // another result is about to  ↩
                                            arrive
                                        if (typeof (result) == "undefined") ↩
                                             {
                                                gui.setListValue ("selector ↩
                                                    .available", Array (" ↩
                                                    ERROR"));
                                                return;
                                        }
                                        gui.setValue ("selector.enabled",  ↩
                                            1);
                                        gui.setListValue ("selector. ↩
                                            available", result);
                                }
                ]]></script>
```

Aquí, *variable* és una propietat que conté un nom d'objecte (p. ex., dins d'un **<varslot>**). Sempre que això canviï, voldreu actualitzar la visualització dels nivells dins del **<valueselector>**, anomenat *selector*. La funció clau aquí és **doRCommand()**, que pren com a primer paràmetre la cadena d'ordre a executar, i com a segon paràmetre el nom d'una funció a cridar, quan l'ordre hagi finalitzat. Tingueu en compte que l'ordre s'està executant asíncronament, i això fa les coses una mica més complexes. Per a una cosa que voleu estar segur, el **<valueselector>** roman desactivat, mentre que no contingui informació actualitzada. Una altra cosa és que podríeu haver posat a la cua més d'una ordre, abans d'obtenir els primers resultats. Aquesta és la raó per la qual cada ordre té una "id", i la guardem en *last_command_id* per a una referència posterior.

Quan es fa l'ordre, es crida la crida de retorn especificada (*commandFinished*, en aquest cas) amb dos paràmetres: el resultat en si, i l'identificador de l'ordre corresponent. El resultat serà d'un tipus similar a la representació en R, és a dir, una matriu numèrica, si el resultat és numèric, etc. Fins i tot pot ser una **list()** de l'R, però en aquest cas es representarà com un **Array()** JS sense noms.

Cal tenir en compte que fins i tot aquest exemple és una mica simplificat. En realitat, haureu de prendre precaucions addicionals, p. ex., per a evitar posar una quantitat extrema de nivells al selector. La bona notícia és que probablement no cal fer tot això vós mateix. L'exemple anterior es pren del connector **rkward::level_select**, per exemple, que simplement podeu incrustar en el vostre propi connector. Fins i tot us permet especificar una expressió diferent per a executar en lloc de **levels()**.

## 10.5 Referenciar l'objecte actual o el fitxer actual

Per a molts connectors és desitjable treballar en l'objecte «actual». Per exemple, un connector «d'ordenació» podria preseleccionar el «data.frame» que s'està editant actualment per a l'ordenació. El nom de l'objecte actual està disponible per als connectors com una propietat predefinida anomenada *current_object*. Podeu connectar-vos a aquesta propietat de la manera habitual. Si no hi ha cap objecte actual, la propietat equival a una cadena buida. De la mateixa manera, l'URL del fitxer de script actual és accessible com una propietat predefinida anomenada *current_filename*. Aquesta propietat està buida si no s'està editant cap fitxer de script, o el fitxer de script encara no s'ha desat.

Actualment, el *current_object* només pot ser de classe `data.frame`, però no confieu en això, ja que això s'ampliarà a altres tipus de dades en el futur. Si només teniu interès en els objectes `data.frame`, connecteu amb la propietat *current_dataframe* en el seu lloc. Alternativament, podeu forçar els requisits de tipus utilitzant restriccions apropiades a **<varslot>**, o utilitzant la creació de scripts lògics d'IGU.

## 10.6 Repetir (un conjunt d') opcions

A vegades voleu repetir un conjunt d'opcions per a un nombre arbitrari d'elements. Per exemple, suposeu que voleu implementar un connector per a ordenar un «data.frame». És possible que vulgueu permetre l'ordenació per un nombre arbitrari de columnes (en cas d'enllaços entre les primeres columnes). Això es podria realitzar simplement permetent a l'usuari seleccionar diverses variables en un **<varslot>** amb *multi="true"*. Però si voleu ampliar-ho, p. ex., permetent a l'usuari especificar per a cada variable si s'ha de convertir a caràcter/numèric, o si l'ordenació ha de ser ascendent o descendent, necessitareu més flexibilitat. Altres exemples serien dibuixar diverses línies en un diagrama (permetent seleccionar objecte, estil de línia, color de línia, etc. per a cada línia), o especificar un mapatge per a la recodificació des d'un conjunt de valors antics a valors nous.

Introduïu l'**<optionset>**. Mirem un exemple senzill, primer:

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

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

Aquí, hem creat una interfície d'usuari per a especificar un nombre de persones (p. ex., autors). La interfície d'usuari requereix almenys una entrada (*min_rows="1"*). Dins de l'element **<optionset>**, comencem especificant el **<content>**, és a dir, els elements que pertanyen al conjunt d'opcions. Estareu familiaritzat amb la majoria d'elements dins del **<content>**.

A continuació, especifiqueu les variables d'interès que voldrem llegir des de l'opció establerta al nostre fitxer JS. Com que ens ocuparem d'un nombre arbitrari d'articles, no podem llegir només `getString ("firstname")` en JS. Més aviat, per a cada valor d'interès, especifiqueu un **<optioncolumn>**. Per a la primera «optioncolumn» a l'exemple, **<connect="firstname.text">** significa que el contingut de l'element **<input>** "firstname" es llegeix per a cada element. Les **<optioncolumn>** per a la qual es proporciona una *label*, es mostrarà a la pantalla, en una columna amb aquesta etiqueta. Al JS, ara podem obtenir els noms de tots els autors utilitzant `getList("set.firstname")`, `getList("set.lastnames")` pels cognoms, i `getList("set.gender")` per a una matriu de cadenes "m"/"f".

Tingueu en compte que no hi ha restriccions sobre el que podeu col·locar dins d'un **<optionset>**. Fins i tot podeu utilitzar components incrustats. Igual que amb qualsevol altre element, tot el

que heu de fer és recollir les variables de sortida d'interès en una especificació **<optioncolumn>**. En el cas dels connectors incrustats, aquesta és sovint una secció de la propietat "code". P. ex.:

```
<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>
```

Per descomptat, també podeu utilitzar la lògica UI dins d'un «optionset». Hi ha dues opcions per a fer això: podeu fer-ho fent la connexió (o creació de scripts) a la secció principal **<logic>** del connector, com és habitual. No obstant això, accedireu als elements de la IU a la regió de continguts com (p. ex.) «set.contents.firstname.XYZ». Tingueu en compte el prefix "set" (l'`id` que heu assignat al conjunt i "contents"). Alternativament, podeu afegir una secció **<logic>** separada com a element fill de l'**<optionset>**. En aquest cas, els `id` s'adreçaran en relació amb la regió de continguts, p. ex., "firstname.XYZ". Només l'element **<script>** no està permès a la secció lògica d'un «optionset». Si voleu utilitzar la creació de scripts, haureu d'utilitzar la secció principal **<logic>** del connector.

> NOTA
> Quan la lògica de creació de scripts en un «optionset», tot el que podeu fer és accedir a la regió de contingut *actual*. Per tant, normalment, només té sentit connectar elements dins de la regió de contingut entre ells. Connectar una propietat fora de l'**<optionset>** a una propietat dins de la regió de contingut, pot ser útil per a la inicialització. No obstant això, modificar la regió de contingut després de la inicialització *no* s'aplicarà als elements que l'usuari ja ha definit. Només a l'element seleccionat actualment en el conjunt.

### 10.6.1 «Driven» «optionsets»

Fins ara hem considerat un **<optionset>** que proporciona botons per a afegir/eliminar elements. No obstant això, en alguns casos, és molt més natural seleccionar elements fora de l'**<optionset>**, i proporcionar només opcions per a personalitzar alguns aspectes de cada element en un **<optionset>**. Per exemple, suposeu que voleu permetre a l'usuari traçar diversos objectes dins d'un diagrama. Per a cada objecte, l'usuari hauria de poder especificar el color de la línia. *Podeu* resoldre-ho col·locant un **<varselector>** i **<varslot>** dins de l'àrea **<content>**, permetent a l'usuari seleccionar un element alhora. Tanmateix, significarà menys clics per a l'usuari, si en el seu lloc utilitzeu un **<varslot multi="true">** *fora* de l'**<optionset>**. A continuació, connectareu aquesta selecció d'objectes a un «optionset» anomenat «driven». Així és com es fa:

```
<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>
```

Començarem a veure l'exemple a la part inferior. Tindreu en compte que dues especificacions de **<optioncolumn>** tenen *external="true"*. Això li indica al RKWard que estan controlats des de fora de l'**<optionset>**. Aquí, l'únic motiu de l'opció «optioncolumn» «varnames» és proporcionar etiquetes fàcils de llegir a la pantalla de l'«optionset» (està connectat al modificador «shortname» de la propietat que conté els objectes seleccionats). El motiu de «optioncolumn» «vars» és servir com a columna «key», tal com especifica **<optionset keycolumn="vars"...>**. Això vol dir que per a cada entrada en aquesta llista, el conjunt oferirà un conjunt d'opcions, i les opcions estan lligades lògicament a aquestes entrades. Aquesta columna està connectada a la propietat que conté els objectes seleccionats a **<varslot>**. Això és per a cada objecte que hi ha seleccionat, l'**<optionset>** permetrà especificar el color de la línia.

> NOTA
> La columna externa també pot ser connectada amb *connect* a les propietats dins de la regió **<content>**. No obstant això, és important tenir en compte que les «optioncolumn» declarades *external="true"* mai no s'han de modificar des de dins de l'**<optionset>**, i les «optioncolumn» declarades *external="false"* (predeterminat) mai no s'han de modificar des de fora de l'**<optionset>**.

### 10.6.2 Alternatives: quan no s'usen els «optionsets»

Els «optionset» són una eina potent, però de vegades poden fer més mal que bé, ja que afegeixen una complexitat considerable, tant des de la perspectiva d'un desenvolupador de connectors, com des de la perspectiva d'un usuari. Per tant, penseu dues vegades, quan les utilitzeu. Aquí teniu un consell:

- Per alguns casos simples, l'element **<matrix>** pot proporcionar una alternativa útil més senzilla.
- No feu que la vostra extensió faci massa. Hem donat l'exemple d'utilitzar un «optionset» per a un connector per a dibuixar diverses línies dins d'un diagrama. Però en general no és una bona idea crear un connector que produeixi diagrames individuals per a cada element en un «optionset». Més aviat, feu que el connector produeixi un diagrama, i l'usuari el pot cridar diverses vegades.
- Si no espereu més de dos o tres elements en un conjunt, considereu repetir les opcions manualment.

# Capítol 11

# Gestió de dependències i problemes de compatibilitat

## 11.1 Compatibilitat de la versió del RKWard

Fem tot el possible per a assegurar-nos que els connectors desenvolupats per a una versió antiga del RKWard romandran funcionals en versions posteriors del RKWard. No obstant això, el contrari no sempre és cert, ja que s'han afegit característiques noves. Com que no tots els usuaris estan executant l'última versió del RKWard, això vol dir que el vostre connector podria no funcionar per a tothom.

Quan tingueu coneixement d'aquests problemes de compatibilitat, haureu d'assegurar-vos de documentar aquest fet en el fitxer `.pluginmap`, utilitzant l'element **<dependencies>**. Les **<dependencies>** es poden especificar com un fill directe de l'element <document> del `.pluginmap`, o com a element fill de definicions de **<component>** individual. En el primer cas, les dependències s'apliquen a *tots els connectors* del mapa. En l'últim cas només per al **<component>** individual. També podeu barrejar dependències ″global″ i ″specific″. En aquest cas, les dependències ″globals″ s'afegeixen a les del component individual.

Mirem un petit exemple:

```
<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>
```

En aquest exemple, se sap que tots els connectors requereixen almenys la versió 0.5.0c del RKWard. Un connector, amb *id=″myplugin″* es proporciona en dues variants alternatives. La primera versió, reduïda, s'utilitzarà per a les versions del RKWard abans de la 0.6.1. L'últim utilitza característiques que són noves en el RKWard 0.6.1, i només s'utilitzaran a partir del RKWard 0.6.1 i posteriors.

Proporcionar variants alternatives com aquesta és una manera molt fàcil d'usar per a fer ús de característiques noves, tot i que encara manté el suport per a versions anteriors del RKWard. Les versions alternatives haurien de compartir el mateix `id` (en cas contrari es produiran avisos), i només es poden definir *dins del mateix fitxer* `.pluginmap`.

El connector que no és compatible amb la versió en execució del RKWard, i que no ve amb una versió alternativa s'ignorarà amb un avís.

> NOTA
> En realitat, el RKWard 0.6.1 és la primera versió per a interpretar les dependències i per a informar dels errors de dependències. Per tant, contràriament al que l'exemple pot suggerir, especificar versions anteriors en les dependències no tindrà cap efecte directe (però pot ser una bona idea per a propòsits de documentació).

*De vegades* fins i tot serà possible gestionar els problemes d'incompatibilitat de versions *dins* d'un únic fitxer `.pluginmap`, utilitzant l'element **<dependency_check>**, descrit a la secció següent.

## 11.2 Compatibilitat de la versió de l'R

Similar a `rkward_min_version` i `rkward_max_version`, l'element **<dependencies>** permet l'especificació dels atributs `R_min_version` i `R_max_version`. No obstant això, hi ha les diferències següents:

- Els connectors que no compleixen el requisit de la versió de l'R *no* s'ometen actualment en llegir un fitxer `.pluginmap`. L'usuari encara pot cridar al connector, i no veurà cap avís immediat (en versions futures, probablement es mostrarà un missatge d'avís)
- En conseqüència, *no* és possible definir versions alternatives d'un connector depenent de la versió en execució de l'R.
- No obstant això, sovint és fàcil aconseguir compatibilitat cap enrere com es mostra a continuació. Si esteu al corrent dels problemes de compatibilitat de l'R, considereu utilitzar aquest mètode, en lloc de definir una dependència d'una versió particular de l'R.

En molts casos, és possible proporcionar fàcilment una funcionalitat reduïda, si una característica determinada no està disponible en la versió en execució de l'R. Considereu l'exemple curt següent d'un fitxer `.xml`:

```
<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>
```

En aquest exemple, l'opció de compressió ″xz″ simplement es desactivarà quan la versió R en temps d'execució sigui anterior a 2.10.0 (que no admet la compressió «xz»). L'element **<dependency_check>** admet els mateixos atributs que l'element **<dependencies>** en fitxers `.pluginmap`. Crea una propietat booleana, el qual és cert, si es compleixen les dependències especificades, fals en cas contrari.

## 11.3 Dependències de paquets de l'R

Es poden definir dependències en paquets específics de l'R, però a partir del RKWard 0.6.1, aquestes dependències no es marquen, ni s'instal·len/es carreguen automàticament. No obstant això, es mostren en els fitxers d'ajuda del connector. Aquesta és una definició d'exemple:

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

> NOTA
> Assegureu-vos sempre d'afegir les crides `require()` apropiades, si el connector necessita que es carreguin determinats paquets.

> NOTA
> Si distribuïu el vostre `.pluginmap` com un paquet R, i tots els connectors depenen d'un paquet en particular, aleshores hauríeu de definir aquesta dependència a nivell del paquet R. Definir les dependències als paquets de l'R a nivell del mapa de connectors del RKWard és molt útil, si només alguns dels vostres connectors necessiten la dependència, la dependència no està disponible al CRAN, o el vostre `.pluginmap` no es distribueix com un paquet de l'R.

## 11.4 Dependències d'altres RKWard `.pluginmaps`

Si els vostres connectors depenen dels connectors definits en un altre `.pluginmap` (és a dir, *no* són part del paquet) podreu definir aquesta dependència així:

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

Actualment, no carregarà, ni instal·larà, ni tan sols avisarà sobre els `.pluginmap` que manquen, però almenys es mostrarà informació sobre les dependències (i on obtenir-les) a la pàgina d'ajuda del connector. No cal (i no hauríeu de) declarar dependències en els `.pluginmap` que es distribueixen amb la distribució oficial del RKWard, o en els `.pluginmap` que es troben dins del vostre propi paquet. A més, si el `.pluginmap` requerit és distribuït com un paquet R, declareu una dependència del paquet (com es mostra a la secció anterior), en lloc del mapa.

Per a assegurar-vos que els connectors requerits es carreguen realment, utilitzeu l'etiqueta **<require>** (vegeu la referència per a més detalls).

## 11.5 Un exemple

Per a aclarir com es poden barrejar les definicions de dependències, aquí hi ha un exemple combinat:

