

Introduzione alla scrittura delle estensioni per RKWard

Thomas Friedrichsmeier

Meik Michalke

Traduzione italiana: Vincenzo Reale



Introduzione alla scrittura delle estensioni per RKWard

Indice

1	Introduzione	8
2	Preliminari: cosa sono le estensioni in RKWard? Come funzionano?	9
3	Creazione delle voci di menu	10
3.1	Controllo dell'ordine delle voci di menu	12
4	Definizione della GUI	14
4.1	Definire una finestra	14
4.2	Aggiunge un'interfaccia di procedura guidata	17
4.3	Alcune considerazioni sulla struttura della GUI	18
4.3.1	<radio> vs. <checkbox> vs. <dropdown>	19
5	Generare il codice di R dalle impostazioni della GUI	20
5.1	Utilizzare JavaScript nelle estensioni di RKWard	20
5.1.1	preprocess()	20
5.1.2	calculate()	21
5.1.3	printout()	22
5.2	Convenzioni, criteri e sfondo	22
5.2.1	Comprendere l'ambiente <code>local()</code>	22
5.2.2	Formattazione del codice	23
5.2.3	Gestire le operazioni complesse	23
5.3	Suggerimenti e trucchi	24
6	Scrivere una pagina della guida	25
7	Interazioni logiche tra gli elementi della GUI	28
7.1	Logica della GUI	28
7.2	Logica della GUI programmata	30
8	Integrazione di estensioni in estensioni	32
8.1	Casi di utilizzo per l'integrazione	32
8.2	Integrazione in una finestra	32
8.3	Generazione di codice durante l'integrazione	33
8.4	Integrazione in una procedura guidata	33
8.5	Integrazione meno integrata: pulsante Altre opzioni	34
8.6	Integrazione/definizione di estensioni incomplete	34

9	Gestire molte estensioni singole	36
9.1	Panoramica su diversi approcci	36
9.2	Utilizzare l'istruzione include di JS	36
9.3	Includere i file .xml	37
9.4	Utilizzare <snippets>	38
9.5	<include> e <snippets> vs. <embed>	39
10	Concetti per l'uso in estensioni specializzate	41
10.1	Estensioni che producono un grafico	41
10.1.1	Tracciare un grafico nella finestra del risultato	41
10.1.2	Aggiungere una funzionalità di anteprima	41
10.1.3	Opzioni generiche grafico	42
10.1.4	Un esempio classico	43
10.2	Anteprime dei dati, uscita e altri risultati	44
10.2.1	Anteprima del risultato (HTML)	44
10.2.2	Anteprima dei dati (importati)	45
10.2.3	Anteprime personalizzate	46
10.3	Estensioni dipendenti dal contesto	46
10.3.1	Contesto del dispositivo X11	47
10.3.2	Importa contesto di dati	47
10.4	Interrogazione delle informazioni di R	48
10.5	Riferimento all'oggetto attuale o al file attuale	50
10.6	Ripetere (un insieme di) opzioni	50
10.6.1	Insiemi di opzioni «guidati»	52
10.6.2	Alternative: quando non utilizzare gli insiemi di opzioni	53
11	Gestione delle dipendenze e problemi di compatibilità	54
11.1	Compatibilità di versione di RKWard	54
11.2	Compatibilità versione di R	55
11.3	Dipendenze dei pacchetti di R	56
11.4	Dipendenze da altre .pluginmap di RKWard	56
11.5	Un esempio	57
12	Traduzione delle estensioni	58
12.1	Considerazioni generali	58
12.2	i18n nei file xml di RKWard	58
12.3	i18n nei file js e nelle sezioni di RKWard.	59
12.3.1	i18n e virgolette	60
12.4	Manutenzione delle traduzioni	61
12.5	Tradurre le estensioni	61

13 Informazioni su autore, licenza e versione	62
14 Condividere il proprio lavoro con altri	64
14.1 Estensioni esterne	64
14.2 Perché le estensioni esterne?	64
14.3 Struttura del pacchetto di un'estensione	65
14.3.1 Gerarchia dei file	65
14.3.1.1 Componenti di base di un'estensione	66
14.3.1.2 Informazioni aggiuntive (facoltativo)	66
14.3.1.3 Test automatico dell'estensione (facoltativo)	67
14.4 Creazione del pacchetto dell'estensione	67
15 Sviluppo di estensioni con il pacchetto rkwarddev	68
15.1 Panoramica	68
15.2 Esempio pratico	68
15.2.1 Descrizione della GUI	69
15.2.2 Codice JavaScript	71
15.2.3 Mappa delle estensioni	73
15.2.4 Pagina della guida	73
15.2.5 Genera i file delle estensioni	73
15.2.6 Lo script completo	74
15.3 Aggiunge una pagina della guida	76
15.4 Traduzione delle estensioni	77
A Riferimento	78
A.1 Tipi di proprietà/Modificatori	78
A.2 Elementi di uso generale da utilizzare in qualsiasi file XML (.xml, .rkh, .pluginmap)	80
A.3 Elementi da utilizzare nella descrizione XML dell'estensione	80
A.3.1 Elementi generali	81
A.3.2 Definizioni dell'interfaccia	81
A.3.3 Elementi della struttura	82
A.3.4 Elementi attivi	83
A.3.5 Sezione logica	91
A.4 Proprietà degli elementi dell'estensione	94
A.5 Estensioni integrabili fornite dalla versione ufficiale di RKWard	98
A.6 Elementi da utilizzare nei file di .pluginmap	99
A.7 Elementi da utilizzare nei file .rkh (guida)	103
A.8 Funzioni disponibili per la programmazione della logica della GUI	105
B Risoluzione dei problemi durante lo sviluppo delle estensioni	108
C Licenza	109

Elenco delle tabelle

A.1 Estensioni standard integrabili	99
---	----

Sommario

Questa è una guida per scrivere estensioni per RKWard.

Capitolo 1

Introduzione

Questo documento descrive come scrivere le proprie estensioni. La documentazione è cresciuta abbastanza nel tempo. Non lasciatevi spaventare. Vi consigliamo di leggere i quattro passaggi essenziali (come indicato di seguito) per ottenere un'idea di base su come funzionano le cose. Dopo di che potreste voler scremare l'indice per vedere quali argomenti avanzati potrebbero essere rilevanti per le vostre esigenze.

Per domande e commenti, scrivere alla lista di distribuzione di sviluppo di RKWard.

Non è necessario leggere questo documento per utilizzare RKWard. Questo documento riguarda l'estensione di RKWard. È rivolto a utenti esperti o persone che desiderano contribuire a migliorare RKWard.

La scrittura di un'estensione standard è fondamentalmente un processo in quattro fasi:

- Inserimento di una nuova azione nella gerarchia dei menu
- Descrivere l'aspetto e il comportamento dell'estensione e della GUI
- Definire come deve essere generato il codice R dalle impostazioni che l'utente effettua nella GUI
- Aggiunta di una pagina della guida alla propria estensione

Queste saranno trattate a turno.

Alcuni concetti avanzati possono essere utilizzati in queste quattro fasi, ma sono trattati in capitoli separati, per semplificare le cose:

- Logica della GUI
- Integrazione di estensioni in estensioni
- Concetti utili per creare molte serie di estensioni simili

Inoltre, nessuno dei capitoli mostra tutte le opzioni, ma piuttosto solo i concetti di base. Una guida completa alle opzioni è fornita separatamente.

Capitolo 2

Preliminari: cosa sono le estensioni in RKWard? Come funzionano?

Naturalmente la prima domanda che potreste avere è: quali parti della funzionalità di RKWard sono realizzate utilizzando le estensioni? Oppure: cosa possono fare le estensioni?

Un modo per rispondere è: deselezionare tutti i file di `.pluginmap` in **Impostazioni** → **Configura RKWard** → **Estensione** e vedere cosa manca. Una risposta leggermente più utile: la maggior parte delle funzioni statistiche effettive accessibili tramite la GUI è realizzata utilizzando estensioni. Inoltre, è possibile creare GUI abbastanza flessibili per tutti i tipi di operazioni utilizzando le estensioni.

Il paradigma di base dietro le estensioni di RKWard è quello che con cui vi guideremo in questo documento: un file XML descrive l'aspetto della GUI. Un ulteriore file JavaScript viene utilizzato per generare la sintassi di R dalle impostazioni della GUI. Cioè, le estensioni non devono realmente eseguire calcoli statistici. Piuttosto le estensioni generano la sintassi di R necessaria per eseguire tali calcoli. La sintassi di R viene quindi inviata al backend di R per la valutazione e in genere un risultato viene visualizzato nella finestra di uscita.

Leggere nei prossimi capitoli per vedere come viene fatto.

Capitolo 3

Creazione delle voci di menu

Quando si crea una nuova estensione, è necessario indicarlo a RKWard. Quindi la prima cosa da fare è scrivere un file `.pluginmap` (o modificarne uno esistente). Il formato del file `.pluginmap` è XML. Vi guideremo attraverso un esempio (assicurarsi, ovviamente, di aver configurato RKWard per caricare la propria `.pluginmap` - **Impostazioni** → **Configura RKWard** → **Estensioni**):

SUGGERIMENTO

Dopo aver letto questo capitolo, dare un'occhiata anche al pacchetto [rkwarddev](#). Fornisce alcune funzioni di R per creare al vostro posto la maggior parte dei tag XML di RKWard.

```
<!DOCTYPE rkpluginmap>
```

Il doctype non è realmente interpretato, ma impostarlo comunque a `"rkpluginmap"`.

```
<document base_prefix="" namespace="mie_estensioni" id=" ←  
mia_mappa_estensioni">
```

L'attributo `base_prefix` può essere utilizzato, se tutte le estensioni si trovano in una cartella comune. Fondamentalmente, quindi è possibile omettere quella cartella dai nomi dei file specificati di seguito. È sicuro lasciarlo su `""`.