```
<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>
```

# Capítol 12

# Traduccions d'un connector

Fins ara hem utilitzat alguns conceptes relatius a les traduccions o «i18n» (abreviatura d'«internacionalització», que té 18 caràcters entre la i i la n) de passada. En aquest capítol donarem una explicació més en profunditat del funcionalment de l'«i18n» per als connectors del RKWard. Per a la majoria, *no* necessitareu tot això als connectors. No obstant això, pot ser una bona idea llegir aquest capítol íntegrament, ja que entendre aquests conceptes hauria d'ajudar-vos a crear connectors que siguin completament traduïbles, i que permetin una alta qualitat de les traduccions.

## 12.1 Consideracions generals

Un punt important per a entendre les traduccions de programari, en contrast amb les traduccions d'altres materials de text, és que els traductors sovint tindran un temps força llarg per a fer-se una imatge completa de *què* estan traduint. Les traduccions de programari es basen necessàriament en fragments de text força curts: cada etiqueta que doneu a una **<option>** en un **<radio>**, cada cadena que marqueu per a la traducció en una crida de funció **i18n()**, formarà una «unitat de traducció» separada. En essència, cada fragment es presentarà al traductor de forma aïllada. Bé, no un aïllament complet, ja que intentem proporcionar al traductor tant context significatiu com es pugui extreure automàticament. Però en alguns punts els traductors necessitaran un context addicional per a donar sentit a una cadena, especialment quan les cadenes siguin curtes.

## 12.2 «i18n» als fitxers «xml» del RKWard

L'«i18n» funcionarà per als fitxers XML del RKWard. Si esteu escrivint el vostre propi **.pluginmap** (p. ex., per a un connector extern), haureu d'especificar un `po_id` al costat del `id` del «pluginmap». Això defineix el "catàleg de missatges" a utilitzar. En general, això s'ha d'establir igual que el `id` del vostre **.pluginmap**, però si proporcioneu diversos **.pluginmap** relacionats en un sol paquet, probablement voldreu especificar un `po_id` comú als vostres mapes. El fitxer `po_id` d'un fitxer **.pluginmap** és heretat per tots els connectors declarats en ell, llevat que això declari un altre `po_id`.

Per a connectors i pàgines d'ajuda, no cal que li indiqueu al RKWard quines cadenes s'han de traduir, perquè generalment això és evident a partir del seu ús. No obstant això, tal com s'ha explicat anteriorment, cal tenir en compte les cadenes que poden ser ambigües o necessiten alguna explicació per a ser traduïdes correctament. Per a les cadenes que poden tenir significats diferents, proporcioneu un `i18n_context` com aquest:

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

Proporcionar un *i18n_context* farà que les dues cadenes es tradueixin per separat. En cas contrari, compartirien una única traducció. A més, el context es mostra al traductor. L'atribut *i18n_context* és compatible amb tots els elements que poden tenir cadenes traduïbles, en algun lloc, inclosos els elements que contenen text dins d'ells (p. ex. els elements **<text>**).

En altres casos, la cadena a traduir té un significat únic no-ambigu, però encara pot necessitar alguna explicació. En aquest cas podeu afegir un comentari que es mostrarà als traductors. Alguns exemples poden incloure:

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

Tingueu en compte que aquests comentaris han de precedir l'element al qual s'apliquen, i han de començar amb "i18n:" o "TRANSLATORS:".

Finalment, en casos rars, és possible que vulgueu excloure determinades cadenes de la traducció. Això pot tenir sentit, per exemple, si oferiu una elecció entre diversos noms de funcions R en un control **<radio>**. Llavors no voleu que es tradueixin, però depenent del context, hauríeu de considerar donar una etiqueta descriptiva, en el seu lloc:

```
<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>
```

Tingueu en compte que ometreu l'atribut *label*, llavors, i especifiqueu *noi18n_label*, en el seu lloc. A més, cal tenir en compte que en contrast amb *i18n_context* i els comentaris, utilitzant *noi18n_label* el vostre connector serà incompatible amb les versions del RKWard anteriors a la 0.6.3.

## 12.3 «i18n» als fitxers i seccions dels fitxers «js» del RKWard

A diferència dels fitxers `.xml`, fer que els fitxers `.js` d'un connector siguin traduïbles requereix més feina personalitzada. La diferència clau, aquí, és que no hi ha cap manera decent automàtica de saber, si una cadena està pensada per a ser mostrada com una cadena llegible per humans, o un tros de codi. Així que heu de marcar-ho vós mateix. Ja hem donat exemples d'això, tot el temps. Aquí hi ha una descripció més completa de les funcions «i18n» disponibles en el codi js, i alguns consells per a casos més complexos:

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

La funció més important. Marca la cadena a traduir. La cadena (traduïda o no) es retorna entre cometes amb cometes dobles («»). Es pot utilitzar un nombre arbitrari de variables de substitució a la cadena com es mostra a continuació. L'ús d'aquestes variables de substitució en lloc de concatenar petites subcadenes és molt més fàcil per als traductors:

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

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

Igual que **i18n()**, però a més proporciona un context de missatge:

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

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

Igual que **i18n()**, però per a missatges que poden ser diferents en forma singular o plural (i alguns idiomes tenen encara més formes numèriques diferenciades). Tingueu en compte que igual que amb **i18n()**, podeu utilitzar un nombre arbitrari de reemplaçaments, però es requereix el primer («%1»), i ha de ser un enter.

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

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

**i18np()** amb context de missatge afegit.

**comment (comentari, [sagnat])**

Fa un comentari de codi, marcat per a la traducció. A diferència de les altres funcions i18n(), això no està entre cometes, però s'afegeix un «#» a cada línia del comentari.

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

Per a afegir comentaris als traductors (vegeu a dalt un debat de les diferències entre el comentari i el context), afegeix un comentari que comenci per ″i18n:″ o ″translators:″ directament per sobre de la crida **i18n()** a comentar. P. ex.:

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

### 12.3.1 «i18n» i cometes

En gran part, no us haureu de preocupar pel comportament de l'i18n() respecte a les cometes. Com que, normalment, les cadenes traduïbles són literals de cadenes, citar-les és el correcte i estalvia una mica d'escriptura. A més, en funcions com **makeHeaderCode()/Header()** que solen citar els seus arguments, les cadenes i18n() estan protegides de cites duplicades. Essencialment, això funciona, enviant primer la cadena traduïda a través de **quote()** (per a fer-la citada), després a través de **noquote()** (per a protegir-la de les cites addicionals). Si necessiteu una cadena traduïble que no estigui entre cometes, utilitzeu **i18n(noquote (″El meu missatge″))**. Si necessiteu una cadena traduïble per a ser citada, una segona vegada, envieu-la a través de **quote()**, *dues vegades*.

Dit això, generalment no és una bona idea fer parts com els noms de funcions o els noms de variables siguin traduïbles. Per una cosa, R, el llenguatge de programació, és inherent en anglès, i no hi ha internacionalització del llenguatge en si. Els comentaris de codi són un tema diferent, però hauríeu d'utilitzar la funció **comment()** per a aquests. En segon lloc, fer que les parts sintàcticament rellevants del codi generat siguin traduïbles significa que les traduccions podrien trencar el vostre connector. Per exemple, si un traductor confiat tradueix una cadena que vol dir un nom de variable en dues paraules diferents amb un espai entremig.

## 12.4 Manteniment d'una traducció

Ara que heu fet que el vostre connector sigui traduïble, com la traduïu realment? En general, només us heu de preocupar d'això quan desenvolupeu un connector extern. Per als connectors en el repositori principal del RKWard, es fa tota la màgia. Aquest és el flux de treball bàsic per a connectors externs. Tingueu en compte que necessiteu les eines «gettext» instal·lades:

- Marqueu totes les cadenes, proporcionant el context i els comentaris segons sigui necessari
- Executeu **python3 scripts/update_plugin_messages.py --extract-only /path/to/my.pluginmap**. scripts/update_plugin_messages.py actualment no forma part de les versions de codi font, però es pot trobar en una extracció del repositori de codi font.
- Distribuïu el fitxer **rkward__*POID*.pot** als vostres traductors. Per a connectors externs, es recomana col·locar-lo en una subcarpeta "po" a inst/rkward.
- El traductor obre el fitxer en una eina de traducció com el **lokalize**. En realitat, encara que no prepareu cap traducció, cal provar aquest pas per vós mateix. Navegueu per les cadenes extretes buscant problemes/ambigüitats.
- El traductor desa la traducció com a **rkward__*POID*.*xx*.po** (on *xx* és el codi de llengua), i us l'envia de retorn.
- Copieu **rkward__*POID*.*xx*.po** al vostre codi font, al costat de **rkward__*POID*.pot**. Executeu **python3 scripts/update_plugin_messages.py /path/to/my.pluginmap** (Nota: aquesta vegada sense *--extract-only*). Això fusionarà la traducció amb qualsevol canvi de cadena provisional, compilarà la traducció i l'instal·larà a ***DIR_OF_PLUGINMAP*/po/*xx*/LC_MESSAGES/rkward__*POID*.mo** (on *xx* és el codi de llengua, de nou).
- També hauríeu d'incloure la traducció no compilada (és a dir, **rkward__*POID*.*xx*.po**) a la vostra distribució, al subdirectori «po».
- Per a qualsevol actualització del connector, executeu **python3 scripts/update_plugin_messages.py /path/to/my.pluginmap** per a actualitzar el fitxer .pot, però també els fitxers .po existents i els catàlegs de missatges compilats.

## 12.5 Escriure traduccions d'un connector

Suposem que coneixeu el vostre ofici com a traductor, o que esteu disposat a estudiar-lo en altres llocs. Unes poques paraules específicament sobre les traduccions dels connectors del RKWard, però:

- Els connectors del RKWard no es poden traduir fins a la versió 0.6.3, i abans no s'havien escrit pensant en l'«i18n». Per tant, trobareu cadenes força ambigües i altres problemes d'«i18n» respecte a altres projectes madurs. No treballeu en silenci per a solucionar-ho, però feu-nos-ho saber (o als mantenidors dels connectors), de manera que puguem solucionar aquests problemes.
- Molts connectors del RKWard es refereixen a termes molt especialitzats, des de la gestió de dades i estadístiques, però també d'altres camps de la ciència. En molts casos, una bona traducció requerirà almenys un coneixement bàsic d'aquests camps. En alguns casos, no *hi ha* una traducció bona per a un terme tècnic, i la millor opció pot ser deixar el terme sense traduir, o incloure el terme anglès entre parèntesis. No us centreu massa a arribar al 100% de les cadenes traduïdes, centreu-vos a proporcionar una bona traducció, fins i tot si això significa ometre algunes cadenes (o fins i tot ometre alguns catàlegs de missatges en el seu conjunt). Altres usuaris poden omplir qualsevol buit en els termes tècnics.

# Capítol 13

# Autor, llicència i informació de la versió

Així que heu escrit un conjunt de connectors, i us esteu preparant per a compartir el vostre treball. Per a assegurar-vos que els usuaris sàpiguen de què tracta el vostre treball, amb quins termes poden utilitzar-lo i distribuir-lo, i a qui han de contactar sobre problemes o idees, heu d'afegir informació *sobre* els vostres connectors. Això es pot fer utilitzant l'element **<about>**. Es pot utilitzar en el `.pluginmap` o en els fitxers `.xml` d'un connector individual (en ambdós casos com a fill directe de l'etiqueta «document»). Quan s'especifica en el `.pluginmap` s'aplicarà a tots els connectors. Si s'especifica **<about>** en ambdós llocs, la informació **<about>** del fitxer `.xml` del connector substituirà la del fitxer `.pluginmap`. També podeu afegir un element **<about>** a les pàgines .rkh, que no estan connectades a cap connector, si cal.

Aquest és un exemple del fitxer `.pluginmap` amb només algunes explicacions, a continuació. En cas de dubte, es pot disposar de més informació a la referència.

```
<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>
```

La majoria d'això s'explica per si mateix, de manera que no debatrem cada element de l'etiqueta. Però mirem alguns detalls que probablement necessiten un comentari per a una comprensió més fàcil.

L'element *category* a **<about>** es pot definir de forma bastant lliure, però hauria de ser significatiu, ja que es creu que s'utilitza per a ordenar connectors en grups. Tots els altres atributs d'aquesta etiqueta d'obertura són obligatoris i s'han d'omplir amb contingut raonable.

També s'ha d'indicar almenys un **<author>** amb una adreça de correu electrònic vàlida i també s'ha de donar el rol 'aut' ('author'). En cas que el vostre connector causi problemes o algú vulgui compartir la seva gratitud amb vosaltres, hauria de ser fàcil contactar algú que hi estigui implicat. Per a més informació sobre altres rols vàlids, com 'ctb' per als col·laboradors de codi o 'cre' per al manteniment de paquets, consulteu la documentació de l'R sobre person().

NOTA

Recordeu que podeu utilitzar **<include>** i **<insert>** per a repetir informació a través de diversos fitxers `.xml` (p. ex., informació sobre un autor que estava implicat en diversos connectors). Més informació.

SUGGERIMENT

No heu d'escriure aquest codi XML a mà. Si utilitzeu la funció `rk.plugin.skeleton()` des del paquet rkwarddev i proporcioneu tota la informació necessària a través de l'opció *about*, es crearà automàticament un fitxer `.pluginmap` amb una secció <about> funcional.

# Capítol 14

# Compartiu el vostre treball amb altres persones

## 14.1 Connectors externs

A partir de la versió 0.5.5, el RKWard proporciona una manera còmoda d'instal·lar connectors addicionals de tercers que no pertanyen al paquet principal. Anomenem aquests «connectors externs». Venen en forma d'un paquet R i es poden gestionar directament a través de les característiques habituals de gestió de paquets de l'R o el RKWard.

Aquesta secció de la documentació descriu com s'han d'empaquetar els connectors externs de manera que el RKWard els pugui utilitzar. La creació del connector en si és idèntica a les seccions anteriors. És a dir, probablement hauríeu d'escriure primer un connector funcional i després tornar aquí per a aprendre a distribuir-lo.

Com que els connectors externs són una característica relativament jove, els detalls d'això probablement canviaran en futures versions. Us convidem a contribuir amb les vostres idees per a millorar el procés.

> SUGGERIMENT
> Aquests documents expliquen els detalls dels connectors externs perquè pugueu aprendre com funcionen. A més a més d'això, cal mirar el paquet rkwarddev, dissenyat per a automatitzar gran part del procés d'escriptura.

## 14.2 Per què connectors externs?

El nombre de paquets per a ampliar la funcionalitat de l'R ja és immens, i està pujant. D'una banda, volem animar-vos a escriure connectors fins i tot per a les tasques més especialitzades que necessiteu resoldre. D'altra banda, l'usuari mitjà no hauria de perdre's en grans arbres de menús plens de termes estadístics desconeguts. Per tant, també semblava raonable mantenir la gestió dels connectors en el RKWard bastant modular. L'equip del RKWard manté el seu repositori propi de paquets públics a https://files.kde.org/rkward/R/ , designat per a allotjar els vostres connectors externs.

Com a regla general, els connectors que semblen servir a un propòsit àmpliament utilitzat (p. ex., t-Tests) haurien de formar part del paquet central, mentre que els que serveixen a un grup bastant limitat de persones amb interessos especials haurien de proporcionar-se com un paquet opcional. Per a vós, com a autor d'un connector, la millor pràctica és començar amb un connector extern.

## 14.3 Estructura d'un paquet de connector

Perquè els connectors externs s'instal·lin i funcionin correctament han de seguir algunes directrius estructurals pel que fa a la seva jerarquia de fitxers.

### 14.3.1 Jerarquia de fitxers

Donem un cop d'ull a la jerarquia de fitxers prototípica d'un arxiu de connectors elaborat. No heu d'incloure tots aquests directoris i/o fitxers perquè funcioni un connector (llegiu per a saber què és absolutament necessari), considereu això un exemple de «millor pràctica»:

```
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
```

> **NOTA**
> En aquest exemple, tots els casos de plugin_name, testsuite_name i sometest_name s'han de substituir pels seus noms correctes, d'acord amb això. A més, ll és un marcador de posició per a una abreviatura de l'idioma (p. ex., 'de', 'en' o 'es').

> **SUGGERIMENT**
> No heu de crear aquesta jerarquia de fitxers a mà. Si utilitzeu la funció `rk.plugin.skeleton()` del paquet rkwarddev, es crearan automàticament tots els fitxers i directoris necessaris, excepte el directori `po` que es crea i gestiona amb l'script de traducció.

#### 14.3.1.1 Components bàsics del connector

És obligatori incloure almenys tres fitxers: un `.pluginmap`, una descripció .xml del connector i un fitxer .js del connector. És a dir, fins i tot el directori "plugins" és opcional. Pot ajudar a donar una mica d'ordre als vostres fitxers, especialment si incloeu més d'un connector/diàleg a l'arxiu, que no és cap problema per descomptat. Podeu tenir tants directoris per als fitxers de connectors reals com creieu oportú, només s'han d'assemblar al `.pluginmap`, respectivament. També és possible incloure diversos fitxers `.pluginmap`, si s'adapta a les vostres necessitats, però llavors hauríeu d'incloure'ls tots a 'plugin_name.pluginmap'.

Cada paquet R ha de tenir un fitxer `DESCRIPTION` vàlid, el qual també és crucial per al RKWard reconeixent-lo com a proveïdor de connectors. La major part de la informació que porta també és necessària en el connector Meta-information i possiblement dependencies, però en un format diferent (la documentació de l'R explica el fitxer DESCRIPTION en detall).

A més del contingut general d'un fitxer `DESCRIPTION`, assegureu-vos també d'incloure la línia 'Enhances: rkward'. Això farà que el RKWard escanegi automàticament el paquet cercant connectors si està instal·lat. Un exemple de fitxer `DESCRIPTION` té aquest aspecte:

```
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"))
```

> **SUGGERIMENT**
> No heu d'escriure aquest fitxer a mà. Si utilitzeu la funció `rk.plugin.skeleton()` del paquet rkward-dev i proporcioneu tota la informació necessària a través de l'opció 'about', es crearà automàticament un fitxer `DESCRIPTION` que funciona.

#### 14.3.1.2 Informació addicional (opcional)

`ChangeLog`, `README`, `AUTHORS`, `LICENSE` haurien de ser autoexplicatius i són totalment opcionals. En realitat, no seran interpretats pel RKWard, de manera que estan més aviat destinats a portar informació addicional que podria ser rellevant, p. ex., per als distribuïdors. La majoria del seu contingut rellevant (crèdits d'autor, termes de llicència, etc.) s'inclourà de totes maneres en els fitxers dels connectors reals (vegeu la secció sobre metainformació). Tingueu en compte que tots aquests fitxers també es poden col·locar en algun lloc del directori "inst", si voleu que no només estiguin presents a l'arxiu d'origen, sinó també al paquet instal·lat.

#### 14.3.1.3 Proves automatitzades de connectors (opcional)

Un altre directori opcional és "tests", que està destinat a proporcionar els fitxers necessaris per a proves automatitzades de connectors. Aquestes proves són útils per a comprovar ràpidament si els connectors encara funcionen amb versions noves de l'R o del RKWard. Si voleu incloure proves, realment hauríeu de restringir-vos a l'esquema de noms i jerarquia que es mostra aquí. És a dir, les proves han de residir en un directori anomenat `tests`, que inclou un fitxer `testsuite.R` i una carpeta amb estàndards de proves anomenats després del conjunt de proves apropiat. No obstant això, podeu proporcionar més d'un conjunt de proves; en aquest cas, si no voleu afegir-les totes en el conjunt de proves `testsuite.R`, podeu dividir-los, p. ex., un fitxer per a cada paquet de proves i crear un `testsuite.R` amb crides `source()` a cada fitxer de la suite. En qualsevol cas, creeu subdirectoris separats amb estàndards de prova per a cada suite definida.

Els avantatges de mantenir aquesta estructura és que les proves dels connectors es poden executar simplement cridant `rktests.makplugintests()` del paquet rkwardtests sense arguments addicionals. Consulteu la documentació en línia de Proves de connectors automatitzades per a més detalls.

## 14.4 Construcció del paquet del connector

Com s'ha explicat anteriorment, els connectors externs del RKWard són, en efecte, paquets R i, per tant, el procés d'empaquetatge és idèntic. En contrast amb els paquets R «reals», un paquet de connector pur no porta cap codi R addicional (tot i que per descomptat podeu afegir connectors del RKWard als paquets R habituals, utilitzant els mateixos mètodes que s'expliquen aquí). Això hauria de fer encara més fàcil crear un paquet operatiu, sempre que tingueu un fitxer `DESCRIPTION` vàlid i s'adhereixi a la jerarquia de fitxers explicada en les seccions anteriors.

La manera més fàcil de construir i provar el vostre connector és utilitzar l'ordre R a la línia d'ordres, per exemple:

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

> SUGGERIMENT
> No cal construir el paquet com aquest a la línia d'ordres. Si utilitzeu la funció `rk.build.package()` del paquet rkwarddev, es construirà i/o comprovarà el paquet de connectors.

# Capítol 15

# Desenvolupament de connectors amb el paquet rkwarddev

## 15.1 Vista general

Escriure connectors externs implica escriure fitxers en tres llenguatges (XML, JavaScript i R) i la creació d'una jerarquia estandarditzada de directoris. Per a fer-ho molt més fàcil als desenvolupadors voluntaris de connectors, estem proporcionant el paquet rkwarddev. Proporciona una sèrie de funcions simples de l'R per a crear el codi XML per a tots els elements de diàleg com els «tabbooks», caselles de selecció, llistes desplegables o navegadors de fitxers, així com funcions per a crear el codi JavaScript i fitxers d'ajuda del RKWard per a començar. La funció `rk.plugin.skeleton()` crea l'arbre de directoris esperat i tots els fitxers necessaris on se suposa que són.

Aquest paquet no està instal·lat de manera predeterminada, però s'ha d'instal·lar manualment des del repositori propi del RKWard. Podeu fer-ho utilitzant la interfície de la IGU (**Arranjament** → **Configura els paquets**), o des de qualsevol sessió R en execució:

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

El rkwarddev depèn d'un altre paquet petit anomenat 'XiMpLe', que és un analitzador i generador de XML molt senzill i també està present en el mateix repositori.

També es pot trobar la documentació completa en format PDF. Es pot trobar una introducció més detallada per a treballar amb el paquet al resum bàsic del «rkwarddev».

## 15.2 Exemple pràctic

Per a tenir una idea de com «crear scripts en un connector», en comparació amb l'enfocament directe que heu vist en els capítols anteriors, crearem el connector complet de la prova t una vegada més; aquesta vegada només amb les funcions R del paquet rkwarddev.

> SUGGERIMENT
> El paquet afegirà un diàleg nou de la IGU al RKWard a **Fitxer** → **Exporta** → **Crea un script de connector del RKWard**. Com suggereix el nom, podeu crear esquelets de connectors per a una edició posterior. Aquest diàleg al seu torn es va generar amb un script del rkwarddev que trobareu al directori 'demo' del paquet i de codi font de paquet instal·lat, com a exemple addicional. També podeu executar-la cridant `demo("skeleton_dialog")`

### 15.2.1 Descripció de la IGU

Us adonareu immediatament que el flux de treball és molt diferent: al contrari d'escriure directament codi XML, no comenceu amb la definició **<document>**, sinó directament amb els elements del connector que us agradaria tenir al diàleg. Podeu assignar cada element d'interfície, ja siguin caselles de selecció, menús desplegables, «varslots» o qualsevol altra cosa, a objectes individuals de l'R, i després combinar aquests objectes amb la IGU real. El paquet té funcions per a cada etiqueta XML que es pot utilitzar per a definir la IGU del connector, i la majoria fins i tot tenen el mateix nom, només amb el prefix `rk.XML.*`. Per exemple, la definició d'una variable **<varselector>** i dos elements **<varslot>** per a la variable *"x"* i *"y"* de l'exemple de prova t es pot fer mitjançant:

```
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")
```

El detall més interessant probablement és *source=variables*: Una característica destacada del paquet és que totes les funcions poden generar ID automàtics, de manera que no us heu de molestar pensant o recordant en valors d'*id* per a referir-vos a un element específic del connector. Senzillament, podeu donar els objectes R com a referència, ja que totes les funcions que necessiten un ID d'algun altre element també poden llegir-lo des d'aquests objectes. `rk.XML.varselector()` és una mica especial, ja que normalment no té contingut específic per a crear un ID (pot, però només si especifiqueu una etiqueta), de manera que hem d'establir un nom d'ID. Però `rk.XML.varslot()` no necessitaria els arguments *id.name*, així que això seria suficient:

```
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)
```

Per tal de tornar a crear el codi d'exemple fins aquest punt, haureu d'establir tots els valors d'ID manualment. Però com que el paquet facilitarà les nostres vides, a partir d'ara ja no ens preocuparà.

> **SUGGERIMENT**
> El rkwarddev és capaç de fer molta automatització per a ajudar-vos a construir els vostres connectors. No obstant això, podria ser preferible no utilitzar-ho en tota la seva màxima extensió. Si el vostre objectiu és produir codi que no només funcioni, sinó que també es pugui llegir fàcilment i en comparació l'script generador amb un ésser humà, hauríeu de considerar establir sempre ID útils amb *id.name*. Anomenar els vostres objectes R idèntics a aquests ID també ajudarà a obtenir codi de script que sigui fàcil d'entendre.

Si voleu veure com es veu el codi XML de l'element definit si l'exporteu a un fitxer, només podeu cridar l'objecte pel seu nom. Per tant, si heu anomenat 'var.x' en la vostra sessió R, hauríeu de veure quelcom com això:

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

Algunes etiquetes només són útils en el context d'altres. Per tant, per exemple, no trobareu una funció per a l'etiqueta **<option>**. En lloc d'això, tant els botons d'opció com les llistes desplegables es defineixen incloent les seves opcions com una llista amb nom, on els noms representen les etiquetes que es mostraran en el diàleg, i el seu valor és un vector amb nom que pot tenir dues entrades, *val* per al valor d'una opció i el booleà *chk* per a especificar si aquesta opció està marcada de manera predeterminada.

```
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")
        )
)
```

El resultat quedarà així:

```
<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>
```

Tot el que falta als elements de la pestanya «Configuració bàsica» és la casella de selecció per a mostres aparellades, i l'estructuració de tots aquests elements en files i columnes:

```
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()` és una excepció rara on el nom de la funció no conté el nom complet de l'etiqueta, per a desar algun tecleig per a aquest element utilitzat sovint. Això és el que `basic.settings` conté ara:

```
<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>
```

De manera similar, les línies següents crearan objectes R per als elements de la pestanya «Opcions», introduint funcions per a «spinboxes» (botons de selecció de valors), «frames» i «stretch»:

```
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")
```

Tot el que hem de fer ara és reunir els objectes en un «tabbook», i col·locar-lo en una secció «dialog»:

```
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)))
)
```

També podem crear la secció de l'assistent amb les seves dues pàgines utilitzant els mateixos objectes, de manera que els seus ID s'extrauran per a les etiquetes **<copy>**:

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

Això és per a la IGU. El document global es combinarà al final amb `rk.plugin.skeleton()`.

### 15.2.2 Codi JavaScript

Fins ara, l'ús del paquet rkwarddev podria no haver ajudat tant. Això canviarà ara mateix.

En primer lloc, de la mateixa manera que no ens havien d'importar els ID dels elements en definir la disposició de la IGU, no ens haurà d'importar els noms de les variables JavaScript en el pas següent. Si voleu més control, podeu escriure codi JavaScript net i enganxar-lo al fitxer generat. Però probablement és molt més eficient fer-ho de la manera del rkwarddev.

El més notable és que no haureu de definir cap variable, ja que `rk.plugin.skeleton()` pot explorar el codi XML i definir automàticament totes les variables que probablement necessitareu; per exemple, no us molestareu a incloure una casella de selecció si després no utilitzeu el seu valor o estat. Així que podem començar a escriure el codi R real que genera JS immediatament.

> SUGGERIMENT
> La funció `rk.JS.scan()` també pot explorar els fitxers XML existents per a les variables.

El paquet té algunes funcions per a construccions de codi JS que s'utilitzen habitualment en connectors del RKWard, com la funció `echo()` o les condicions `if() {...} else {...}`. Hi ha algunes diferències entre el JS i l'R, p. ex., per a `paste()` a l'R utilitzeu la coma per concatenar les cadenes de caràcters, mentre que per a `echo()` en el JS utilitzeu «+», i les línies han d'acabar amb un punt i coma. Mitjançant l'ús de les funcions R, gairebé podeu oblidar-vos d'aquestes diferències i continuar escrivint codi R.

Aquestes funcions poden prendre diferents classes d'objectes d'entrada: text net, objectes R amb codi XML com a dalt, o al seu torn resultats d'altres funcions JS del paquet. Al final, sempre cridareu `rk.paste.JS()`, el qual es comporta de manera similar a `paste()`, però depenent dels objectes d'entrada els reemplaçarà amb el seu ID XML, el nom de variable JavaScript o fins i tot blocs de codi JavaScript complets.