Come si vedrà di seguito, tutte le estensioni ottengono un identificatore univoco, *id*. *namespace* è un modo per organizzare tali ID e rendere meno probabile la creazione accidentale di un identificatore duplicato. Internamente, fondamentalmente lo spazio dei nomi e quindi un `'::'` viene anteposto a tutti gli identificatori specificati in questa `.pluginmap`. In generale, se si intende [distribuire le estensioni in un pacchetto di R](#), è una buona idea utilizzare il nome del pacchetto come parametro *namespace*. Le estensioni fornite con la distribuzione ufficiale di RKWard hanno `namespace="rkward"`.

L'attributo `id` è facoltativo, ma specificando un id per la propria `.pluginmap` si consente ad altre persone di poter caricare automaticamente la vostra `.pluginmap` dalle loro `.pluginmap` (vedere [la sezione sulle dipendenze](#)).

```
<components>
```

Componenti? Non stiamo parlando di estensioni? Sì, ma in futuro le estensioni non saranno altro che una classe speciale di componenti. Quello che facciamo qui, quindi, è registrare tutti i componenti/estensioni con RKWard. Diamo un'occhiata a una voce di esempio:

```
<component type="standard" id="t_test_two_vars" file="test_t_due_variabili. ←  
xml" label="Test t a due variabili" />
```

Introduzione alla scrittura delle estensioni per RKWard

Innanzitutto l'attributo *type*: lasciarlo su *"standard"* per ora. Altri tipi non sono ancora implementati. L'*id* a cui abbiamo già accennato. A ciascun componente deve essere assegnato un identificativo univoco (nel suo spazio dei nomi). Sceglierne uno facilmente riconoscibile. Evitare spazi e caratteri speciali. Finora non sono stati vietati, ma potrebbero avere significati speciali. Con l'attributo *file*, si specifica dove si trova la [descrizione dell'estensione attuale](#). Questo è relativo alla cartella in cui si trova il file di *.pluginmap* e il *base_prefix* sopra. Infine, fornire un'etichetta al componente. Questa etichetta sarà mostrata nel momento in cui l'estensione viene inserita nel menu (o in futuro, forse anche in altri posti).

Tipicamente un file di *.pluginmap* conterrà diversi componenti, quindi eccone alcuni:

```
<component type="standard" id="test_non_implementato" file="means/ ←
  unimplemented.xml" />
  <component type="standard" id="test_t_fittizio" file="means ←
    /ttests/fictional.xml" label="Questo è un test t ←
    fittizio" />
  <component type="standard" id="descrittiva" file=" ←
    descriptive.xml" label="Statistiche descrittive" />
  <component type="standard" id="corr_matrix" file=" ←
    corr_matrix.xml" label="Matrice di correlazione" />
  <component type="standard" id="anova_semplice" file=" ←
    simple_anova.xml" label="Anova semplice" />
</components>
```

OK, questo è stato il primo passo. RKWard ora sa che esistono quelle estensioni. Ma come invocarle? Devono essere inserite in una gerarchia di menu:

```
<hierarchy>
  <menu id="analisi" label="Analisi">
```

Proprio sotto il tag **<hierarchy>**, si inizia a descrivere, in quale **<menu>** le proprie estensioni dovrebbero andare. Con la riga sopra, in pratica si indica che la propria estensione dovrebbe essere nel menu **Analisi** (non necessariamente direttamente lì, ma in un sottomenu). Il menu **Analisi** è standard in RKWard, quindi non deve essere creato da zero. Tuttavia, se non esistesse ancora, usando l'attributo *label* si attribuisce il suo nome. Infine, *id* identifica ancora una volta questo **<menu>**. Questo è necessario, in modo che diversi file di *.pluginmap* possano posizionare le loro estensioni negli stessi menu. Lo fanno cercando un **<menu>** con il *id* indicato. Se l'ID non esiste ancora, sarà creato un nuovo menu. Altrimenti le voci saranno aggiunte al menu esistente.

```
<menu id="medie" label="Medie">
```

Fondamentalmente la stessa cosa qui: ora definiamo un sottomenu nel menu **Analisi**. Deve essere chiamato **Medie**.

```
<menu id="test_t" label="Test t">
```

E un livello finale nella gerarchia dei menu: un sottomenu del sottomenu **Medie**.

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

Ora, finalmente, questo è il menu in cui vogliamo inserire l'estensione. Il tag **<entry>** segnala che in realtà è il posto effettivo, invece di un altro sottomenu. L'attributo *component* si riferisce all'*id* fornito all'estensione/componente suddetta.

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

Introduzione alla scrittura delle estensioni per RKWard

In caso di smarrimento: questo è un altro sottomenu del menu **Analisi**. Vedere la schermata seguente. Salteremo parte di ciò che non è visibile, contrassegnato da [...].

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

Queste sono le voci finali visibili nelle schermate seguenti.

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

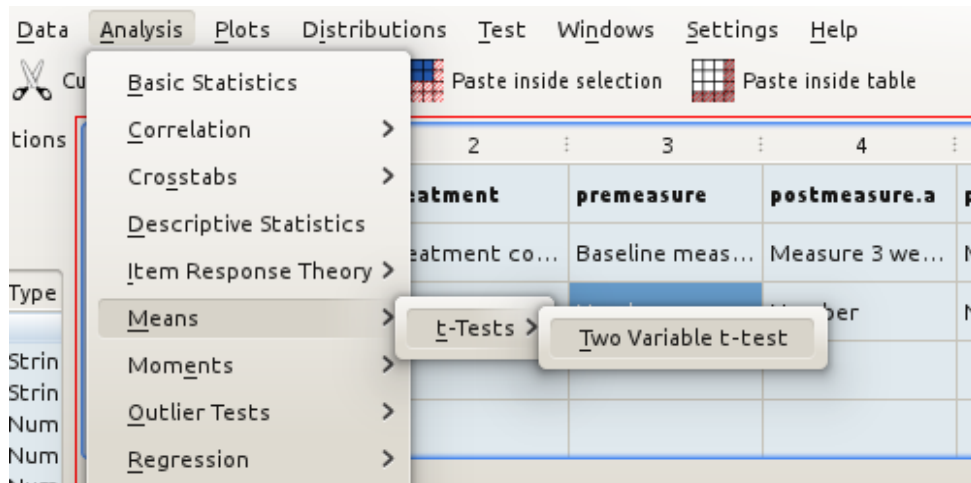
Ovviamente è possibile inserire anche le proprie estensioni in menu diversi da **Analisi**.

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

Anche nei menu standard come **File**. Tutto ciò di cui si ha bisogno è l'*id* corretto.

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

Ecco come farlo. E questa schermata mostra il risultato:



Confusi? Il modo più semplice per iniziare è probabilmente prendere alcuni dei file di `.pluginm` ap esistenti forniti con la distribuzione e modificandoli in base alle proprie esigenze. Inoltre, se si ha bisogno di aiuto, non esitare a scrivere alla lista di distribuzione di sviluppo.

3.1 Controllo dell'ordine delle voci di menu

In modo predefinito, tutti gli elementi (voci / sottomenu) all'interno di un menu saranno ordinati alfabeticamente, automaticamente. In *alcuni* casi si potrebbe volere un maggiore controllo. In tal caso, è possibile raggruppare gli elementi come segue:

- È possibile definire gruppi all'interno di qualsiasi menu come questo. Tutti gli elementi appartenenti allo stesso gruppo saranno raggruppati insieme:

Introduzione alla scrittura delle estensioni per RKWard

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

- Se si desidera che il gruppo sia separato visivamente da altre voci, utilizzare:

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

- Voci, menu e gruppi possono essere aggiunti a un gruppo specificato, utilizzando:

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

- In effetti, è anche possibile definire i gruppi (senza linee di separazione) implicitamente:

```
<entry component="primo" group="a"/>  
  <entry component="terzo"/>  
  <entry component="secondo" group="a"/>
```

- I nomi dei gruppi sono specifici per ciascun menu. Il gruppo «a» nel menu «Dati» non è in conflitto con il gruppo «a» nel menu «Analisi», ad esempio.
- Il caso d'uso più comune è la definizione di gruppi nella parte superiore o nella parte inferiore di un menu. Per questo, ci sono gruppi predefiniti «top» e «bottom» in ciascun menu.
- Le voci all'interno di ciascun gruppo sono ordinate in ordine alfabetico. I gruppi compaiono nell'ordine della dichiarazione (a meno che non vengano aggiunti a un altro gruppo, ovviamente).
- Anche i menu e le voci senza specifica di gruppo formano logicamente un gruppo ("").

Capitolo 4

Definizione della GUI

4.1 Definire una finestra

Nel [capitolo precedente](#) si è visto come registrare un'estensione con RKWard. L'ingrediente più importante è stato specificare il percorso di un file XML con una descrizione dell'aspetto dell'estensione. In questo capitolo imparerete come creare questo file XML.