Per a l'exemple de la prova t, necessitem dos objectes JS: un per a calcular els resultats, i un per a imprimir-los a la funció `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)
```

Com podeu veure, el rkwarddev també proporciona una implementació R de la funció `echo()`. Torna exactament una cadena de caràcters amb una versió JS vàlida d'ella mateixa. També us podeu adonar que tots els objectes R aquí són els que hem creat abans. Se substituiran automàticament pels seus noms de variables, per la qual cosa això hauria de ser bastant intuïtiu. Sempre que necessiteu aquest reemplaçament, es pot utilitzar la funció `id()`, que també retornarà exactament una cadena de caràcters de tots els objectes que s'han indicat (podríeu dir que es comporta com `paste()` amb una substitució d'objecte molt específica).

La funció `js()` és un embolcall que permet utilitzar les condicions **if(){...} else {...}** de l'R com les que esteu acostumat a fer. Es traduiran directament al codi JS. També conserva alguns operadors com **<**, **>=** o **||**, de manera que podeu comparar lògicament els objectes R sense la necessitat de posar cometes la major part del temps. Mirem l'objecte 'JS.calc' resultant, que ara conté una cadena de caràcters amb aquest contingut:

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

```
        } else {}
        echo(")\n");
```

> NOTA
> Alternativament per a les condicions imbricades `if()` en `js()`, podeu utilitzar la funció `ite()`, que es comporta de manera similar a `ifelse()` de l'R. No obstant això, les sentències condicionals construïdes utilitzant `ite()` solen ser més difícils de llegir i s'han de substituir per `js()` sempre que sigui possible.

### 15.2.3 Mapa de connectors

Aquesta secció és molt curta: no cal escriure cap `.pluginmap`, ja que es pot generar automàticament amb `rk.plugin.skeleton()`. La jerarquia del menú es pot especificar mitjançant l'opció *`pluginmap`*:

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

### 15.2.4 Pàgina d'ajuda

Aquesta secció també és molt curta: `rk.plugin.skeleton()` no pot escriure una pàgina d'ajuda completa a partir de la informació que té. Però també pot explorar el document XML cercant elements que probablement mereixen una entrada de pàgina d'ajuda, i crear automàticament una plantilla de pàgines d'ajuda per al nostre connector. Tot el que hem de fer després és escriure algunes línies per a cada secció llistada.

> SUGGERIMENT
> La funció `rk.rkh.scan()` també pot explorar els fitxers XML existents per a crear un esquelet de fitxer d'ajuda.

### 15.2.5 Generació dels fitxers del connector

Ara ve el pas final, en el qual lliurarem tots els objectes generats a `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)
```

Els fitxers es crearan en un directori temporal predeterminat. Les últimes tres opcions no són necessàries, però són molt útils: *load=TRUE* afegirà automàticament el connector nou a la configuració del RKWard (ja que està en un directori temporal i, per tant, deixarà d'existir quan es tanqui el RKWard, s'eliminarà automàticament de nou pel RKWard durant l'inici següent), *edit=TRUE* obrirà tots els fitxers creats per a editar-los a les pestanyes de l'editor del RKWard, i *show=TRUE* intentarà llançar directament el connector, de manera que podreu examinar com es veu sense un clic. Podríeu considerar afegir *overwrite=TRUE* si executareu l'script repetidament (p. ex., després dels canvis al codi), ja que de manera predeterminada no se sobreescriuran fitxers.

L'objecte resultant 'plugin.dir' conté el camí al directori on es va crear el connector. Això pot ser útil en combinació amb la funció `rk.build.package()`, per a construir un paquet R real per a compartir el vostre connector amb altres; p. ex., per enviant-lo a l'equip de desenvolupament del RKWard per a afegir-lo al nostre repositori de connectors.

### 15.2.6 L'script complet

Per a recapitular tot l'anterior, aquí hi ha l'script complet per a crear l'exemple de prova t que funciona. Afegint el codi ja explicat, també carrega el paquet si cal, i utilitza l'entorn `local()`, de manera que tots els objectes creats no acabaran a l'espai de treball actual (excepte el 'plugin.dir'):

```
require(rkwarddev)

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

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

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

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

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

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

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

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

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

## 15.3 Afegir pàgines d'ajuda

Si voleu escriure una pàgina d'ajuda per al connector, la manera més directa de fer-ho és afegir les instruccions particulars directament a les definicions dels elements XML als quals pertanyen:

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

El text donat al paràmetre *help* es pot obtenir amb `rk.rkh.scan()` i s'escriu a la pàgina d'ajuda d'aquest component del connector. Perquè això funcioni tècnicament, tanmateix, `rk.rkh.scan()` ha de saber quins objectes R pertanyen a un component del connector. Per això també heu de proporcionar el paràmetre *component* i assegurar-vos que és idèntic per a tots els objectes que pertanyen junts.

Com que normalment combinareu molts objectes en un diàleg i també us podria agradar reutilitzar objectes com el **<varslot>** en diversos components dels vostres connectors, és possible definir globalment un component amb `rk.set.comp()`. Si es defineix, s'assumeix que tots els objectes següents utilitzats en el vostre script pertanyen a aquest component en particular, fins que `rk.set.comp()` es torna a cridar amb un nom de component diferent. Després podeu ometre el paràmetre *component*:

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

Per a afegir seccions globals com **<summary>** o **<usage>** a la pàgina d'ajuda, utilitzeu funcions com `rk.rkh.summary()` o `rk.rkh.usage()` d'acord amb això. Els seus resultats s'utilitzen per a establir els elements de la llista com *summary* o *usage* en el paràmetre *rkh* de `rk.plugin.component()`/`rk.plugin.skeleton()`.

## 15.4 Connectors de traducció

El paquet rkwarddev és capaç de produir connectors externs amb suport complet d'«i18n». Per exemple, totes les funcions rellevants que generen objectes XML ofereixen un paràmetre opcional per a especificar *i18n_context* o *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)
```

Els exemples anteriors produeixen una sortida com aquesta:

```
# 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" />
```

També hi ha suport per al codi JS traduïble. De fet, el paquet intenta afegir les crides `i18n()` predeterminades en els llocs on això sol ser útil. La funció `rk.JS.header()` és un bon exemple:

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

Això produeix el codi JS següent:

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

Però també podeu marcar manualment les cadenes del codi JS com a traduïbles, utilitzant la funció `i18n()` tal com ho faríeu si escriviu el fitxer JS directament.

# Apèndix A

# Referència

## A.1 Tipus de propietats/Modificadors

En alguns llocs d'aquesta introducció hem parlat de «propietats» d'elements de la IGU o d'altres. De fet, hi ha diversos tipus de propietats. Normalment no cal que us preocupeu per això, ja que podeu utilitzar el sentit comú per a connectar qualsevol propietat a qualsevol altra propietat. No obstant això, internament, hi ha tipus diferents de propietats. El que importa és quan s'obtenen alguns valors especials a la plantilla JS. A les sentències getString (″id″)/getBoolean (″id″)/getList (″id″) també podeu especificar alguns dels anomenats «modificadors» com aquest: `getString (″id.modifier″)`. Aquest modificador afectarà, de manera que s'imprimirà el valor. Llegiu la llista de propietats i els modificadors que publiquen:

**Propietats de cadena**

El tipus de propietat més senzill, utilitzat per a contenir un tros de text. Modificadors:

**Sense modificador (″″)**
: La cadena com a definida/establerta.

**quoted**
: La cadena en la forma citada (adequada per a passar a R com a caràcter).

**Propietats booleanes**

Propietats que poden estar activades o desactivades, certes o falses. Per exemple, les propietats creades per les etiquetes <convert>, també la propietat que acompanya una <checkbox> (vegeu a sota). Es retornaran els valors següents segons el modificador indicat:

**Sense modificador (″″)**
: Per defecte, la propietat retornarà 1 si és certa i 0 en cas contrari. La manera recomanada d'obtenir els valors booleans és utilitzant `getBoolean()`. Tingueu en compte que per a `getString()`, es retornarà la cadena ″0″ quan la propietat sigui falsa. Aquesta cadena s'avaluaria a certa, no a falsa en JS.

**″labeled″**
: Retorna la cadena «true» (certa) quan és certa, «false» (falsa) quan és falsa, o qualsevol cadena personalitzada que s'hagi especificat (normalment en una <checkbox>).

**″true″**
: Retorna la cadena com si la propietat fos certa, encara que sigui falsa

**″false″**
: Retorna la cadena com si la propietat fos falsa, encara que sigui certa

**"not"**
: Això realment retorna una altra propietat booleana, la qual és la inversa de l'actual (és a dir, falsa si és certa, certa si és falsa)

**"numeric"**
: Obsoleta, proporcionada per a la compatibilitat cap enrere. Igual que sense modificador "". Retorna "1" si la propietat és certa, o "0" si és falsa.

**Propietats dels enters**

Una propietat dissenyada per a tenir un valor enter (però, per descomptat, encara retorna una cadena de caràcters numèrics a la plantilla JS). No accepta cap modificador. Utilitzat en les <spinbox> (vegeu a sota)

**Propietats dels nombres reals**

Una propietat dissenyada per a tenir un valor de nombre real (però, per descomptat, encara retorna una cadena de caràcters numèrics a la plantilla JS). Utilitzat en les <spinbox> (vegeu a sota)

**Sense modificador ("")**
: Per a `getValue()/getString()`, això retorna el mateix que "formatted". En canvi, en versions futures serà possible obtenir una representació numèrica.

**"formatted"**
: Retorna el nombre amb format (com a cadena).

**Propietats dels RObject**

Una propietat destinada a una selecció d'un o més objectes R. S'utilitza de forma més destacada en «varselectors» i «varslots». Es retornaran els valors següents segons el modificador indicat:

**Sense modificador ("")**
: De manera predeterminada, la propietat retornarà el nom complet de l'objecte seleccionat. Si se selecciona més d'un objecte, els noms dels objectes se separaran amb salts de línia ("\n").

**"shortname"**
: Com a dalt, però només retorna els noms curts dels objectes. Per exemple, un objecte dins d'una llista només rebria el nom que té dins de la llista, sense el nom de la llista.

**"label"**
: Com a dalt, però retorna les etiquetes del RKWard dels objectes (si no hi ha cap etiqueta disponible, aquesta serà la mateixa que el nom curt)

**Propietats de les llistes de cadenes**

Aquesta propietat conté una llista de cadenes.

**Sense modificador ("")**
: Per a `getValue()/getString()`, això retorna totes les cadenes separades per «\n». Qualsevol caràcter «\n» de cada element s'escapa com a literal «\n». No obstant això, l'ús recomanat és obtenir el valor amb `getList()`, en el seu lloc, el qual retornarà una matriu de cadenes.

**"joined"**
: Retorna la llista com una única cadena, amb els elements units amb «\n». En contrast a sense modificador (""), les cadenes individuals _no_ s'escapen.

**Propietats del codi**

Una propietat dels connectors que han generat el codi. Això és important pels connectors incrustadors, per tal d'incrustar el codi generat pel connector incrustat en el codi generat pel connector incrustador (nivell superior). Es retornaran els valors següents segons el modificador indicat:

**Sense modificador ("")**
: Retorna el codi complet, és a dir, les seccions preprocess», «calculate», «printout» i (però no «preview») concatenades en una cadena.

**"preprocess"**
: Retorna només la secció «preprocess» del codi

**"calculate"**
: Retorna només la secció «calculate» del codi

**"printout"**
: Retorna només la secció «printout» del codi

**"preview"**
: Retorna la secció «preview» del codi

## A.2 Elements de propòsit general que s'utilitzaran en qualsevol fitxer XML (`.xml`, `.rkh`, `.pluginmap`)

**<snippets>**
: Permès com a fill directe del node <document> i només allà. S'ha de col·locar a prop de la part superior del fitxer. Vegeu la secció sobre l'ús de fragments. Només pot estar present un element <snippets>. Opcional, sense atributs.

**<snippet>**
: Defineix un fragment únic. Només es permet com a fill directe de l'element <snippets/>. Atributs:

  **<id>**
  : Una cadena d'identificador per al fragment. Requerit.

**<insert>**
: Insereix el contingut d'un <snippet>. Permès en qualsevol lloc. Atributs:

  **<snippet>**
  : La cadena de l'identificador del fragment a inserir. Requerit.

**<include>**
: Inclou el contingut d'un altre fitxer XML (tot dins de l'element <document> d'aquest fitxer). Permès en qualsevol lloc. Atributs:

  **<file>**
  : El nom del fitxer, relatiu al directori a on és el fitxer actual. Requerit.

## A.3 Elements a utilitzar en la descripció XML del connector

Les propietats dels elements s'enumeren en una secció separada.

### A.3.1 Elements generals

**<document>**

Cal que sigui present en cada fitxer «description.xml» com a node arrel. No hi ha cap funció especial. Sense atributs

**<about>**

Informació sobre aquest connector (autor, llicència, etc.). Aquest element està permès tant en el fitxer `.xml` d'un connector individual com en els fitxers `.pluginmap`. Consulteu la referència del fitxer `.pluginmap` per als detalls de referència, el capítol sobre la informació «about» per a una introducció.

**<code>**

Defineix on cercar la plantilla JS al connector. Utilitzeu-ho només una vegada per fitxer, com a fill directe de l'etiqueta del document. Atributs:

***file***

Nom de fitxer de la plantilla JS, relatiu al directori on està l'«xml» del connector

**<help>**

Defineix on cercar el fitxer d'ajuda del connector. Utilitzeu-ho només una vegada per fitxer, com a fill directe de l'etiqueta del document. Atributs:

***file***

Nom de fitxer del fitxer d'ajuda, relatiu al directori on està l'«xml» del connector

**<copy>**

Es pot utilitzar com a fill (directe o indirecte) dels elements principals de la disposició, és a dir, <dialog> i <wizard>. Això s'utilitza per a copiar 1:1 tot un bloc a elements XML. Atributs:

***id***

L'ID a cercar. L'etiqueta <copy> cercarà un element XML anterior que tingui el mateix ID, i el copiarà incloent-hi tots els elements descendents.

***copy_element_tag_name***

En alguns casos, voldreu una còpia gairebé literal, però canvieu el nom d'etiqueta de l'element a copiar. L'exemple més important d'això és quan voleu copiar tot un <tab> des d'una interfície de diàleg a <page> d'una interfície assistent. En aquest cas, estableu copy_element_tag_name="page" per a fer aquesta conversió automàticament.

### A.3.2 Definicions d'interfície

**<dialog>**

Defineix una interfície de tipus diàleg. Col·loca la definició de la IGU dins d'aquesta etiqueta. Utilitzeu només una vegada per fitxer, com a fill directe de l'etiqueta document. Es requereix almenys una de les etiquetes «dialog» o «wizard» per a un connector. Atributs:

***label***

Llegenda per al diàleg

***recommended***

S'ha d'utilitzar el diàleg com a interfície «recomanada» (és a dir, la interfície que es mostrarà de manera predeterminada, llevat que l'usuari hagi configurat el RKWard a una interfície específica com a predeterminada)? Aquest atribut actualment no té cap efecte, ja que és implícitament «true», llevat que es recomani l'assistent.

**<wizard>**

Defineix una interfície de tipus assistent. Col·loca la definició de la IGU dins d'aquesta etiqueta. Utilitzeu només una vegada per fitxer, com a fill directe de l'etiqueta del document. Es requereix almenys una de les etiquetes «dialog» o «wizard» per a un connector. Accepta només etiquetes <page> o <embed> com a fills directes. Atributs:

***label***

Llegenda per a l'assistent

***recommended***

S'ha d'utilitzar l'assistent com a interfície «recomanada» (és a dir, la interfície que es mostrarà de manera predeterminada, llevat que l'usuari hagi configurat el RKWard a una interfície específica com a predeterminada)? Opcional, el valor predeterminat és «false».

### A.3.3 Elements de disposició

Tots els elements d'aquesta secció accepten un atribut id=″identifierstring″. Aquest atribut és opcional per a tots els elements. Es pot utilitzar, per exemple, per a ocultar/inhabilitar tot l'element de la disposició i tots els elements continguts en ell (vegeu el capítol de la lògica de la IGU). La cadena «id» no pot contenir «.» (punt) ni «;» (punt i coma), i generalment s'ha de limitar a caràcters alfanumèrics i el guió baix («_»). Només es llisten els atributs addicionals.

**<page>**

Defineix una pàgina nova dins d'un assistent. Només es permet com a fill directe d'un element <wizard>.

**<row>**

Tots els fills directes d'una etiqueta «row» es col·locaran d'esquerra a dreta.

**<column>**

Tots els fills directes d'una etiqueta «column» es col·locaran de dalt a baix.