SUGGERIMENTO

Dopo aver letto questo capitolo, dare un'occhiata anche al pacchetto [rkwdev](#). Fornisce alcune funzioni di R per creare al vostro posto la maggior parte dei tag XML di RKWard.

Ancora una volta vi guideremo attraverso un esempio. L'esempio è una versione (leggermente semplificata) del test `t` a due variabili.

```
<!DOCTYPE rkplugin>
```

Il doctype non è ancora realmente interpretato. Impostarlo comunque a `rkplugin`.

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

Tutti le estensioni generano del codice. Attualmente l'unico modo per farlo è utilizzare JS, come descritto nel [prossimo capitolo](#). Questo definisce dove cercare il codice JS. Il nome file è relativo alla cartella in cui si trova l'estensione XML.

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

Di solito è una buona idea fornire anche una pagina della guida per la propria estensione. Il nome del file di tale pagina, qui fornito, si riferisce, alla cartella in cui si trova l'estensione XML. La scrittura delle pagine della guida è documentata [qui](#). Se non si fornisce un file della guida, omettere questa riga.

```
<dialog label="Test t a due variabili">
```

Come è noto, le estensioni possono avere una finestra di dialogo o un'interfaccia guidata o entrambi. Qui iniziamo a definire un'interfaccia di dialogo. L'attributo `label` specifica la didascalia della finestra di dialogo.

```
<tabbook>
  <tab label="Impostazioni di base">
```

Introduzione alla scrittura delle estensioni per RKWard

Gli elementi della GUI possono essere organizzati utilizzando un tabbook. Qui definiamo una scheda come primo elemento nella finestra di dialogo. Utilizzare `<tabbook>[...]</tabbook>` per definire il tabbook e poi per ciascuna pagina nel tabbook utilizzare `<tab>[...]</tab>`. L'attributo *label* nell'elemento `<tab>` ti consente di specificare una didascalia per quella pagina del tabbook.

```
<row id="riga_configurazione_principale">
```

I tag `<row>` e `<column>` specificano la struttura degli elementi della GUI. Qui si indica che si desidera posizionare alcuni elementi affiancati (da sinistra a destra). L'attributo *id* non è strettamente necessario, ma sarà utilizzato in seguito, quando si aggiunge un'interfaccia guidata alla nostra estensione. Il primo elemento da posizionare nella riga è:

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

Utilizzando questo semplice tag si crea un elenco da cui l'utente può selezionare le variabili. È necessario specificare un *id* per questo elemento, in modo che RKWard sappia come trovarlo.

ATTENZIONE

NON si dovrebbe utilizzare un punto (.) Nella stringa *id*.

```
<column>
```

Successivamente, annidiamo una `<column>` all'interno della riga. Cioè i seguenti elementi saranno posizionati uno sopra l'altro (dall'alto verso il basso) e tutti saranno a destra 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=" <
                                confronta con"/>
```

Questi elementi sono la controparte del `<varselector>`. Rappresentano 'slot' in cui l'utente può inserire variabili. Noterete che *source* è impostato sullo stesso valore di *id* del `<varselector>`. Ciò significa che i `<varslot>` prenderanno ciascuno le proprie variabili dal varselector. Anche i `<varslot>` devono avere un *id*. Possono avere un'etichetta e possono essere impostati su *required*. Ciò significa che il pulsante **Invia** non sarà abilitato finché il `<varslot>` non conterrà un valore valido. Infine l'attributo *type* non è ancora interpretato, ma sarà utilizzato per fare in modo che solo i tipi corretti di variabili siano consentiti nel `<varslot>`.

Nel caso ci si stia chiedendo dell'attributo *i18n_context*: questo serve a fornire un contesto per aiutare la traduzione corretta della parola «contro», utilizzata come etichetta del `<varslot>`, ma non influisce direttamente sulla funzionalità dell'estensione. Altre informazioni al riguardo in [un capitolo separato](#).

```
<radio id="hypothesis" label="ipotesi del test in uso">
                                <option value="due. <
                                lati" label="Due <
                                lati"/>
                                <option value=" <
                                maggiore" label <
                                ="Il primo è più <
                                grande"/>
                                <option value=" <
                                minore" label=" <
                                Il secondo è più <
                                grande"/>
```

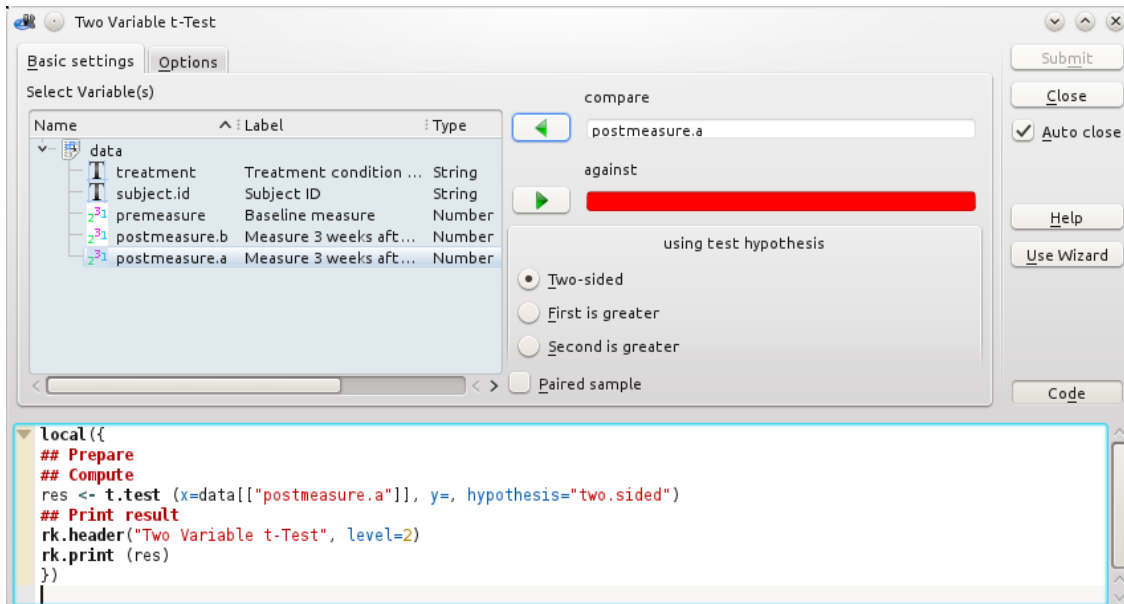
Introduzione alla scrittura delle estensioni per RKWard

```
</radio>
```

Qui definisci un gruppo di pulsanti esclusivi `<radio>`. Il gruppo ha un'etichetta e un *id*. Ogni `<opzione>` (pulsante) ha un'etichetta e gli viene assegnato un *valore*. Questo è il valore che l'elemento `<radio>` restituirà quando l'opzione è selezionata.

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

Ogni tag deve essere chiuso. Abbiamo inserito tutti gli elementi che volevamo (i due `<varslots>` e il `<radio>`) nel `<column>`. Inseriamo tutti gli elementi che vogliamo (il `<varselector>` e il `<column>` con quegli elementi) nel `<row>`. E abbiamo inserito tutti gli elementi che volevamo nella prima pagina del `<tabbook>`. Non abbiamo ancora finito di definire il `<tabbook>` (altre pagine a venire), e ovviamente c'è altro in arrivo anche nel `<dialog>`. Ma questa schermata è fondamentalmente ciò che abbiamo fatto finora:



Notare che non abbiamo specificato i pulsanti **Invia**, **Chiudi** ecc. o la vista del codice. Questi elementi sono generati automaticamente. Ma ovviamente dobbiamo ancora definire la seconda pagina del `<tabbook>`:

```
<tab label="Opzioni">
                                <checkbox id="varequal" label="assumi <-
                                varianze uguali" value="", var.equal=TRUE <-
                                ">
```

Per impostazione predefinita, gli elementi saranno posizionati dall'alto verso il basso come in una `<colonna>`. Poiché questo è ciò che vogliamo qui, non dobbiamo dichiarare esplicitamente una disposizione `<riga>` o `<colonna>`. Il primo elemento che definiamo è una casella di controllo. Proprio come i `<radio>``<opzione>`, la casella di controllo ha un'etichetta e un *valore*. Il *valore* è ciò che viene restituito, se la casella di controllo è selezionata. Ovviamente la casella di controllo necessita anche di un *id*.

```
<frame label="Intervallo di confidenza" id="frame_conf_int">
```

Ecco ancora un altro elemento di struttura: per segnalare che i due elementi sottostanti sono uniti, tracciamo un `<frame>` (scatola). Quella cornice può avere una *label* (didascalia). Poiché la

cornice è solo un elemento passivo della struttura, non ha bisogno di un *id*, ne definiamo ancora uno qui, come faremo riferimento in seguito, quando definiremo un'interfaccia aggiuntiva della procedura

```
<checkbox id="confint" label="stampa l'intervallo di confidenza" value="1" ←
  checked="true"/>
                                <spinbox type="real" id="conflevel" ←
                                  label="livello di confidenza" ←
                                  min="0" max="1" initial="0.95"/>
                                </frame>
```

All'interno del **<frame>** inseriamo un altro **<checkbox>** (utilizzando *checked="true"*, segnaliamo che la casella di selezione dovrebbe essere selezionata per impostazione predefinita) e un **<spinbox>**. La casella numerica consente all'utente di selezionare un valore compreso tra *min* e *max* con il valore predefinito/iniziale *0.95*. L'impostazione di *type* su *real* segnala che i numeri reali sono accettati al contrario di *type="integer"* che accetterebbe solo numeri interi.

NOTA

È anche possibile, e spesso preferibile, rendere il **<frame>** stesso controllabile, invece di aggiungere una **<checkbox>** all'interno. Vedere il riferimento per i dettagli. Questo non è fatto qui, a scopo illustrativo.

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

Questo è tutto per la seconda pagina del **<tabbook>**, tutte le pagine nel **<tabbook>** e tutti gli elementi nella **<dialog>**. Abbiamo finito di definire l'aspetto della finestra.

```
</document>
```

Alla fine chiudiamo il tag **<document>**, e basta. La GUI è definita. È possibile salvare il file ora. Ma come viene generata la sintassi di R dalle impostazioni della GUI? Ci occuperemo di ciò nel [prossimo capitolo](#). In primo luogo, tuttavia, esamineremo l'aggiunta di un'interfaccia guidata e alcune considerazioni generali.

4.2 Aggiunge un'interfaccia di procedura guidata

In realtà non è necessario definire un'interfaccia aggiuntiva **<wizard>**, ma ecco come dovrebbe essere fatto. Per aggiungere un'interfaccia della procedura guidata, aggiungere un tag **<wizard>** allo stesso livello del tag **<dialog>**:

```
<wizard label="Test t a due variabili">
  <page id="prima_pagina">
    <text>Come primo passo, seleziona le due ←
      variabili che desideri confrontare.
      E specifica, quale ipotizzi sia la ←
      più grande. Seleziona due lati, ←
      se la tua ipotesi non indica quale ←
      variabile sia la più grande.</ ←
      text>
    <copy id="riga_impostazioni_principali"/>
  </page>
```

Alcune di queste sono piuttosto autoesplicative: aggiungiamo un tag `<wizard>` con un'etichetta per la procedura guidata. Poiché una procedura guidata può contenere diverse pagine visualizzate una dopo l'altra, definiamo quindi il primo `<page>` e inseriamo una nota esplicativa `<text>`. Quindi utilizziamo un tag `<copy>`. Ciò che fa è che in realtà ci evita di dover definire ancora una volta ciò che abbiamo già scritto per il `<dialog>`: il tag copy cerca un altro tag con lo stesso *id* in precedenza nel XML. Questo sembra essere definito nella sezione `<dialog>`, ed è un `<row>` in cui sono presenti il `<varselector>`, `<varslots>` e il controllo `<radio>` 'ipotesi'. Tutto questo viene copiato 1:1 e inserito direttamente nell'elemento `<copy>`.

Ora alla seconda pagina:

```
<page id="seconda_pagina">
    <text>Di seguito sono disponibili alcune ←
        opzioni avanzate. È generalmente sicura ←
        non assumere che
            le variabili abbiano varianze ←
            uguali. Una correzione ←
            appropriata sarà quindi ←
            applicata.
        Tuttavia, la scelta di "assumi ←
            varianze uguali" potrebbe ←
            incrementare la potenza del test ←
        .</text>
    <copy id="varequal"/>
    <text>A volte è utile ottenere una stima ←
        dell'intervallo di confidenza della
        differenza tra le medie. Sotto, ←
        puoi specificare se una dovrebbe ←
        essere mostrata, e
        quale livello di confidenza ←
        dovrebbe essere applicata (95% ←
        corrisponde a un livello di ←
        significatività
        del 5%).</text>
    <copy id="frame_conf_int"/>
</page>
</wizard>
```

Quasi la stessa cosa qui. Aggiungiamo alcuni testi e tra queste ulteriori sezioni `<copy>` dall'interfaccia.

Ovviamente, è possibile rendere l'interfaccia della procedura guidata molto diversa dalla semplice finestra e non utilizzare affatto il tag `<copy>`. Assicurarsi, tuttavia, di assegnare agli elementi corrispondenti lo stesso *id* in entrambe le interfacce. Questo non viene utilizzato solo per trasferire le impostazioni dall'interfaccia di dialogo all'interfaccia della procedura guidata e viceversa, quando l'utente cambia interfaccia (che non si verifica ancora nella versione attuale di RKWard), ma semplifica anche la scrittura del modello di codice (vedere sotto) .

4.3 Alcune considerazioni sulla struttura della GUI

Questa sezione contiene alcune considerazioni generali su quali elementi della GUI utilizzare. Se questo è il primo tentativo di creare un'estensione, non esitare a saltare questa sezione, poiché non è rilevante per ottenere una GUI di base funzionante. Tornare qui, più tardi, per vedere se è possibile perfezionare la GUI della propria estensione in un modo o nell'altro.

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

I tre elementi <radio>, <checkbox>, <dropdown>, hanno tutti una funzione simile: selezionare una tra diverse opzioni. Ovviamente, una casella di selezione consente di scegliere solo tra due opzioni: selezionata o non selezionata, quindi non è possibile utilizzarla se ci sono più di due opzioni tra cui scegliere. Ma quando utilizzare quale di questi elementi? Ecco alcune regole pratiche:

Se vi ritrovate a creare un <radio> o <dropdown> con solo due opzioni, chiedetevi se la domanda è essenzialmente un tipo di domanda sì/no. Ad esempio, una scelta tra 'modifica i risultati' e 'non modificare i risultati', o tra 'rimuovi i valori mancanti' e 'mantieni i valori mancanti'. In questo caso un <checkbox> è la scelta migliore: utilizza poco spazio, avrà il minor numero di parole di etichette ed è più facile da leggere per l'utente. Ci sono pochissime situazioni in cui dovrete scegliere un <radio> invece di un <checkbox>, quando ci sono solo due opzioni. Un esempio potrebbe essere: 'Metodo di calcolo: 'pearson'/'spearman''. In questo caso potrebbero essere pensabili più metodi, che in realtà non formano una coppia di opposti.

Scegliere tra un <radio> e un <dropdown> è principalmente una questione di spazio. Il <dropdown> ha il vantaggio di occupare poco spazio, anche se ci sono molte opzioni tra cui scegliere. D'altra parte, un <radio> ha il vantaggio di rendere visibili all'utente tutte le scelte possibili contemporaneamente, senza fare clic sulla freccia del menu a tendina. Generalmente, se ci sono sei o più opzioni tra cui scegliere, è preferibile un <dropdown>. Se ci sono cinque o meno opzioni, un <radio> è la scelta migliore.

Capitolo 5

Generare il codice di R dalle impostazioni della GUI

5.1 Utilizzare JavaScript nelle estensioni di RKWard

Ora abbiamo una GUI definita, ma dobbiamo ancora generare del codice R da quello. Per questo, abbiamo bisogno di un altro file di testo, `code.js`, situato nella stessa cartella di `description.xml`. Potreste avere familiarità o meno con JavaScript (o, per essere tecnicamente precisi: script ECMA). La documentazione su JS può essere trovata in abbondanza, sia in forma stampata che su Internet (ad es.: https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide). Ma per la maggior parte degli scopi non avrete bisogno di sapere molto di JS, poiché utilizzeremo solo alcune funzionalità di base.

SUGGERIMENTO

Dopo aver letto questo capitolo, date un'occhiata anche al pacchetto `rkwarddev`. Fornisce alcune funzioni R per creare codice JavaScript comunemente utilizzato in RKWard. Può anche rilevare automaticamente le variabili utilizzate in un file XML di un'estensione e creare codice JavaScript di base da cui partire.

NOTA

Si presume che i file `.js` dell'estensione siano codificati in UTF-8. Assicurarsi di controllare la codifica dell'editor, se si utilizzano caratteri non ascii.

Per il test `t` a due variabili, il file `code.js` ha il seguente aspetto (con commenti nel mezzo):

5.1.1 preprocess()

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

Il file JS è organizzato in tre funzioni separate: `preprocess()`, `calculate()` e `printout()`. Questo perché non tutto il codice è necessario in tutte le fasi. Attualmente la funzione di pre-elaborazione non è realmente utilizzata in molti punti (in genere sarà omessa del tutto).

5.1.2 calculate()

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

Questa funzione genera l'attuale sintassi di R da eseguire dalle impostazioni della GUI. Vediamo questo in dettaglio: il codice da utilizzare è generato utilizzando l'istruzione `echo()`. Osservando l'istruzione `echo()` passo dopo passo, la prima parte è

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

come testo semplice. Successivamente dobbiamo inserire il valore selezionato dall'utente come prima variabile. Lo recuperiamo utilizzando `getString ("x")` e lo aggiungiamo alla stringa da 'echo'. Questo stampa il valore dell'elemento della GUI con `id="x"`: il nostro primo **<checkbox>**. Successivamente, aggiungiamo un `,` e facciamo lo stesso per recuperare il valore dell'elemento `"y"` - il secondo **<checkbox>**. Per l'ipotesi (il gruppo **<radio>**) e le varianze uguali **<checkbox>**, la procedura è molto simile.

NOTA

Invece di concatenare le stringhe utilizzando l'operatore `+`, puoi anche usare un «template literal», come questo (nota che la stringa è racchiusa tra apici inversi `'`):