**<stretch>**

De manera predeterminada, els elements de la IGU ocupen tot l'espai disponible. Per exemple, si teniu dues columnes una al costat de l'altra, l'esquerra està empaquetada amb elements, però la dreta només conté un sol **<radio>**, el control **<radio>** s'expandirà verticalment, tot i que realment no necessita l'espai disponible, i es veurà lleig. En aquest cas, realment voleu afegir un «blank» a sota del **<radio>**. Per a això, utilitzeu l'element <stretch>. Simplement utilitzarà una mica d'espai. No utilitzeu en excés aquest element, normalment és una bona idea que els elements de la IGU obtinguin tot l'espai disponible, només de vegades la disposició es veurà espaiada. L'element <stretch> no pren cap argument, ni tan sols una «id». Tampoc podeu posar un fill dins de l'element <stretch> (en altres paraules, només l'utilitzareu com a «<stretch/>»)

**<frame>**

Dibuixa un marc/quadre al voltant dels seus fills directes. Es pot utilitzar per a agrupar visualment les opcions relacionades. La disposició dins d'un marc és de dalt a baix, llevat que col·loqueu una <row> a dins. Atributs:

***label***

Llegenda per al marc (opcional)

***checkable***

Els marcs es poden marcar. En aquest cas, tots els elements continguts s'inhabilitaran quan el marc no estigui marcat i s'habilitaran quan estigui marcat. (Opcional, de manera predeterminada és ″false″)

***checked***
Només per als marcs que es poden marcar: s'ha de marcar el marc de manera predeterminada? El valor predeterminat és «true». No s'interpreta per a marcs que no es poden marcar.

**<tabbook>**
Organitza elements en un «tabbook». Accepta només etiquetes <tab> com a fills directes.

**<tab>**
Defineix una pàgina en un «tabbook». Col·loca la definició de la IGU per a la pestanya dins d'aquesta etiqueta. Només es pot utilitzar com a fill directe d'una etiqueta <tabbook>. Un <tabbook> hauria de tenir almenys dues pestanyes definides. Atributs:

***label***
Llegenda per a la pàgina de pestanya (requerit)

**<text>**
Mostra el text envoltat per aquesta etiqueta a la IGU. S'admeten alguns marcadors senzills d'estil HTML (en particular <b>, <i>, <p>, i <br/>). No obstant això, mantingueu el format al mínim. Inserir una línia completament buida afegeix un salt de línia dur. Atributs:

***type***
Tipus del text. Un de ″normal″, ″avís″ o ″error″. Això influeix en l'aspecte del text (opcional, el predeterminat és ″normal″)

### A.3.4 Elements actius

Tots els elements d'aquesta secció accepten l'atribut id=″identifierstring″. Aquest atribut és obligatori per a tots els elements. Només es llisten els atributs addicionals. La cadena d'identificació no pot contenir ″.″ (punts).

**<varselector>**
Proporciona una llista d'objectes disponibles des dels quals l'usuari pot seleccionar-ne un o més. Requereix una o més <varslot> com a contrapartida per a ser útil. Atributs:

***label***
Etiqueta per al «varselector» (opcional, el predeterminat és ″Select variable(s)″)

**<varslot>**
S'utilitza conjuntament amb un ″varselector″ per a permetre que l'usuari seleccioni una o més variables. Atributs:

***label***
Etiqueta per al «varslot» (recomanada, la predeterminada és «Variable:»)

***source***
El «varselector» des d'on obtenir la selecció (requerit, llevat que es connecti manualment o usant «source_property»)

***source_property***
Una propietat arbitrària d'on copiar els valors, quan es fa clic al botó de selecció. Si s'especifica, això anul·la l'atribut «source».

***required***
Si es requereix que per a enviar el codi aquest «varslot» tingui un valor vàlid. Vegeu la propietat requerida (opcional, predeterminada a ″false″)

***multi***
Si el «varslot» només conté un (predeterminat, «fals») o diversos objectes

***allow_duplicates***
Si el «varslot» només pot acceptar objectes únics (predeterminat, «fals»), o si el mateix objecte es pot afegir diverses vegades.

***min_vars***
: Només té sentit si és multi="true": nombre mínim de variables que s'han de seleccionar perquè la selecció es consideri vàlida (opcional, el valor predeterminat és "1")

***min_vars_if_any***
: Només té sentit si és multi="true": Alguns «varslots» es poden considerar vàlids, si, per exemple, el «varslot» està buit o té almenys dos valors. Això especifica quantes variables s'han de seleccionar si n'hi ha (2 a l'exemple). (Opcional, el valor predeterminat és "1")

***max_vars***
: Només té sentit si és multi="true": nombre màxim de variables a seleccionar (opcional, el valor predeterminat és "0", el qual vol dir sense màxim)

***classes***
: Si especifiqueu un o més noms de classe R (separats per espais (" ")), aquí, el «varslot» només acceptarà objectes pertanyents a aquestes classes (opcional, *useu-ho amb molta cura*, no s'hauria d'impedir a l'usuari prendre decisions vàlides, i R té *moltes* classes diferents)

***types***
: Si especifiqueu un o més tipus de variables (separades per espais («)»), aquí, el «varslot» només acceptarà objectes d'aquests tipus. Els tipus vàlids són "unknown", "number", "string", "factor", "invalid". (Opcional, *useu-ho amb molta cura*, no s'ha d'impedir a l'usuari que prengui decisions vàlides, i el RKWard no sempre coneix el tipus d'una variable)

***num_dimensions***
: El nombre de dimensions que un objecte necessita tenir. "0" (predeterminat), vol dir que qualsevol nombre de dimensions és acceptable. (Opcional, el valor predeterminat és "0")

***min_length***
: La longitud mínima que un objecte ha de tenir per a ser acceptable. (Opcional, el valor predeterminat és "0")

***max_length***
: La longitud màxima que un objecte ha de tenir per tal de ser acceptable. (Opcional, predeterminat pel nombre enter més gran representable al sistema)

**<valueselector>**
: Proporciona una llista de cadenes disponibles (no objectes R) que se seleccionaran en una o més àrees d'acompanyament <valueslot>. Les opcions de cadena es poden definir utilitzant etiquetes <option> com a fills directes (vegeu més avall), o establir-les utilitzant propietats dinàmiques. Atributs:

***label***
: Etiqueta per al «valueselector» (opcional, predeterminat a sense etiqueta)

**<valueslot>**
: S'utilitza conjuntament amb un <valueselector> per a permetre a l'usuari seleccionar un o més elements de cadena. Aquest element és majoritàriament idèntic a <varslot>, i comparteix els mateixos atributs, excepte els que es refereixen a les propietats dels elements acceptables (és a dir classes, types, num_dimensions, min_length, max_length).

**<radio>**
: Defineix un grup de botons exclusius d'opcions (només es pot seleccionar un alhora). Requereix almenys dues etiquetes <option> com a fills directes. No es permeten altres etiquetes com a filles. Atributs:

***label***
: Etiqueta per al control d'opcions (recomanada, la predeterminada és «Select one:»)

**<dropdown>**

Defineix un grup d'opcions de les quals es pot seleccionar una i només una alhora, utilitzant una llista desplegable. Això és funcionalment equivalent a un <radio>, però es veu diferent. Requereix almenys dues etiquetes <option> com a filles directes. No es permeten altres etiquetes com a filles. Atributs:

***label***
Etiqueta per a la llista desplegable (recomanada, la predeterminada és «Select one:»)

**<select>**

Proporciona una llista de cadenes disponibles des de les quals l'usuari pot seleccionar un nombre arbitrari. Les opcions de cadena es poden definir utilitzant etiquetes <option> com a filles directes (vegeu més avall), o establir-les utilitzant propietats dinàmiques. Atributs:

***label***
Etiqueta per a <select> (opcional, predeterminat a sense etiqueta)

***single***
Si s'estableix a cert, només es podrà seleccionar un valor únic, en lloc de diversos valors alhora (booleà, predeterminat a fals)

**<option>**

Només es pot utilitzar com a fill directe d'un element <radio>, <dropdown>, <valueselector> o <select>. Representa una opció seleccionable en un control d'opcions o llista desplegable. Com que els elements <option> sempre formen part d'un dels elements de selecció, normalment no tenen una ″id″ pròpia, però vegeu a continuació. Atributs:

***label***
Etiqueta per a l'opció (requerit)

***value***
El valor de la cadena que retornarà l'element pare si aquesta opció està marcada/seleccionada (requerit)

***checked***
Si l'opció s'ha de marcar/seleccionar de manera predeterminada a «true» o «false». En un <radio> o <dropdown>, només es pot establir una opció a `checked="true"`, i si no s'estableix cap opció a «checked», es marcarà/seleccionarà automàticament la primera opció de l'element pare. En un <select>, es pot establir com a «checked» qualsevol nombre d'opcions. (Opcional, predeterminat a ″false″)

***id***
És opcional especificar el paràmetre «id» per als elements <option> (i de fet, es recomana no establir un «id», llevat que realment en necessiteu un). No obstant això, especificar un ″id″ us permetrà habilitar/inhabilitar dinàmicament <option>, connectant-vos a la propietat booleana `id_of_radio.id_of_optionX.enabled`. Actualment això funciona per a opcions dins d'elements <radio> o <dropdown>, només; actualment les opcions <valueselector> i <select> no admeten els identificadors.

**<checkbox>**

Defineix una casella de selecció, és a dir, una opció única que es pot activar o desactivar. Atributs:

***label***
Etiqueta per a la casella de selecció (requerit)

***value***
El valor que la casella de selecció retornarà si està marcada (requerit)

***value_unchecked***
El valor que es retornarà si la casella de selecció no està marcada (opcional, el valor predeterminat és ″″, és a dir, una cadena buida)

***checked***
Si l'opció s'ha de marcar per defecte ″true″ o ″false″ (opcional, predeterminat a ″false″)

**<frame>**

L'element del marc s'utilitza generalment com un element de disposició pur, i es llista a la secció sobre elements de disposició. No obstant això, també es pot fer que es pugui marcar, actuant així com una casella de selecció senzilla al mateix temps.

**<input>**

Defineix un camp d'entrada de text lliure. Atributs:

***label***
: Etiqueta per al camp d'entrada (requerida)

***initial***
: Text inicial del camp de text (opcional, el valor predeterminat és ″″, és a dir, una cadena buida)

***size***
: Un de ″small″, ″medium″, o ″large″. ″large″ defineix un camp d'entrada multilínia, ″small″ i ″medium″ són camps d'una sola línia (opcional, el predeterminat és ″medium″)

***required***
: Si es requereix que per a enviar el codi aquesta entrada no estigui buida. Vegeu la propietat requerida (opcional, predeterminat a ″false″)

**<matrix>**

Una taula per a introduir dades de matrius (o vectors) a la IGU.

> NOTA
> Aquest element d'entrada *no* està optimitzat per a introduir/editar grans quantitats de dades. Si bé no hi ha un límit estricte en la mida d'una <matrix>, en general no hauria de superar al voltant de deu files/columnes. Si espereu dades més grans, permeteu als usuaris seleccionar-lo com un objecte R (el qual pot ser una bona idea com a opció alternativa, en gairebé *cada* instància on utilitzeu un element de matriu).

Atributs:

***label***
: Etiqueta per a la taula (requerida)

***mode***
: Un d'entre ″integer″, ″real″, o ″string″. El tipus de dades que s'acceptaran a la taula (requerit)

***min***
: Valor mínim acceptable (per a matrius de tipus ″integer″ o ″real″) (opcional, predeterminat al valor més petit representable)

***max***
: Valor màxim acceptable (per a matrius de tipus ″integer″ o ″real″) (opcional, predeterminat al valor més gran representable)

***allow_missings***
: Si es permeten valors (buits) a la matriu. Això està implícit per a matrius o mode «cadena» (opcional, el valor predeterminat és ″false″).

***allow_user_resize_columns***
: Quan s'estableix a «true» (cert), l'usuari pot afegir columnes escrivint a les cel·les (inactives) més a la dreta (opcional, predeterminat a ″true″).

***allow_user_resize_rows***
: Quan s'estableix a «true» (cert), l'usuari pot afegir files escrivint a les cel·les (inactives) de la part de baix de tot (opcional, predeterminat a ″true″).

***rows***
: Nombre de files a la matriu. No té cap efecte per allow_user_resize_rows=″true″.

> NOTA
> Això també es pot controlar establint la propietat ″rows″.

(opcional, predeterminat a 2).

***columns***
Nombre de columnes a la matriu. No té cap efecte per allow_user_resize_columns=″true″.

> NOTA
> Això també es pot controlar establint la propietat ″columns″.

(opcional, predeterminat a 2).

***min_rows***
Nombre mínim de files a la matriu. La matriu refusarà reduir-se per sota d'aquesta mida. (Opcional, el valor predeterminat és 0; vegeu també: `allow_missings`).

***min_columns***
Nombre mínim de columnes a la matriu. La matriu refusarà reduir-se per sota d'aquesta mida. (Opcional, el valor predeterminat és 0; vegeu també: `allow_missings`).

***fixed_height***
Força l'element de la IGU a romandre a la seva alçada inicial. No l'utilitzeu en combinació amb matrius, on el nombre de files pot canviar de qualsevol manera. Especialment útil quan es crea un element d'entrada vectorial (columns=″1″). Amb aquesta opció establerta a «true» (cert), no es mostrarà cap barra de desplaçament horitzontal, fins i tot si la matriu excedeix l'amplada disponible (ja que això afectaria l'alçada). (Opcional, el valor predeterminat és ″false″).

***fixed_width***
Nom lleugerament incorrecte: suposa que el comptador de columnes no canviarà. L'última columna (o normalment només) s'estirarà per a agafar l'amplada disponible. No l'utilitzeu en combinació amb matrius, on el nombre de columnes pot canviar de qualsevol manera. Especialment útil quan es crea un element d'entrada vectorial (rows=″1″). (Opcional, el valor predeterminat és ″false″).

***horiz_headers***
Cadenes a usar per a la capçalera horitzontal, separades per «;». La capçalera s'ocultarà, si s'estableix a ″″. (Opcional, el valor predeterminat és el número de columna).

***vert_headers***
Cadenes a usar per a la capçalera vertical, separades per «;». La capçalera s'ocultarà, si s'estableix a ″″. (Opcional, el valor predeterminat és el número de fila).

**<optionset>**

Una interfície d'usuari per a repetir un conjunt d'opcions per a un nombre arbitrari d'elements (introducció als «optionsets»). Atributs:

***min_rows***
Si s'especifica, el conjunt es marcarà com a no vàlid, llevat que tingui almenys aquest nombre de files (opcional, enter).

***min_rows_if_any***
Com «min_rows», però només es provaran si hi ha almenys una fila (opcional, enter).

***max_rows***
Si s'especifica, el conjunt es marcarà com a no vàlid, llevat que tingui com a màxim aquest nombre de files (opcional, enter).

***keycolumn***
L'ID de la columna per a actuar com a «keycolumn». Un «optionset» amb una «keycolumn» (vàlida) actuarà com un «optionset» ″driven″. Un «optionset» sense «keycolumn» permetrà la inserció/eliminació manual d'elements. La «keycolumn» s'ha de marcar com a externa. (Opcional, de manera predeterminada no hi ha cap «keycolumn»).

Elements fills:

**<optioncolumn>**

Declara una «optioncolumn» del conjunt. Per a cada valor que voleu obtenir des de l'«optionset», haureu de declarar una <optioncolumn> separada. Atributs:

***id***
L'identificador de l'«optioncolumn» (requerit, cadena).

***external***
Establiu-ho a «true» (cert), si l'«optioncolumn» es controla des de fora de l'«optionset» (opcional, booleà, el valor predeterminat és ″false″).