```
echo('res <- t.test (x=${ getString("x") }, y=${ getString("y") }, ←
  hypothesis="${ getString("hypothesis") }"); // ecc.
```

Tenere presente che invece di concatenare gli snippet di uscita con `+`, è possibile anche utilizzare diverse istruzioni `echo()`. Tutto è stampato su una sola riga. Per produrre un'interruzione di riga nel codice generato, inserire un `"\n"` nella stringa stampata. In teoria, è possibile persino produrre molte righe con una singola istruzione `echo`, ma consigliamo limitarsi a una riga (o meno) di codice generato per `echo()`.

NOTA

Oltre a `getString()`, ci sono anche funzioni `getBoolean()`, che proveranno a restituire il valore come logico (adatto per l'uso in un'istruzione `if()`) e `getList()`, che proveranno a restituire dati simili a quelli di una lista in un `Array()` JS. Ne mostreremo esempi in seguito.

Quando si cercano le estensioni esistenti, si troveranno molte estensioni utilizzando `getValue()`, invece di `getString()`, e in effetti le due sono *quasi* identiche. Tuttavia, l'utilizzo di `getString()`, `getBoolean()` e `getList()` è la pratica consigliata dalla versione 0.6.1.

Diventa un po' più complicato per il livello di confidenza. Per ragioni estetiche, non vogliamo specificare esplicitamente il livello di confidenza da utilizzare, se corrisponde al valore predefinito. Quindi, invece di stampare il valore incondizionatamente, recuperiamo prima una variabile. Quindi controlliamo se quella variabile differisce da `"0.95"` e in tal caso stampiamo un argomento aggiuntivo. Infine, aggiungiamo una parentesi di chiusura e un'interruzione di riga: `")\n"`. Questo è tutto per la funzione di calcolo.

5.1.3 printout()

```
function printout () {
  echo ('rk.header (' + i18n ("Test t a due variabili") + ')\n');
  echo ('rk.print (res)\n');
}
```

E questo era tutto ciò che c'era nella funzione di stampa nella maggior parte dei casi. `rk.header()` stampa un'intestazione standard per i risultati. Tenere presente che nei file `.js` bisogna contrassegnare tutte le stringhe traducibili, usando `i18n()`, o alcuni comandi alternativi. Maggiori informazioni a riguardo sono disponibili nel [capitolo sull'internazionalizzazione](#). È possibile anche aggiungere altre informazioni a questo, se lo si desidera, ad es.:

```
function printout () {
  new Header (i18n ("Test t a due variabili"))
    .addFromUI ("varequal")
    .add (i18n ("Livello di confidenza"), getString (" ←
      conflevel")) // Nota: scritto in questo modo a scopo ←
      illustrativo. Più automatico:
  //      .addFromUI ("conflevel")
    .print ();
  echo ('rk.print (res)\n');
}
```

`rk.print()` utilizza il pacchetto `R2HTML` per fornire output HTML formattato. Un'altra funzione utile è `rk.results()`, che può anche generare diversi tipi di tabelle dei risultati. In caso di dubbio, tuttavia, usare semplicemente `rk.print()` e il gioco è fatto. La classe `JS Header` è un helper di livello JS per generare una chiamata a `rk.header()` (basta dare un'occhiata al codice R generato). In alcuni casi potreste voler chiamare `echo ('rk.header (...)')` direttamente per stampare un'intestazione per il vostro output.

Nota che internamente, l'output è un semplice documento HTML in questo momento. Pertanto potreste essere tentati di aggiungere HTML personalizzato, utilizzando `rk.cat.output()`. Anche se funzionerà, non fatelo. Il formato di output potrebbe cambiare (ad es. in ODF) in futuro, quindi è meglio non introdurre HTML codice specifico. Piuttosto mantenete le cose semplici con `rk.header()`, `rk.print()`, `rk.results()` e, se necessario, `rk.print.literal()`. Se questi non sembrano soddisfare le vostre esigenze di formattazione, contattateci sulla mailing list per ricevere assistenza.

Congratulazioni! Avete creato la vostra prima estensione. Continuare a leggere nei prossimi capitoli per concetti più avanzati.

5.2 Convenzioni, criteri e sfondo

Esistono molti modi per scrivere codice R per un determinato compito e ci sono ancora più modi per generare questo codice R da JS. Come farlo esattamente, è lasciato a voi. Ci sono ancora alcune considerazioni che si dovrebbero seguire e informazioni di base che si dovrebbero comprendere.

5.2.1 Comprendere l'ambiente `local()`

Molto spesso sarà necessario creare uno o più oggetti temporanei di R nel codice generato dalla propria estensione. Normalmente, non si desidera che vengano posizionati nell'area di lavoro dell'utente, potenzialmente anche sovrascrivendo le variabili utente. Quindi, tutto il codice generato dall'estensione viene eseguito in un ambiente `local()` (consultare la pagina della guida di R sulla funzione `local()`). Ciò significa che tutte le variabili create sono temporanee e non saranno salvate in modo permanente.

Se l'utente chiede esplicitamente il salvataggio di una variabile, sarà necessario assegnarla a quell'oggetto usando `.GlobalEnv$objectname <- valore`. In generale, non utilizzare l'operatore `<<-`. Non sarà assegnato necessariamente in `.GlobalEnv`.

Un errore importante è usare `eval()`. Notare che `eval` utilizzerà per impostazione predefinita l'ambiente attuale per la valutazione, cioè quello locale. Funzionerà bene la maggior parte delle volte, ma non sempre. Quindi, se si ha bisogno di usare `eval()`, probabilmente si vorrà specificare il parametro `envir`: `eval(..., envir=globalenv())`.

5.2.2 Formattazione del codice

La cosa più importante perché il proprio codice R generato funzioni, ma dovrebbe anche essere facile da leggere. Pertanto, fare attenzione anche alla formattazione. Alcune considerazioni:

Le istruzioni di primo livello di R dovrebbero essere allineate a sinistra.

Le istruzioni in un blocco inferiore devono essere rientrate con una tabulazione (vedere l'esempio seguente).

Se si eseguono calcoli molto complessi, aggiungere un commento ogni tanto, specialmente per contrassegnare sezioni logiche. Si noti che esiste una funzione dedicata `comment()` per l'inserimento di commenti traducibili nel codice generato.

Ad esempio, il codice generato potrebbe apparire così. Lo stesso codice senza rientri o commenti sarebbe piuttosto difficile da leggere, nonostante la sua modesta complessità:

```
# determina prima l'oscillazione e la rotazione
my.wobble <- wobble (x, y)
my.rotation <- wobble.rotation (my.wobble, z)

# il metodo preciso deve essere scelto in base alla rotazione
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 Gestire le operazioni complesse

Molte estensioni possono fare più di una cosa. Ad esempio, l'estensione 'Statistiche descrittive' può calcolare media, intervallo, somma, prodotto, mediana, lunghezza, ecc. Tuttavia, in genere l'utente sceglierà di eseguire solo alcuni di questi calcoli. In questo caso, provare a mantenere il codice generato il più semplice possibile. Dovrebbe contenere solo parti rilevanti per le opzioni effettivamente selezionate. Per raggiungere questo obiettivo, ecco un esempio di un modello di progettazione comune come potrebbe essere utilizzato (in JS; qui, "domean", "domedian" e "dosd" sarebbero elementi <checkbox>):

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

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

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

5.3 Suggerimenti e trucchi

Ecco alcuni trucchi assortiti che possono rendere meno noiosa la scrittura delle estensioni:

Se si ha bisogno del valore di un'impostazione della GUI in diversi punti del codice della propria estensione, considerare di assegnarlo a una variabile in JS e usarlo invece di recuperarlo ancora e ancora con `getString()/getBoolean()/getList()`. Questo è più veloce, più leggibile e comporta una minore digitazione allo stesso tempo:

```
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 semplice funzione di supporto `makeOption()` può rendere più semplice l'omissione di parametri che sono al loro valore predefinito, in molti casi:

```
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");  
  //...  
}
```

Capitolo 6

Scrivere una pagina della guida

Quando la propria estensione praticamente funziona, è arrivato il momento di fornire una pagina di guida. Mentre in genere non si vorranno spiegare tutti i concetti sottostanti in profondità, si potrebbe voler aggiungere qualche spiegazione in più per alcune delle opzioni e fornire collegamenti alle estensioni correlate e alle funzioni di R.

SUGGERIMENTO

Dopo aver letto questo capitolo, dare un'occhiata anche al pacchetto [rkwarddev](#). Fornisce alcune funzioni di R per creare la maggior parte dei tag XML di RKWard. È anche in grado di creare scheletri di file di aiuto di base da file XML di estensioni esistenti cui iniziare.

Ci si dovrebbe ricordare di inserirlo nel codice XML della propria estensione (se non è stato inserito, farlo subito):

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

Dove, ovviamente, si desidera sostituire `nomefile` con un nome più appropriato. Ora è il momento di creare questo file `.rkh`. Ecco un esempio auto-descrittivo:

```
<!DOCTYPE rkhelp>
<document>
  <summary>
In questa sezione, fornirete alcune brevi informazioni di base su ciò che ←
fa l'estensione
Questa sezione sarà sempre visualizzata nella parte superiore della pagina ←
di aiuto.
  </summary>

  <usage>
La sezione sull'utilizzo può contenere alcune informazioni più pratiche. ←
Tuttavia, non spiega tutte
le impostazioni in dettaglio (questo è fatto nella sezione di "settings").