***label***
Si s'indica, l'«optioncolumn» es mostrarà en una columna amb aquesta etiqueta (opcional, cadena, de manera predeterminada no es mostrarà).

***connect***
La propietat a la qual connectar aquesta «optioncolumn», donada com a «id» dins de l'àrea <content>. Per a <optioncolumn> externes, el valor corresponent s'establirà al valor establert externament. Per a les <optioncolumn> normals (no externes), la fila corresponent de la propietat <optioncolumn> s'establirà quan la propietat canviï dins de l'àrea de contingut. (Opcional, cadena, de manera predeterminada no està connectada).

***default***
Només per a columnes externes: el valor que s'ha d'assumir per a aquesta columna, si no es coneix cap valor per a una entrada. És útil rares vegades. (Opcional, el valor predeterminat és una cadena buida)

**<content>**

Declara el contingut/IU del conjunt. Sense atributs. Es permeten tots els elements actius, passius i de disposició habituals com a elements de nom fill. A més, en versions anteriors del RKWard (fins a 0,6.3), es va permetre l'element fill especial **<optiondisplay>**. Això és obsolet en el RKWard 0.6.4, i simplement s'hauria d'eliminar dels connectors existents.

**<logic>**

Especificació opcional de la lògica de la IU a aplicar *dins de* la regió de contingut de l'«optionset». Vegeu la referència a <logic>

**<browser>**

Un element dissenyat per a seleccionar un únic nom de fitxer (o nom de directori). Tingueu en compte que aquest camp prendrà qualsevol cadena, encara que estigui destinada a utilitzar-se només per a fitxers:

***label***
Etiqueta per al navegador (opcional, el valor predeterminat és ″Enter filename″)

***initial***
Text inicial del navegador (opcional, predeterminat a ″″, és a dir, una cadena buida)

***type***
Un d'entre ″file″, ″dir″, o ″savefile″. Per a seleccionar un fitxer existent, un directori existent o un fitxer no existent, respectivament (opcional, el valor predeterminat és ″file″)

***allow_urls***
Si es poden seleccionar URL (no locals) (opcional, predeterminat a ″false″)

***filter***
Filtre del tipus de fitxer, p. ex. (″*.txt *.csv″ per als fitxers .txt i .csv). Automàticament s'afegeix una entrada separada per a ″All files″ (opcional, per defecte a ″″, és a dir ″All files″)

***required***
Si es requereix que per a enviar el codi el camp no estigui buit. Tingueu en compte que això no significa necessàriament que el nom de fitxer seleccionat sigui vàlid. Vegeu la propietat requerida (opcional, predeterminat a ″true″)

**<saveobject>**

Un element dissenyat per a seleccionar el nom d'un objecte R a on desar (és a dir, generalment no existeix prèviament, en contrast amb un «varslot»):

***label***
Etiqueta per a l'entrada (opcional, predeterminada a ″Save to:″)

***initial***
Text inicial de l'entrada (opcional, predeterminat a ″my.data″)

***required***
Si es requereix que per a enviar el codi el camp tingui un nom d'objecte admès. Vegeu la propietat requerida (opcional, predeterminat a ″true″)

***checkable***
En molts casos d'ús, desar a un objecte R és opcional. En aquests casos, es pot integrar una casella de selecció a l'element «saveobject» usant aquest atribut. Quan s'estableix a «true» (cert), la casella de selecció activarà/desactivarà el «saveobject». Vegeu la propietat activa de «saveobject» (opcional, predeterminat a ″false″)

***checked***
Per a elements «saveobject» seleccionables, només: si el control està marcat/activat de manera predeterminada (opcional, el valor predeterminat és ″false″)

**<spinbox>**

Un «spinbox» en el qual l'usuari pot seleccionar un valor numèric, utilitzant l'entrada directa del teclat o les fletxes amunt/avall petites. Atributs:

***label***
Etiqueta per al «spinbox» (recomanada, la predeterminada és «Enter value:»)

***min***
El valor més baix que l'usuari pot introduir en el «spinbox» (opcional, de manera predeterminada el valor més baix que es pot representar tècnicament en el «spinbox»)

***max***
El valor més gran que l'usuari pot introduir al botó de selecció de valors (opcional, de manera predeterminada el valor més alt que es pot representar tècnicament en el «spinbox»)

***initial***
El valor inicial que es mostra al botó de selecció de valors (opcional, el valor predeterminat és ″0″)

***type***
Un entre ″real″ o ″integer″. Si el botó de selecció de valors accepta nombres reals o només enters (opcional, el valor predeterminat és ″real″)

***default_precision***
Només té sentit si el «spinbox» és de tipus=″real″. Especifica el nombre predeterminat de xifres decimals que es mostren al botó de selecció de valors (només es mostraran aquests zeros finals). Quan l'usuari prem les fletxes amunt/avall, aquest lloc decimal es canviarà. Tanmateix, l'usuari pot ser capaç d'introduir valors amb una precisió més alta (vegeu més avall) (opcional, predeterminat a ″2″)

***max_precision***
El nombre màxim de dígits que es poden representar significativament (opcional, predeterminat a ″8″)

**<formula>**

Aquest element avançat permet a l'usuari seleccionar una fórmula/conjunt d'interaccions a partir de les variables seleccionades. Per exemple, per un GLM, aquest element es pot utilitzar per a permetre a l'usuari especificar els termes d'interacció en el model. Atributs:

***fixed_factors***
L'ID del «varslot» que conté els factors fixos seleccionats (requerit)

***dependent***
L'ID del «varslot» que conté la variable dependent seleccionada (requerit)

**<embed>**
Incrusta un connector diferent en aquest (vegeu el capítol sobre la incrustació). Atributs:

***component***
El nom registrat del component a incrustar (vegeu el capítol sobre el registre de components) (requerit)

***as_button***
Si s'estableix a "true", només es col·locarà un botó a la IGU incrustada, la IGU incrustada només es mostrarà (en una finestra separada) quan es premi el botó (opcional, de manera predeterminada és "false")

***label***
Només té sentit si as_button="true": l'etiqueta del botó (recomanada, la predeterminada és «Options»)

**<preview>**
Una casella de selecció per a commutar la funcionalitat de vista prèvia. Tingueu en compte que a partir de la versió 0.6.5 del RKWard els elements de vista prèvia **<preview>** són casos especials en els diàlegs de connectors (no assistents): es col·locaran a la columna de botons, independentment d'on estiguin exactament definits a la interfície d'usuari. Continua sent una bona idea definir-les en un lloc assenyat de la disposició, per a la compatibilitat cap endarrere. Atributs:

***label***
Etiqueta del quadre (opcional, el valor predeterminat és "Preview")

***mode***
Tipus de vista prèvia. Els tipus admesos són "plot" (vegeu el capítol sobre les previsualitzacions de gràfics), "output" (vegeu el capítol sobre les previsualitzacions de sortida (HTML), "data" (vegeu les previsualitzacions de dades), i "custom" (vegeu les previsualitzacions personalitzades ). (Opcional, el valor predeterminat és "plot")

***placement***
Col·locació de la vista prèvia: "attached" (al lloc de treball principal), "detached" (finestra independent), "docked" (adjunt al diàleg del connector) i "default" (actualment és el mateix que "docked", però pot arribar a ser configurable per l'usuari en algun moment). En general, es recomana deixar-ho com a paràmetre predeterminat per a la millor coherència de la interfície d'usuari (opcional, el valor predeterminat és "default")

***active***
Si la vista prèvia està activa de manera predeterminada. En general, només les vistes prèvies acoblades es faran actives de manera predeterminada, i fins i tot per a aquestes, hi ha un motiu pel qual el valor predeterminat és de vistes prèvies inactives (opcional, el valor predeterminat és "false")

### A.3.5 Secció de lògica

**<logic>**
L'element que conté la secció lògica. Tots els elements següents només es permeten dins de l'element <logic>. L'element <logic> només es permet com a fill directe de l'element <document> (com a màxim una vegada per document), o dels elements <optionset> (com a màxim una vegada per «optionset»). La secció lògica del document s'aplica tant a la IGU de <dialog> com a <wizard> de la mateixa manera.

**<external>**

Crea una propietat nova (cadena) que se suposa que està connectada a una propietat externa si el connector s'incrusta. Vegeu la secció sobre els connectors «incomplets». Atributs:

***id***
L'ID de la propietat nova (requerit)

***default***
El valor predeterminat de la cadena de la propietat nova, és a dir, el valor utilitzat si la propietat no està connectada a una propietat externa (opcional, el valor predeterminat és una cadena buida)

**<i18n>**

Crea una propietat nova (cadena) que se suposa que proporciona una etiqueta que es pot traduir (i18n). Atributs:

***id***
L'ID de la propietat nova (requerit)

***label***
L'etiqueta. Això es traduirà. (Requerida)

**<set>**

Estableix una propietat a un valor fix (per descomptat, si connecteu addicionalment la propietat a alguna altra propietat, el valor no es manté fix). Per exemple, si incrusteu un connector, però voleu ocultar alguns dels seus elements, podeu establir la propietat de visibilitat d'aquests elements a fals. Especialment útil per a connectors incrustats/incrustants. Nota: si hi ha diversos elements <set> per a un únic `id`, l'últim a definir tindrà prioritat. Això de vegades serà útil per a confiar quan s'usin les parts <include>. Atributs:

***id***
L'ID de la propietat a establir (requerit)

***to***
El valor de la cadena a establir la propietat (requerit). Nota: Per a les propietats booleanes com la visibilitat, l'habilitació, normalment establireu l'atribut a «to="true"» o a «to="false"».

**<convert>**

Crea una propietat booleana nova que depèn de l'estat d'una o més propietats diferents. Atributs:

***id***
L'ID de la propietat nova (requerit)

***sources***
Els identificadors de les propietats de les quals dependrà aquesta propietat. Es poden especificar una o més propietats, separades per «;» (requerit)

***mode***
El mode de conversió/operació. Un d'entre "equals", "notequals", "range", "and", "or". Si és en mode "equals", la propietat només serà certa si el valor de totes les seves fonts és igual que l'estàndard de l'atribut (vegeu a sota). Si està en el mode "notequals", la propietat només serà certa si el valor de totes les seves fonts són diferents de l'estàndard de l'atribut (vegeu a sota). Si està en mode "range", les fonts han de ser numèriques (entera o real). La propietat només serà certa, si totes les fonts estan en l'interval especificat pels atributs mín i màx. Si està en mode "and", les fonts han de ser propietats booleanes. La propietat només serà certa, si totes les fonts són certes simultàniament. Si està en mode "or", les fonts han de ser propietats booleanes. La propietat només serà certa, si almenys una de les fonts és certa. (Requerit)

***standard***
Només té sentit en els modes "equal" o "notequals": el valor de la cadena contra el qual comparar (requerit si està en un d'aquests modes)

***min***
Només té sentit en el mode ″range″: el valor mínim contra el qual comparar (opcional, predeterminat al número de coma flotant més baix representable a la màquina)

***max***
Només té sentit en el mode ″range″: el valor màxim contra el qual comparar (opcional, predeterminat al número de coma flotant més gran representable a la màquina)

***require_true***
Si s'estableix a ″true″, la propietat serà obligatòria, i només es considerarà vàlida si el seu estat és cert/activat. Per tant, si la propietat és falsa, bloquejarà el botó **Envia** (opcional, el valor predeterminat és ″false″).

> ATENCIÓ
> Si utilitzeu això, assegureu-vos que l'usuari pot detectar fàcilment el què està malament, com ara mostrar un <text> d'explicació.

**<switch>**
Crea una propietat nova que retransmetrà a diferents propietats de destinació (o cadenes fixes) en funció del valor d'una propietat de condició. Això permet crear una lògica similar a les construccions `if()` o `switch()`. Atributs:

***id***
L'ID de la propietat nova (requerit)

***condition***
L'identificador de la propietat de condició (requerit)

Elements fills:

**<true>**
Si la propietat condició és booleana, podeu especificar els dos elements fills <true> i <false> (i només aquests). (Requerit, si també es proporciona <false>)

**<false>**
Si la propietat condició és booleana, podeu especificar els dos elements fills <true> i <false> (i només aquests). (Requerit, si també es proporciona <true>)

**<case>**
Si la propietat condició no és booleana, podeu proporcionar un nombre arbitrari d'elements <case>, un per a cada valor de la propietat condició que voleu que coincideixi (almenys es requereix un d'aquests elements, si la propietat condició no és booleana)

**<default>**
Si la propietat condició no és booleana, l'element opcional <default> permet especificar el comportament, si no hi ha cap element <case> que coincideixi amb el valor de la propietat condició (opcional, permès només una vegada, en combinació amb un o més elements <case>).

Els elements fills <true>, <false>, <case>, i <default> prenen els atributs següents:

***standard***
Només per als elements <case>: el valor amb el qual coincidir la propietat condició (requerit, cadena).

***fixed_value***
Una cadena fixa que s'ha de proporcionar com el valor de la propietat <switch>, si la condició actual coincideix (requerit, si no es proporciona dynamic_value).

***dynamic_value***
L'*id* de la propietat de destinació que s'ha de proporcionar com el valor de la propietat <switch>, si la condició actual coincideix (requerit, si no es proporciona fixed_value).

**<connect>**

Connecta dues propietats. La propietat del client es canviarà sempre que la propietat del governador canviï (però no a l'inrevés). Atributs:

***client***
L'ID de la propietat del client, és a dir, la propietat que s'ajustarà (requerit)

***governor***
L'ID de la propietat del governador, és a dir, la propietat que ajustarà la propietat del client. Això pot incloure un modificador (requerit)

***reconcile***
Si és «true», la propietat del client ajustarà la propietat del governador en la connexió de tal manera que la propietat del governador només acceptarà valors que també siguin acceptables pel client (p. ex., suposem que el governador és una propietat numèrica amb valor mínim «0», i el client és una propietat numèrica amb valor mínim «100». El mínim d'ambdues propietats s'ajustarà a 100, si reconcile="true"). Generalment només funciona per a propietats del mateix tipus bàsic (opcional, predeterminat a "false")

**<dependency_check>**

Crea una propietat booleana que és certa, si es compleixen les dependències especificades, falsa en cas contrari. La sintaxi XML de l'element és la mateixa que la de l'element **<dependencies>**, descrit en la referència del `.pluginmap`. A partir del RKWard 0.6.1, només es tindran en compte les especificacions de versió del RKWard i R, no les dependències en els paquets o els `.pluginmap`.

**<script>**

Defineix el codi de script per a controlar la lògica de la IU. Vegeu la secció sobre la lògica de script de GUJI per a més detalls. El codi de l'script a executar es pot donar utilitzant l'atribut *"file"*, o com un text (comentat) de l'element. L'element **<script>** no està permès a la secció **<logic>** d'un «optionset». Atributs:

***file***
Nom de fitxer del fitxer de script. (Requerit)

## A.4 Propietats dels elements del connector

Tots els elements de disposició, i tots els elements actius tenen les propietats següents, accessibles a través de "id_of_element.name_of_property":

**visible**
Si l'element de la IGU és visible o no (booleà)

**enabled**
Si l'element de la IGU està habilitat o no (booleà)

**required**
Si es requereix o no l'element de la IGU (per a mantenir una configuració vàlida). Tingueu en compte que qualsevol element que estigui desactivat o ocult també és implícitament no requerit (booleà).