Per iniziare un nuovo paragrafo, inserire una riga vuota, come mostrato ←
sopra.
Questa riga, al contrario, sarà nello stesso paragrafo.
```

Introduzione alla scrittura delle estensioni per RKWard

In tutte le sezioni puoi inserire del semplice codice HTML, come `<b>bold</b>` o `<i>italic</i>` text. Comunque, mantenere la formattazione al minimo necessario.

La sezione sull'utilizzo è sempre la seconda sezione mostrata in una pagina della guida.

```
</usage>
```

```
<section id="sectionid" title="Sezione generica" short_title="Generica">
```

Se necessario, è possibile aggiungere ulteriori sezioni tra le sezioni di utilizzo e impostazioni.

Tuttavia, di solito, non sarà necessario durante la documentazione delle estensioni. L'attributo «id» fornisce un punto di ancoraggio per saltare a questa sezione dal menu di navigazione. L'attributo «short_title» fornisce un breve titolo da utilizzare nella barra di navigazione. Questo è facoltativo, in modo predefinito sarà utilizzato il titolo principale «title» sia come intestazione alla sezione, sia come nome del collegamento nella barra di navigazione.

In ogni sezione potreste voler inserire dei collegamenti a ulteriori informazioni. Potete farlo aggiungendo

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

Dove URL può essere un collegamento esterno del tipo `https://rkward.kde.org`.

Sono supportati diversi URL speciali nelle pagine della guida:

```
<link href="rkward://pagina/percorso/id_pagina"/>
```

Questo collega a una pagina della guida di primo livello di rkward (non per un'estensione).

```
<link href="rkward://componente/[spazio_dei_nomi/]id_componente"/>
```

Questo collega alla pagina della guida di un'altra estensione. La parte `[ spazio_dei_nomi/]` potrebbe essere omessa (in questo caso, rkward è assunto come spazio dei nomi standard, ad es.: `<link href="rkward://componente/importa_spss"/>` o `<link href="rkward://componente/rkward/importa_spss"/>` sono equivalenti). L'id_componente è lo stesso che hai specificato nella `.pluginmap`.

```
<link href="rkward://rhelprfunction"/>
```

Collega alla pagina della guida di R per «rfunction».

Notare che i nomi dei collegamenti saranno generati automaticamente per questi tipi di collegamenti.

```
</section>
```

```
<settings>
```

```
<caption id="id_scheda_o_cornice"/>
```

```
<setting id="id_elemento">
```

Descrizione dell'elemento della GUI identificato dall'ID specificato

Introduzione alla scrittura delle estensioni per RKWard

```
        </setting>
        <setting id="id_elementob" title="descrizione">
Normalmente, il titolo dell'elemento della GUI sarà estratto dalla
definizione XML dell'estensione, automaticamente. Tuttavia,
per alcuni elementi della GUI, questa descrizione potrebbe non essere ←
        sufficiente per identificarli, in modo affidabile.
In questo caso, potete aggiungere un titolo esplicito utilizzando l' ←
        attributo «title».
        </setting>
        <setting id="id_elementoc">
Descrizione dell'elemento della GUI identificato da «id_elementoc»
        </setting>
        [...]
    </settings>

    <related>
La sezione relativa tipicamente contiene solo alcuni elementi, come:

<ul>
    <li><link href="rkward://rhelph/mean"/></li>
    <li><link href="rkward://rhelph/median"/></li>
    <li><link href="rkward://componente/componente_relativo"/></li>
</ul>

    </related>

    <technical>
La sezione tecnica (facoltativa, sempre ultima) può contenere alcuni ←
        dettagli tecnici dell'implementazione
dell'estensione, che sono d'interesse solo per gli sviluppatori di RKWard. ←
        Questo è particolarmente rilevante
per le estensioni che sono progettate per essere integrate in altre ←
        estensioni, e può dettagliare quali
opzioni sono disponibili per personalizzare l'estensione integrata, e quali ←
        sezioni di codice contengono quale
codice di R.
        </technical>
</document>
```

Capitolo 7

Interazioni logiche tra gli elementi della GUI

7.1 Logica della GUI

Tutti i concetti di base per la creazione di un'estensione per RKWard sono stati descritti nei capitoli precedenti. Tali concetti di base dovrebbero essere sufficienti per molti, se non nella maggior parte dei casi. Tuttavia, a volte si potrebbe desiderare un maggiore controllo sul comportamento della GUI della propria estensione.

Ad esempio, supponiamo di voler estendere l'esempio del test t utilizzato in questa documentazione per consentire entrambi: il confronto di una variabile con un'altra variabile (come mostrato) e il confronto di una variabile con un valore costante. Ora, un modo per farlo sarebbe quello di aggiungere un controllo radio che commuta tra le due modalità e aggiungere una casella di selezione per inserire il valore costante da confrontare. Considera questo esempio semplificato:

```
<!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" ←
            il8n_context="Noun; a variable"/>
        <spinbox id="constant" initial="0" label=" ←
          constant" il8n_context="Noun; a constant ←
            "/>
      </column>
    </row>
```

```
</dialog>
</document>
```

Fin qui tutto bene, ma ci sono diversi problemi con questa GUI. Innanzitutto vengono sempre visualizzati sia il varslot che la spinbox, mentre solo uno dei due viene realmente utilizzato. Peggio ancora, varslot richiede sempre una selezione valida, anche se confrontata con una costante. Ovviamente, se creiamo una GUI in questo modo, vogliamo maggiore flessibilità. Immettere: la sezione **<logic>** (inserita allo stesso livello di **<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 prima riga all'interno della sezione logica è un tag **<convert>**. Fondamentalmente, questo fornisce una nuova proprietà booleana (on o off, true o false), che può essere utilizzata in seguito. Questa proprietà ("varmode") è vera ogni volta che viene selezionato il pulsante di opzione superiore e falsa ogni volta che viene selezionato il pulsante di opzione inferiore. Come si fa?

Innanzitutto, sotto *sources*, sono elencate le proprietà delle fonti su cui lavorare (in questo caso solo una ciascuna; potresti elencarne diverse come *sources="mode.string; somethingelse"*, quindi "varmode" sarebbe vero solo se sia "mode.string" che "qualcosa" sono uguali alla stringa "variabile"). Notare che in questo caso non scriviamo solo "mode" (come faremmo in `getString("mode")`), ma "mode.string". Questo è in realtà il modo interno in cui funziona un radiocomando: ha una proprietà 'string', che ne mantiene il valore stringa. `getString("mode")` è solo una scorciatoia ed equivalente a `getString("mode.string")`. Vedere il riferimento per tutte le proprietà dei diversi elementi della GUI.

In secondo luogo, impostiamo la modalità di conversione su *mode="uguale a"*. Ciò significa che vogliamo verificare se la(e) fonte(i) è(sono) uguale a un certo valore. Infine standard è il valore con cui confrontare, quindi con *standard="variabile"*, controlliamo se la proprietà "mode.string" è uguale alla stringa "variabile" (il valore dell'opzione radio superiore). Se è uguale, la proprietà varmode è vera, altrimenti è falsa.

Veniamo ora alle cose reali: **<connect>** la proprietà "varmode" a *y.visible*, che controlla se il varslot "y" viene mostrato o meno. Tenere presente che qualsiasi elemento reso invisibile è implicitamente non obbligatorio. Pertanto, se viene selezionata l'opzione radio superiore, il varslot "y" è richiesto e visibile. Altrimenti non è richiesto e nascosto.

Per la spinbox, vogliamo l'esatto contrario. Fortunatamente, non abbiamo bisogno di un altro **<convert>** per questo: le proprietà booleane possono essere negate molto facilmente aggiungendo il modificatore "not", quindi **<connect>** "varmode.not" alla proprietà di visibilità della casella numerica. In effetti, viene mostrato e richiesto il varslot, o la casella numerica - a seconda dell'opzione selezionata nel radiocomando. La GUI si sta modificando in base all'opzione radio. Provare l'esempio, se lo si desidera.

Per un elenco completo delle proprietà, fare riferimento alla [guida](#). Un'altra proprietà, tuttavia, è speciale in quanto tutte gli elementi della GUI la hanno: 'enabled'. Questo è leggermente meno drastico di 'visibile'. Non mostra/nasconde l'elemento della GUI, ma lo abilita/disabilita solamente. Gli elementi disabilitati sono generalmente visualizzati in grigio e non reagiscono al comando dell'utente.

NOTA

Oltre a **<convert>** e **<connect>**, sono disponibili diversi altri elementi utilizzabili nella sezione **<logic>**. Ad esempio, i costrutti condizionali possono essere implementati anche utilizzando l'elemento **<switch>**. Per maggiori dettagli, consultare il [riferimento sugli elementi logici](#).

7.2 Logica della GUI programmata

Mentre connettere le proprietà come descritto sopra è spesso sufficiente, a volte è più flessibile o più conveniente utilizzare JS per creare script della logica della GUI. In questo modo, l'esempio sopra potrebbe essere riscritto come:

```
[...]
    <code file="code.js"/>
    ,
    <logic>
        <script><![CDATA[
            // [...] qualsiasi codice al livello superiore ←
            viene invocato una sola volta
            gui.addChangeCommand("mode.string", function() {
                // mentre questa funzione anonima sarà ←
                invocata ogni volta che «mode.string» ←
                cambia
                var varmode = (gui.getString("mode.string") ←
                    == "variabile");
                gui.setValue("y.enabled", varmode);
                gui.setValue("constant.enabled", !varmode);

            });
        ]]></script>
    </logic>

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

Registra una funzione anonima da invocare ogni volta che cambia il valore della casella radio *id* = "mode". All'interno di questa funzione, definiamo una variabile helper *varmode* che è vera quando la modalità è *variabile*, falsa perché è *costante*. Quindi utilizziamo `gui.setValue()` per impostare le proprietà 'enabled' di *y* e *constant*, proprio come facevamo prima utilizzando le istruzioni **<connect>**.

NOTA

Se la stessa funzione deve essere invocata per modifiche in più elementi, potete anche passare un array dei rispettivi *id* a **gui.addChangeCommmand()**. Inoltre, per comodità, **gui.addChangeCommmand()** restituisce il suo secondo parametro, che è utile se volete fare riferimento a questa funzione altrove (ad esempio per invocarla una volta durante l'inizializzazione). Ad esempio:

```
let update = gui.addChangeCommand(["mode.string", "y.available"], ←
    function() {
        // fai qualcosa a ogni cambiamento di mode.string o y. ←
        available

    });
update(); // fai lo stesso una volta durante l'inizializzazione
```

Introduzione alla scrittura delle estensioni per RKWard

L'approccio basato su script alla logica della GUI diventa particolarmente utile quando si desidera modificare l'opzione disponibile in base al tipo di oggetto che l'utente ha selezionato. Vedere [la guida](#) per le funzioni disponibili.

Notare che l'approccio basato su script alla GUI la logica può essere mischiata con le istruzioni **<connect>** e **<convert>**, se lo si desidera. Tenere inoltre presente che il tag **<script>** consente di specificare un nome di file di script in aggiunta o in alternativa all'incorporamento del codice dello script. In genere, tuttavia, è più conveniente incorporare il codice dello script come mostrato sopra.

Capitolo 8

Integrazione di estensioni in estensioni

8.1 Casi di utilizzo per l'integrazione

Quando si scrivono estensioni, spesso si scoprirà che si sta creando diverse estensioni che differiscono solo per alcuni aspetti, ma hanno molto di più in comune. Ad esempio, per il tracciamento dei grafici, ci sono numerose opzioni generiche di R che possono essere utilizzate con quasi tutti i tipi di grafici. Vorreste creare una GUI e un modello JS ancora e ancora?

Ovviamente sarebbe una vera seccatura. Fortunatamente, non è necessario farlo. Piuttosto, si crea una volta la funzionalità comune e successivamente la si può integrare in diverse estensioni. In effetti è possibile integrare qualsiasi estensione in un'altra estensione, anche se l'autore originale dell'estensione integrata non lo ha mai immaginato, qualcuno potrebbe voler integrare l'estensione in un'altra.

8.2 Integrazione in una finestra

OK, abbiamo detto abbastanza. Come funziona? Semplice: utilizzare il tag `<embed>`. Ecco un esempio ridotto:

```
<dialog>
  <tabbook>
    <tab [...]>
      [...]
    </tab>
    <tab label="Plot Options" i18n_context="Opzioni relative al ↔
      grafico">
      <embed id="plotoptions" component="rkward:: ↔
        plot_options"/>
    </tab>
    <tab [...]>
      [...]
    </tab>
  </tabbook>
</dialog>
```

Ciò che succede qui è che l'intera GUI o l'estensione delle opzioni di tracciamento dei grafici (tranne ovviamente per gli elementi standard come il pulsante **Invia**, ecc.) è integrato direttamente nella propria estensione (da provare!).

Come si può vedere, la sintassi del tag `<embed>` è abbastanza semplice. Richiede un *id* come la maggior parte degli elementi. Il componente parametro specifica quale estensione incorporare, come definito nella `.pluginmap` (`"rkward::plot_options"` è il risultato della concatenazione dello spazio dei nomi `'rkward'`, un separatore `::` e il nome del componente `'plot_options'`).

8.3 Generazione di codice durante l'integrazione

Fin qui tutto bene, ma per quanto riguarda il codice generato? Come vengono uniti il codice per l'integrazione e l'estensione integrata? Nel codice JS dell'estensione di integrazione, basta scrivere qualcosa del genere:

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

Quindi, in sostanza, stiamo recuperando il codice generato dall'estensione incorporata proprio come stiamo recuperando qualsiasi altra impostazione della GUI. Qui la stringa `"plotoptions.code.printout"` può essere rielaborata in: 'La sezione di stampa del codice generato dell'elemento con il *id* plotoptions' (plotoptions è l'ID che abbiamo fornito per il tag `<embed>` sopra). E sì, se vuoi un controllo avanzato, puoi anche recuperare i valori delle singole GUI elementi all'interno dell'estensione incorporata (ma non viceversa, poiché l'estensione incorporata non sa nulla di ciò che lo circonda).

8.4 Integrazione in una procedura guidata

Se la propria estensione fornisce una procedura guidata e una GUI, l'integrazione funziona sostanzialmente allo stesso modo. Si utilizzerà generalmente:

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

Se l'estensione integrata fornisce un'interfaccia guidata, le sue pagine saranno inserite tra `"pagina12"` e `"pagina13"` della propria estensione. Se l'estensione integrata fornisce solo un'interfaccia, una singola nuova pagina sarà aggiunta tra le proprie pagine `"pagina12"` e `"pagina13"`. L'utente non lo noterà mai.

8.5 Integrazione meno integrata: pulsante Altre opzioni

Anche se l'integrazione è interessante, si dovrebbe fare attenzione a non esagerare. Troppe funzioni all'interno di una GUI rendono difficile trovare le opzioni pertinenti. Certo, a volte si potrebbe voler integrare una grande quantità di opzioni (come tutte le opzioni per `plot()`), ma poiché queste sono davvero facoltative, non si dovrebbe mettere in risalto nella propria GUI.

Un'alternativa è integrare queste opzioni 'come un pulsante':

```
<dialog>
  <tabbook>
    [...]
    <tab label="Options">
      [...]
      <embed id="plotoptions" component="rkward:: ↵
        plot_options" as_button="true" label="Specifica ↵
        le opzioni del grafico"/>
    </tab>
    [...]
  </tabbook>
</dialog>
```

In questo caso, un singolo pulsante sarà aggiunto all'estensione, etichettato **Specifica le opzioni di tracciamento dei grafici**. Quando si preme tale pulsante, apparirà una finestra separata, con tutte le opzioni dell'estensione integrata. Anche mentre questa GUI integrata non è visibile per la maggior parte del tempo, puoi recuperarne le impostazioni proprio come descritto [sopra](#).

CAUTELA

Probabilmente l'approccio del 'pulsante' dovrebbe essere utilizzato esclusivamente per le estensioni che non possono mai essere non valide (per impostazioni mancanti/errate). Altrimenti l'utente non sarebbe in grado di inviare il codice, ma potrebbe avere difficoltà a trovarlo, il motivo per cui è nascosto dietro un pulsante.

8.6 Integrazione/definizione di estensioni incomplete

Alcune estensioni - e in effetti, le `plot_options` usate sopra come esempio, sono una di queste - non sono complete da sole. Semplicemente non hanno elementi della GUI per selezionare alcuni valori importanti. Sono pensate per essere utilizzate solo integrate in altre estensioni.

In che misura l'estensione `plot_options` è incompleta? Bene, per alcune impostazioni delle opzioni, deve conoscere il nome degli oggetti/espressioni per gli assi x e y (in realtà andrà bene se ne ha solo uno, ma ha bisogno di almeno uno per funzionare correttamente). Tuttavia, non ha un meccanismo per selezionare quegli oggetti o inserirli in altro modo. Quindi come li conosce?

Nella sezione logica dell'estensione `plot_options` ci sono due righe aggiuntive, non ancora coperte:

```
<logic>

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

  [...]

</logic>
```

Introduzione alla scrittura delle estensioni per RKWard

Questo definisce due proprietà aggiuntive nell'estensione `plot_options`, il cui unico scopo è quello di essere collegato ad alcune proprietà (ancora sconosciute) dell'estensione di integrazione. Nell'estensione `plot_options` queste due proprietà sono semplicemente usate come qualsiasi altra, e per esempio ci sono chiamate a `getString("xvar")` nel modello JS di `plot_options`.