A més d'això, alguns elements tenen propietats addicionals a les quals es pot connectar. La majoria dels elements actius també tenen una propietat "default", el valor de la qual es retornarà en les crides a `getBoolean/getString/getList ("...")`, si no s'ha nomenat cap propietat específica, com es descriu a continuació.

**<text>**

La propietat predeterminada és text

**text**
El text mostrat (text)

**<varselector>**

Sense propietat predeterminada

**selected**
Els objectes actualment seleccionats. Probablement, no voldreu utilitzar això. Usat internament (RObject)

**root**
L'objecte arrel/pare dels objectes oferts per a la selecció (RObject)

**<varslot>**

La propietat predeterminada és ″available″ (disponible)

**available**
Tots els objectes continguts en el «varslot» (RObject)

**selected**
Dels objectes que hi ha al «varslot», els que estan seleccionats actualment. Probablement, no voldreu utilitzar això. Usat internament (RObject)

**source**
Una còpia dels objectes seleccionats en el «varselector» corresponent. Probablement no volen utilitzar això. Usat internament (RObject)

**<valueselector>**

La propietat predeterminada és ″selected″

**selected**
Les cadenes actualment seleccionades. Modificador ″labeled″ per a recuperar les etiquetes corresponents. En un <valueselector> probablement no el voldreu utilitzar directament (només en un <select>). (Llegeix/escriu StringList)

**available**
La llista de valors de cadena de la qual seleccionar. (Llegeix/escriu StringList)

**labels**
Etiquetes a mostrar per als valors de cadena. (Llegeix/escriu StringList)

**<valueslot>**

El mateix que <varslot>, però les propietats són llistes de cadenes, en lloc de RObjects.

**<radio>**

La propietat predeterminada és ″string″

**string**
El valor de l'opció actualment seleccionada (cadena)

**number**
El nombre de l'opció actualment seleccionada (les opcions estan numerades de dalt a baix, començant en el 0) (enter)

**<dropdown>**

El mateix que <radio>

**<select>**

El mateix que <valueselector>

**<option>**

No hi ha cap propietat per defecte. ″enabled″ és la *única* propietat, i actualment no està disponible per a opcions dins de <select> o <valueselector>. <option> no té les propietats ″visible″ o ″required″.

**enabled**
Si s'ha d'activar o desactivar aquesta opció única. En la majoria dels casos activareu/desactivareu tot el <radio< o <dropdown<. Però això es pot utilitzar per a establir dinàmicament l'activació d'una opció única dins d'un <radio< o <dropdown< (booleà)

**<checkbox>**

La propietat predeterminada és ″state.labeled″, el qual vol dir que es retornen els valors especificats pels atributs `value`, i `value_unchecked`, *no* l'etiqueta mostrada de la casella de selecció.

**state**
Estat de la casella de selecció (activada o desactivada). Tingueu en compte que els modificadors útils d'aquesta propietat (com totes les propietats booleanes) són ″not″ i ″labeled″ (vegeu tipus de propietats). No obstant això, sovint és més útil connectar a la propietat sense modificador, és a dir, *″checkbox_id*.state″, el qual retornarà l'estat de la casella de selecció en un format adequat per al seu ús en una declaració ″if″ (0 o 1). (Booleà)

**<frame>**

La propietat predeterminada és ″checked″, si (i només si) el marc es pot marcar. Per als marcs que no es poden marcar, no hi ha cap propietat per defecte.

**checked**
Disponible només per als marcs que es poden marcar: estat de la casella de selecció (activada o desactivada). Tingueu en compte que els modificadors útils d'aquesta propietat (com totes les propietats booleanes) són ″not″ i ″numeric″ (vegeu tipus de propietats). (Booleà)

**<input>**

La propietat predeterminada és ″text″

**text**
Text actual en el camp d'entrada (cadena)

**<matrix>**

La propietat predeterminada és ″cbind″.

**rows**
Nombre de files a la matriu (enter). Si la matriu permet a l'usuari afegir/eliminar files, aquesta propietat s'ha de tractar com de només lectura. En cas contrari, canviant-la, es canviarà la mida de la matriu.

**columns**
Nombre de columnes a la matriu (enter). Si la matriu permet a l'usuari afegir/eliminar columnes, aquesta propietat s'ha de tractar com de només lectura. En cas contrari, canviant-la, es canviarà la mida de la matriu.

**tsv**
Dades en la matriu en format «tsv» (cadena; lectura-escriptura). Tingueu en compte que en comparació amb la disposició «tsv» habitual, les *columnes*, no files, estan separades per caràcters de línia nova, i les cel·les dins d'una columna estan separades per caràcters tabuladors.

**0,1,2...**
Les dades d'una sola columna (0 per a la columna més a l'esquerra). `getValue()`/`getString()` retorna això com una sola cadena, separada per «\n». No obstant això, la manera recomanada d'obtenir-ho és utilitzant `getList()`, que retorna aquesta columna com una matriu de cadenes.

**row.0,row.1,row.2...**
: Les dades d'una sola fila (0 per a la fila superior). `getValue()/getString()` retorna això com una sola cadena, separada per «\n». No obstant això, la manera recomanada d'obtenir-ho és utilitzant `getList()`, que retorna aquesta fila com una matriu de cadenes.

**cbind**
: Dades en un format adequat per a enganxar a R, embolcallades en una expressió «cbind» (cadena; només lectura).

**<optionset>**

No hi ha cap propietat predeterminada.

**row_count**
: Nombre d'elements a l'«optionset» (enter). Només lectura.

**current_row**
: Element actualment actiu a l'«optionset» (enter). -1 per a cap element actiu. Lectura i escriptura.

***optioncolumn_ids***
: Per a cada <optioncolumn> que definiu, es crearà una propietat de llista de cadenes amb l'identificador especificat.

**<browser>**

La propietat predeterminada és ″selection″

**selection**
: Text actual (nom de fitxer seleccionat) al navegador (cadena)

**<saveobject>**

La propietat predeterminada és ″selection″

**selection**
: Nom complet de l'objecte seleccionat (cadena; només lectura, per a establir-ho des del programa utilitzeu ″parent″ i ″objectname″)

**parent**
: L'objecte pare de l'objecte seleccionat. Aquest sempre és un objecte R existent d'un tipus que pot contenir altres objectes (p. ex., una llista o un «data.frame»). Quan s'estableix a una cadena buida o a un objecte no vàlid, s'assumeix ″.GlobalEnv″ (RObject)

**objectname**
: El nom base de l'objecte seleccionat, és a dir, la cadena introduïda per l'usuari (canviada a un nom R vàlid, si cal) (cadena)

**active**
: Només per a «saveobject» que es puguin marcar: si el control està activat/activat. Sempre cert per als «saveobject» que no es poden marcar (booleà)

**<spinbox>**

La propietat predeterminada és ″int″ o ″real.formatted″, depenent del mode del botó de selecció de valors

**int**
: Valor enter que té el botó de selecció de valors, o enter més proper, si està en mode real (enter)

**real**
: Valor real que té el botó de selecció de valors (o enter, si és enter) (real)

**<formula>**

La propietat predeterminada és ″model″

**model**
La cadena del model actual (cadena)

**table**
El «data.frame» conté les variables requerides. Si només s'utilitzen variables d'un «data.frame», es retornarà el nom d'aquest «data.frame». Altrament es construeix un «data.frame» nou segons sigui necessari (cadena)

**labels**
Si estan implicades les variables de diversos «data.frames», els seus noms es poden barrejar (per exemple, si ambdós «data.frames» contenen una variable anomenada «x»). Això retorna una llista amb els noms entrellaçats com a índexs i l'etiqueta descriptiva com a valor (cadena)

**fixed_factors**
Els factors fixos. Probablement no voldreu utilitzar això. Usat internament (RObject)

**dependent**
Les variables dependents. Probablement no voldreu utilitzar això. Usat internament (RObject)

**<embed>**
Sense propietat predeterminada

**code**
El codi generat pel connector incrustat (codi)

**<preview>**
La propietat predeterminada és ″state″

**state**
Si la casella de previsualització està marcada (no necessàriament si ja s'ha mostrat la previsualització) (booleà)

**<convert>**
Aquest element (utilitzat a la secció <logic>) és especial, ja que tècnicament *és* una propietat, en lloc de només tenir una o més propietats. És de tipus booleà. Tingueu en compte que els modificadors útils d'aquesta propietat (com totes les propietats booleanes) són ″not″ i ″numeric″ (vegeu tipus de propietats)

**<switch>**
Aquest element (utilitzat a la secció <logic>) és especial, ja que tècnicament *és* una propietat (cadena), en lloc de només tenir una o més propietats. Permet canviar entre diverses propietats de destinació depenent del valor d'una propietat de condició, o per a tornar a assignar els valors de la propietat de condició. Qualsevol modificador que proporcioneu es transmet a les propietats de destinació, per tant, p. ex., si totes les propietats de destinació són propietats RObject, també podreu utilitzar el modificador ″shortname″ al commutador. No obstant això, si les propietats de destinació són de tipus diferents, l'ús de modificadors pot provocar errors. Pels `fixed_value`, s'elimina qualsevol modificador, en silenci. Tingueu en compte que les propietats de destinació, quan s'accedeix a través d'un commutador, sempre són de només lectura.

## A.5 Connectors incrustables distribuïts amb la versió oficial del RKWard

Amb el RKWard es distribueixen una sèrie de connectors incrustables, i es poden utilitzar en els vostres propis connectors. Actualment, la documentació detallada només està disponible en aquests fitxers de codi font o d'ajuda dels connectors. No obstant això, aquí hi ha una llista per a donar-vos una visió ràpida del que hi ha disponible:

| ID | Pluginmap | Descripció | Exemple d'ús |
|---|---|---|---|
| rkward::plot_options | embedded.pluginmap | Proporciona una gran varietat d'opcions per als diagrames. La majoria dels connectors de traçat utilitzen això. | Diagrames->Diagrama de barres, la majoria dels altres connectors de traçat |
| rkward::color_chooser | embedded.pluginmap | Connector molt senzill per a especificar un color. La implementació actual proporciona una llista de noms de color. Les implementacions futures poden proporcionar una tria de colors més elaborada. | Diagrames->Histograma |
| rkward::plot_stepfun_options | embedded.pluginmap | Opcions de diagrama amb funció esglaonada | Diagrames->Diagrama ECDF |
| rkward::histogram_options | embedded.pluginmap | Opcions de l'histograma (diagrama) | Diagrames->Histograma |
| rkward::barplot_embed | embedded.pluginmap | Opcions del diagrama de barres | Diagrames->Diagrama de barres |
| rkward::one_var_tabulation | embedded.pluginmap | Proporciona tabulació en una variable única. | Diagrames->Diagrama de barres |
| rkward::limit_vector_length | embedded.pluginmap | Limita la longitud d'un vector (als n elements més grans o més petits). | Diagrames->Diagrama de barres |
| rkward::level_select | embedded.pluginmap | Proporciona un <valueselector> ple amb els nivells (o valors únics) d'un vector. | Dades->Recodifica dades categòriques |
| rkward::multi_input | embedded.pluginmap | Combina els botons de selecció de valors, entrada i control d'opcions per a proporcionar entrada de dades de caràcters, numèriques i lògiques. | Dades->Recodifica dades categòriques |

Taula A.1: Connectors incrustables estàndards

## A.6 Elements que s'utilitzaran en els fitxers `.pluginmap`

**<document>**

Cal que estigui present a cada fitxer `.pluginmap` com a node arrel (exactament una vegada). Atributs:

**base_prefix**

Els noms de fitxer especificats al fitxer `.pluginmap` s'assumeixen que són relatius al directori del fitxer `.pluginmap` + el prefix que especifiqueu aquí. Especialment útil si tots els vostres components es troben sota un únic subdirectori.

**namespace**

Un espai de noms ("namespace") per als identificadors dels components. Quan se cerquin components per a incrustar-los, els components es podran recuperar mitjançant una cadena "namespace::component_id". Establert a "rkward" per ara.

**id**

Una cadena d'identificador opcional per a aquest `.pluginmap`. Especificar això permet als autors tercers referir-se i carregar el vostre `.pluginmap` des del seu (vegeu el capítol sobre la gestió de les dependències).

**priority**

Un d'entre `"hidden"`, `"low"`, `"medium"`, o `"high"`. Els `.pluginmap` amb prioritat «medium» o «high» s'activen automàticament quan el RKWard els troba per primera vegada. Utilitzeu `priority="hidden"` per als `.pluginmap` que no estan destinats a ser activats, el directori (només per a la inclusió). En la implementació actual això no oculta realment el `.pluginmap`. (Opcional, el valor predeterminat és "medium").

**<dependencies>**

Aquest element, especificant dependències, es permet com a fill directe de l'element <document> (un cop), i com a fill dels elements <component> (un cop per a cada element <component>). Especifica les dependències que s'han de complir per a utilitzar els connectors. Consulteu el capítol sobre dependències per a una visió general. Atributs:

**rkward_min_version, rkward_max_version**

Versió mínima i màxima permesa del RKWard. Les especificacions de versió poden incloure sufixos no numèrics, com "0.5.7z-devel1". Si no es compleix una dependència especificada, el/s connector/s al/s que s'aplica *s'ignorarà*. Més informació. Opcional; si no s'especifica, no es requerirà cap versió mínima/màxima del RKWard.

**R_min_version, R_max_version**

Versió mínima i màxima permesa de l'R. Les especificacions de versió poden *no* incloure sufixos no numèrics, com "0.5.7z-devel1". La dependència de la versió de l'R es mostrarà a les pàgines d'ajuda dels connectors, però no té cap efecte directe, a partir del RKWard 0.6.1. Més informació. Opcional; si no s'especifica, no es requerirà cap versió mínima/màxima de l'R.

Elements fills:

**<package>**

Afegeix una dependència d'un paquet R específic. Atributs:

**name**
Nom del paquet (requerit).

**min_version, max_version**
Versió mínima/màxima permesa (opcional).

**repository**
Dipòsit on es troba el paquet. Opcional, però molt recomanat, si el paquet no està disponible al CRAN.

**<pluginmap>**

Afegeix una dependència d'un `.pluginmap` específic del RKWard. Atributs:

**name**
: Cadena d'«id» del connector `.pluginmap` requerit (requerit).

**min_version, max_version**
: Versió mínima/màxima permesa (opcional).

**url**
: URL on es pot trobar el `.pluginmap`. Requerit.

**<about>**

Pot estar present exactament una vegada com a fill directe de l'element <document>. Conté la metainformació sobre el `.pluginmap` (o connector). Consulteu el capítol sobre la informació «about» per a obtenir una visió general. Atributs:

**name**
: Nom visible de l'usuari. Opcional. No ha de ser el mateix que ″id″.

**version**
: Número de versió. Opcional. El format no està restringit, però per a estar sobre segur, seguiu esquemes de versions habituals com ara ″x.y.z″.

**releasedate**
: Especificació de data de publicació. Opcional en format ″AAAA-MM-DD″.

**shortinfo**
: Una descripció *curta* del connector/`.pluginmap`. Opcional.

**url**
: URL a on es pot trobar més informació. Opcional, però recomanat.

**copyright**
: Especificació del copyright, p. ex., ″2012-2013 de John Doe″. Opcional, però recomanat.

**licence**
: Especificació de la llicència, p. ex., «GPL» o «BSD». Assegureu-vos d'acompanyar els fitxers amb una còpia completa de la llicència corresponent. Opcional, però recomanat.

**category**
: Categoria del/s connector/s, p. ex., «Teoria de resposta d'elements». A partir del RKWard 0.6.1, no hi ha categories predefinides. Opcional.

Elements fills:

**<author>**

Afegeix informació sobre un autor. Atributs:

**name, given, family**
: Especifiqueu el nom complet per al *`name`*, o especifiqueu ambdós *`given`* i *`family`*, per separat.

**role**
: Descripció del rol de l'autor (opcional).

**email**
: L'adreça de correu electrònic on es pot contactar amb l'autor. Requerida. Es pot establir a la llista de correu rkward-devel, si esteu subscrit, i el vostre connector està destinat a ser inclòs en la versió oficial del RKWard.

**url**
: L'URL amb més informació sobre l'autor, p. ex., la pàgina web (opcional).

**<components>**

Necessita estar present exactament una vegada com a fill directe de l'element <document>. Conté els elements individuals <component> descrits a continuació. Sense atributs.

**<component>**

Un o més elements <component> s'han de donar com a fills directes de l'element <components> (i només allà). Registra un component/plugin amb el «rkward». Atributs:

**type**
: Per a futures extensions: el tipus de component/connector. S'estableix sempre a «standard» per ara (l'únic tipus admès actualment).

**id**
: L'ID pel qual es pot recuperar aquest component (per a col·locar-lo al menú (vegeu a sota), o per a incrustar). Vegeu l'espai de noms <document> a dalt.

**file**
: Requerit almenys pels components de type="standard": el nom del fitxer XML que descriu la IGU.

**label**
: L'etiqueta d'aquest component, quan es col·loca a la jerarquia del menú.

**<attribute>**

Defineix un atribut d'un component. Només té sentit per a connectors d'importació fins ara. Només es permet com a fill directe de <component>. Atributs:

**id**
: Id de l'atribut

**value**
: Valor de l'atribut

**labels**
: Etiqueta associada amb l'atribut

**<hierarchy>**

Necessita estar present exactament una vegada com a fill directe de l'element <document>. Descriu on s'han de col·locar els components declarats a dalt a la jerarquia del menú. Accepta només elements <menu> com a fills directes. Sense atributs.

**<menu>**

Un o més elements <menu> s'han de donar com a fills directes de l'element <hierarchy>. Declara un (sub)menú nou. Si ja existeix un menú amb l'ID indicat (vegeu més avall), els dos menús es fusionen. Es permet l'element <menu> com a fill directe de l'element <hierarchy> (menú de nivell superior), o com a fill directe en qualsevol altre element <menu> (menú inferior). Per contra, l'element <menu> accepta altres elements <menu> o <entry> com a elements fills. Atributs:

**id**
: Una cadena d'identificació del menú. Útil quan les definicions del menú es llegeixen des de diversos fitxers del `.pluginmap`, per a assegurar-se que els connectors es poden col·locar en el mateix menú. Alguns identificadors de menú com ara «file» es refereixen a menús predefinits (en aquest cas el menú «File»). Assegureu-vos de comprovar amb els fitxers existents del `.pluginmap` per a utilitzar ID coherents.

**label**
: Una etiqueta per al menú.

**group**
: Permet controlar l'ordenació de les entrades del menú. Vegeu ordenació d'elements del menú. Opcional.

**<entry>**

Una entrada de menú, és a dir, una opció de menú per a invocar un connector. Només es pot utilitzar com a fill directe d'un element <menu>, no accepta elements fills. Atributs:

**component**
: L'ID del component que s'ha d'invocar quan s'activa aquesta entrada del menú.

**group**
Permet controlar l'ordenació de les entrades del menú. Vegeu ordenació d'elements del menú. Opcional.

**<group>**
Declara un grup d'elements al menú. Vegeu ordenació d'elements del menú. Atributs:

**id**
El nom d'aquest grup.

**separated**
Opcional. Si s'estableix a «true» (cert), l'element d'aquest grup se separarà visualment dels elements circumdants.

**group**
El nom del grup al qual afegir aquest grup (opcional).

**<context>**
Declara les entrades en un context. Només es permet com a fill directe de l'etiqueta <document>. Només accepta etiquetes <menu> com a filles directes. Atributs:

**id**
L'ID del context. Fins ara només s'han implementat dos contextos: ″x11″ i ″import″.

**<require>**
Incloure un altre fitxer `.pluginmap`. Aquest fitxer `.pluginmap` només es carregarà una vegada, encara que sigui <require> des de diversos fitxers. El cas d'ús més important és incloure un fitxer «pluginmap», el qual declara alguns components que estan incrustats pels components declarats en aquest `.pluginmap`. Només es permeten els elements <require> com a fills directes del node <document>. Atributs:

**file**
El nom del fitxer del `.pluginmap` a incloure. Això es veu en relació amb el directori del fitxer `.pluginmap` actual + el base_prefix (vegeu més amunt l'element <document>). Si no coneixeu el camí relatiu al `.pluginmap` que s'ha d'incloure, utilitzeu l'atribut *`map`* per a referir-vos a ell per ID.

**map**
Per a incloure un fitxer `.pluginmap` des d'un paquet diferent (o un `.pluginmap` del RKWard des del vostre `.pluginmap` extern), podeu referir-vos a ell pel seu *`namespac ename::id`*, com s'especifica en l'element <document> necessari del `.pluginmap`. La inclusió fallarà si no es coneix cap `.pluginmap` per aquest identificador (p. ex., no està instal·lat en el sistema de l'usuari). Hauríeu d'utilitzar aquest mètode per a incloure els `.pluginmap` fora del paquet, només. Per als mapes dins del paquet, especificar un camí relatiu (l'atribut *`file`*) és més ràpid i més fiable.

## A.7 Elements per a utilitzar en fitxers .rkh (ajuda)

**<document>**
Necessita estar present a cada fitxer `.xml` com a node arrel (exactament una vegada). Sense atributs.

**<title>**
Títol de la pàgina d'ajuda. Això *no* s'interpreta per a pàgines d'ajuda d'un connector (això pren el títol del mateix connector), només per a pàgines independents. Sense atributs. El text que conté l'etiqueta <title> es convertirà en la llegenda de la pàgina d'ajuda. Només es pot definir una vegada, com a fill directe del node <document>.

**<summary>**

Un breu resum de la pàgina d'ajuda (o per a què s'utilitza aquest connector). Això sempre es mostrarà a la part superior de la pàgina d'ajuda. Sense atributs. Es mostrarà el text contingut dins de l'etiqueta <summary>. Recomanat però no necessari. Només es pot definir una vegada, com a fill directe del node <document>.

**<usage>**

Un resum una mica més elaborat de l'ús. Això sempre es mostrarà directament després de <summary>. Sense atributs. Es mostrarà el text contingut dins de l'etiqueta <usage>. Recomanat per a les pàgines d'ajuda del connector, però no és necessari. Només es pot definir una vegada, com a fill directe del node <document>.

**<section>**

Secció de propòsits generals. Es pot utilitzar qualsevol nombre de vegades com a fill directe del node <document>. Aquestes seccions es mostren en l'ordre de la seva definició, però totes les *després* de la secció <usage> i *abans* de la secció <settings>. Es mostrarà el text contingut dins de l'etiqueta <section>.

**id**

Un identificador necessari per a saltar a aquesta secció des de la barra de navegació (o un enllaç). Cal que sigui únic dins del fitxer. Requerit, sense valor predeterminat.

**title**

El títol (llegenda) d'aquesta secció. Requerit, sense valor predeterminat.

**short_title**

Un títol curt adequat per a mostrar a la barra de navegació. Opcional, el valor predeterminat és el títol complet.

**<settings>**

Defineix la secció que conté la referència sobre les diverses opcions de la IGU. Només té sentit i només s'utilitza per a les pàgines d'ajuda relacionades amb els connectors. Utilitzeu-ho com a fill directe del <document>. Pot contenir només elements <setting> i <caption> com a fills directes. Sense atributs.

**<setting>**

Explica una configuració única a la IGU. Només es permet com a fill directe de l'element <settings>. Es mostra el text contingut dins de l'element.

**id**

L'ID del paràmetre en el `.xml` del connector. Requerit, sense valor predeterminat.

**title**

Un títol opcional per a la configuració. Si s'omet (es recomana l'omissió en la majoria dels casos), el títol es prendrà del `.xml` del connector.

**<caption>**

Una llegenda per a agrupar visualment diversos paràmetres. Només es pot utilitzar com a fill directe de l'element <settings>.

**id**

L'ID de l'element corresponent (normalment un <frame>, <page> o <tab>) en el `.xml` del connector.

**title**

Un títol opcional per a la llegenda. Si s'omet (es recomana l'omissió en la majoria dels casos), el títol es prendrà del `.xml` del connector.

**<related>**

Defineix una secció que conté enllaços a altra informació relacionada. Sempre es mostrarà després de la secció <settings>. Sense atributs. Es mostrarà el text contingut dins de l'etiqueta <related>. Normalment, això contindrà una llista d'estil HTML. Recomanat per a les pàgines d'ajuda del connector, però no és necessari. Només es pot definir una vegada, com a fill directe del node <document>.

**<technical>**

Defineix una secció que conté informació tècnica sense rellevància per als usuaris finals (com l'estructura interna del connector). Sempre es mostrarà l'últim en una pàgina d'ajuda. Sense atributs. Es mostrarà el text contingut dins de l'etiqueta <related>. No és necessari i no es recomana per a la majoria de les pàgines d'ajuda del connector. Només es pot definir una vegada, com a fill directe del node <document>.

**<link>**

Un enllaç. Es pot utilitzar en qualsevol de les seccions descrites anteriorment.

**href**

L'URL de destinació. Tingueu en compte que hi ha disponibles diversos URL específics del RKWard. Vegeu la secció sobre l'escriptura de les pàgines d'ajuda per a més detalls.

**<label>**

Insereix el valor d'una etiqueta d'interfície d'usuari. Es pot utilitzar en qualsevol de les seccions descrites anteriorment.

**id**

L'«id» de l'element en el connector, del qual copiar l'atribut `label`.

**<etiquetes HTML diverses>**

Les etiquetes HTML més bàsiques estan permeses dins de les seccions. No obstant això, manteniu la formatació manual al mínim.

## A.8 Funcions disponibles per a la creació de scripts de lògica de la IGU

**Classe «Component»**

Classe que representa un únic component o component-propietat. La instància més important d'aquesta classe és la variable "gui" que està predefinida com a propietat arrel del component actual. Hi ha disponibles els mètodes següents per a exemples de la classe «Component»:

**absoluteId(base_id)**

Retorna l'ID absolut de *base_id*, o, si s'omet base_id, l'identificador del component.

**getValue(id)**

Es descoratja. Utilitzeu `getString(), getBoolean() o getList()` en el seu lloc. Retorna el valor de la propietat filla donada. Retorna el valor d'aquesta propietat, si s'omet l'ID.

**getString(id)**

Retorna el valor de la propietat filla donada com a cadena. Retorna el valor d'aquesta propietat, si s'omet l'ID.

**getBoolean(id)**

Retorna el valor de la propietat filla donada com a booleà (si és possible). Retorna el valor d'aquesta propietat, si s'omet l'ID.

**getList(id)**

Retorna el valor de la propietat filla donada com una matriu de cadenes (si és possible). Retorna el valor d'aquesta propietat, si s'omet l'ID.

**setValue(id, valor)**

Estableix el valor de la propietat filla donada a *valor*.

**getChild(id)**
Retorna una instància de la propietat filla amb l'*id* donat.

**addChangeCommand(id, ordre)**
Executa l'*ordre* sempre que canviï la propietat filla donada per *id*.

**Classe «RObject»**

Classe que representa un únic objecte R. Es pot obtenir una instància d'aquesta classe utilitzant **makeRObject(objectname)**. Hi ha disponibles els mètodes següents per a exemples de la classe ″RObject″:

> AVÍS
> Si hi ha ordres pendents al dorsal, la informació proporcionada per aquests mètodes pot estar desactualitzada en el moment en què s'executi el codi del connector. *No* confieu en ell per a operacions crítiques (s'està arriscant la pèrdua de dades).

**getName()**
Retorna el nom absolut de l'objecte.

**exists()**
Retorna si l'objecte existeix. Haureu de comprovar-ho abans d'utilitzar qualsevol dels mètodes llistats a continuació.

**dimensions()**
Retorna una matriu de dimensions (similar a **dim()** en l'R).

**classes()**
Retorna una matriu de classes (similar a **class()** en l'R).

**isClass(classe)**
Retorna «true» (cert), si l'objecte és de classe *class*.

**isDataFrame()**
Retorna «true» (cert), si l'objecte és un «data.frame».

**isMatrix()**
Retorna «true» (cert), si l'objecte és una matriu.

**isList()**
Retorna «true» (cert), si l'objecte és una llista.

**isFunction()**
Retorna «true» (cert), si l'objecte és una funció.

**isEnvironment()**
Retorna «true» (cert), si l'objecte és un entorn.

**isDataNumeric()**
Retorna «true» (cert), si l'objecte és un vector de dades numèriques.

**isDataFactor()**
Retorna «true» (cert), si l'objecte és un vector de dades de factor.

**isDataCharacter()**
Retorna «true» (cert), si l'objecte és un vector de dades de caràcters.

**isDataLogical()**
Retorna «true» (cert), si l'objecte és un vector de dades lògiques.

**parent()**
Retorna una instància de «RObject» que representa el pare d'aquest objecte.

**child(nomfill)**
Retorna una instància de «RObject» que representa el *nom del fill* del fill d'aquest objecte.

**Classe «RObjectArray»**

Una matriu d'instàncies RObject. Una instància d'aquesta classe es pot obtenir utilitzant **makeRObjectArray(objectnames)**. És particularment útil quan es tracta de «varslots» que permeten seleccionar diversos objectes.

**include()-function**

**include(filename)** es pot utilitzar per a incloure un fitxer JS separat.

**doRCommand()-function**

**doRCommand(command, callback)** es pot utilitzar per a consultar l'R per a obtenir informació. Llegiu la secció sobre consultes R des de dins d'un connector per a més detalls i advertències.

# Apèndix B

# Resolució de problemes durant el desenvolupament del connector

Així que heu llegit tota la documentació, ho heu fet tot bé i encara no podeu fer-ho funcionar? No us preocupeu, ho resoldrem. El primer que cal fer és: activar la finestra **Missatges de depuració del RKWard** (disponible des del menú **Finestres** -, o fer clic dret sobre una de les barres d'eines), i després iniciar el connector, de nou. Com a regla general, no hauríeu de veure cap sortida a la finestra de missatges quan s'invoqui el connector, o en qualsevol altre moment. Si n'hi ha una, probablement està relacionada amb la vostra extensió. Mireu si us ajuda.

Si tot sembla estar bé a la consola, intenteu augmentar el nivell de depuració (des de la línia d'ordres, utilitzant **rkward --debug-level 3**, o establint el nivell de depuració a 3 en **Arranjament** → **Configura el RKWard** → **Depuració**). No tots els missatges mostrats a nivells de depuració més alts indiquen necessàriament un problema, però hi ha possibilitats que el problema es mostri en algun lloc entre els missatges.

Si encara no es pot esbrinar què és el que està malament, no espereu. Sabem que això és complicat, i després de tot, possiblement també us heu trobat amb un error en el RKWard, i el RKWard necessita ser corregit. Escriviu a la llista de correu de desenvolupament i expliqueu-nos el problema. Estarem encantats d'ajudar-vos.

Finalment, fins i tot si heu descobert com fer-ho pel vostre compte, però heu trobat que la documentació no és tan útil o fins i tot equivocada en alguns aspectes, digueu-nos-ho també a la llista de correu, perquè puguem arreglar/millorar la documentació.

# Apèndix C

# Llicència

Traductor de la documentació: Josep M. Ferrer txemaq@gmail.com