Ora, per l'estensione incompleta non c'è modo di sapere dove sarà integrata e come saranno chiamate le impostazioni rilevanti nell'estensione di integrazione. Quindi dobbiamo aggiungere anche due righe aggiuntive nella sezione logica dell'estensione di integrazione:

```
<logic>
    [...]

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

In linea di principio questa non è una novità, abbiamo trattato le istruzioni **<connect>** nel capitolo [di logica della GUI](#). Connettere semplicemente i valori in due varslot (chiamati `"xvarslot"` e `"yvarslot"` in questo esempio) alle proprietà 'esterne' riceventi dell'estensione incorporata. Questo è tutto. Tutto il resto viene gestito automaticamente.

Capitolo 9

Gestire molte estensioni singole

9.1 Panoramica su diversi approcci

A volte, si potrebbe voler sviluppare le estensioni per una serie di funzioni simili. Ad esempio, considerare i grafici di distribuzione. Questi generano un codice abbastanza simile e, naturalmente, è desiderabile rendere le interfacce grafiche simili tra loro. Infine, ampie sezioni dei file della guida possono essere identiche. Solo alcuni parametri sono diversi per ogni estensione.

L'approccio banale a questo è quello di sviluppare un'estensione, quindi sostanzialmente copiare e incollare l'intero contenuto dei file `.js`, `.xml` e `.rkh`, quindi modificando le poche parti che sono diverse. Tuttavia, cosa succede se qualche tempo dopo si individua un errore di ortografia che è stato copiato e incollato su tutte le estensioni? E se si volesse aggiungere il supporto per una nuova funzionalità? Sarebbe necessario rivedere tutte le estensioni e cambiare ognuna di esse. Un processo faticoso e noioso.

Un secondo approccio potrebbe essere quello di utilizzare l'**incorporamento**. Tuttavia, in alcuni casi questo non si presta bene al problema in questione, soprattutto perché i 'pezzi' che puoi incorporare sono talvolta troppo grandi per essere utili e pone alcuni vincoli sul layout. In questi casi, i concetti [che includono file .js](#), [che includono file .xml](#) e [frammenti](#) possono essere molto utili (ma vedere le [considerazioni su quando è preferibile utilizzare l'incorporamento](#)).

Un avvertimento, prima di iniziare a leggere, però: questi concetti possono aiutare a semplificare la gestione di molte estensioni simili e possono migliorare la manutenibilità e la leggibilità delle estensioni stesse. Tuttavia, esagerare può facilmente portare all'effetto contrario. Utilizzare con una certa cautela.

9.2 Utilizzare l'istruzione `include` di JS

Si può facilmente includere un file di script in un altro nelle estensioni di RKWard. Il valore di questo diventa immediatamente ovvio, se alcune sezioni del codice JS sono simili tra le estensioni. È possibile semplicemente definire quelle sezioni in un file `.js` separato e includerlo in tutti i file `.js` dell'estensione. Ad esempio, come in:

```
// questo è un file chiamato «common_functions.js»

function doCommonStuff () {
  // forse prende qualche opzione, ecc.
  // ...
  comment ("Questo è il codice R che vuoi in diverse estensioni\n");
  // ...
}
```

Introduzione alla scrittura delle estensioni per RKWard

```
// questo è uno dei file .js delle tue normali estensioni

// includi le funzioni comuni
include ("common_functions.js");

function calculate () {
    // fai qualcosa
    // ...

    // inserisci il codice comune
    doCommonStuff ();
}
```

Notare che a volte è ancora più utile invertire questa situazione e definire lo ‘scheletro’ delle funzioni `preprocess()`, `calculate()` e `printout()` in un file comune e fare in modo che queste richiamino quelle parti che sono diverse tra le estensioni. Ad es:

```
// questo è un file chiamato "common_functions.js"

function calculate () {
    // fare qualcosa che è uguale in tutte le estensioni
    // ...

    // aggiungere qualcosa che è diverso tra le estensioni
    getSpecifics ();

    // ...
}
```

```
// questo è uno dei normali file .js dell'estensione

// includi le funzioni comuni
include ("common_functions.js");

// nota: nessuna funzione calculate() è definita qui.
// è, invece, la common_functions.js.

function getSpecifics() {
    // stampa codice R
}
```

Un problema da tenere presente quando si utilizza questa tecnica è l'ambito delle variabili. Vedere il manuale di JS sugli ambiti delle variabili.

Questa tecnica è ampiamente utilizzata nel grafico di distribuzione e nelle estensioni con creazione di grafici TLC di distribuzione, quindi si potrebbe voler cercare degli esempi.

9.3 Includere i file .xml

Fondamentalmente la stessa funzionalità di includere file è disponibile anche per l'uso nei file .xml, .pluginmap e .rkh. In qualsiasi punto di questi file è possibile inserire un tag `<include>` come mostrato di seguito. L'effetto è che l'intero contenuto di quel file XML (per essere precisi: tutto ciò che è contenuto nel tag `<document>` di quel file) è incluso parola per parola a questo punto nel file. Tenere presente che è possibile includere solo un altro file XML.

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

L'attributo *file* è il nome file relativo alla cartella in cui si trova il file attuale.

9.4 Utilizzare <snippets>

Sebbene includere i file come mostrato nella [sezione precedente](#) sia abbastanza potente, diventa molto utile se usato in combinazione con **<snippets>**. Gli snippet sono sezioni molto più piccole che è possibile inserire in un altro punto del file. Un esempio lo illustra meglio:

```
<document>
  <snippets>
    <snippet id="note">
      <frame>
        <text>
          Questo sarà inserito in due posizioni nell'interfaccia
        </text>
      </frame>
    </snippet>
  </snippets>
  <dialog label="test">
    <column>
      <insert snippet="note"/>
      [...]
      <insert snippet="note"/>
    </column>
  </dialog>
</document>
```

Quindi, definire lo snippet in un punto all'inizio del file XML e quindi **<insert>** in qualsiasi punto si desideri.

Anche se questo esempio non è molto utile di per sé, pensa a combinarlo con un file **<include>**d .xml. Tenere presente che è possibile inserire anche frammenti per il file .rkh nello stesso file. Si dovrebbe semplicemente **<include>** anche il file lì e **<inserire>** il frammento pertinente:

```
<!-- Questo è un file chiamato "common_snippets.xml" -->
<document>
  <snippet id="opzioni_comuni">
    <spinbox id="qualcosa" [...] />
    [...]
  </snippet>
  <snippet id="nota_comune">
    <text>Una nota importante per questo tipo di estensione</ ↔
    text>
  </snippet>

  <snippet id="guida_comune">
    <setting id="qualcosa">Questa fa qualcosa>
    [...]
  </snippet>
</document>
```

Introduzione alla scrittura delle estensioni per RKWard

```
<!-- Questo è il file .xml dell'estensione -->
<document>
  <snippets>
    <!-- Importa i frammenti comuni -->
    <include file="common_snippets.xml"/>
  </snippets>

  <dialog label="test2">
    <insert snippet="nota_comune"/>
    <spinbox id="qualcosa_specifica_della_estensione" [...] />
    <insert snippet="opzioni_comuni"/>
  </dialog>
</document>
```

Simile a [inclusione in JS](#), l'approccio inverso è spesso ancora più utile:

```
<!-- Questo è un file chiamato "common_layout.xml" -->
<document>
  <column>
    <insert snippet="note">
      [...]
    <insert snippet="plugin_parameters">
      [...]
    </column>
  [...]
</document>
```

```
<!-- Questo è il file .xml dell'estensione -->
<document>
  <snippets>
    <snippet id="nota">
      <text>La nota utilizzata per questa estensione ←
        specifica</text>
    </snippet>

    <snippet id="parametri_estensione">
      <frame label="Parametri specifici di questa ←
        estensione">
        [...]
      </frame>
    </snippet>
  </snippets>

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

Infine, è anche possibile **<inserire>** frammenti in altri frammenti, a condizione che: a) ci sia un solo livello di annidamento e b) la sezione **<snippets>** sia posizionata all'inizio del file (prima che venga inserito un frammento annidato); questo perché le istruzioni **<insert>** vengono risolte dall'alto verso il basso.

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

A prima vista, **<include>** e **<snippets>** forniscono funzionalità piuttosto simili all'[incorporamento](#): consente di riutilizzare alcune porzioni di codice tra le estensioni. Allora

qual è la differenza tra questi approcci e quando dovresti usarli?

La differenza fondamentale tra questi concetti è che le estensioni integrabili sono un fascio più stretto. Esse combinano il codice completo di una GUI per generare codice R da questo, e una pagina di aiuto. Al contrario, `include` e `insert` consentono un controllo molto più granulare, ma al prezzo di una minore modularità.

Cioè, un'estensione che incorpora un'altra estensione in genere non avrà bisogno di sapere molto sui dettagli interni dell'estensione incorporata. Un ottimo esempio è l'estensione `plot_options`. Le estensioni che desiderano incorporarla non devono necessariamente conoscere tutte le opzioni fornite o come vengono fornite. Questa è una buona cosa, altrimenti una modifica nell'estensione `plot_options` potrebbe rendere necessario modificare tutte le estensioni che la incorporano (molte). Al contrario, `include` e `insert` espongono davvero tutti i dettagli interni e le estensioni che li utilizzano, ad esempio, dovranno conoscere gli ID esatti e forse anche il tipo degli elementi utilizzati.

Quindi la regola pratica è questa: `include` e `insert` sono ottimi se le opzioni pertinenti sono necessarie solo per un gruppo chiaramente limitato di estensioni. Le estensioni incorporate sono migliori, se il gruppo di estensioni a cui può essere utile non è chiaramente definito e se la funzionalità può essere facilmente modularizzata. Un'altra regola pratica: se potete mettere le parti comuni in un singolo 'blocco', allora fatelo e usate l'incorporamento. Se hai bisogno di molti piccoli frammenti per definire le parti comuni, beh, usate **<snippets>**. Un ultimo modo di vedere la cosa: se tutte le estensioni forniscono funzionalità *altamente* simili, gli `include` e gli `insert` sono probabilmente una buona idea. Se condividono semplicemente uno o due 'moduli' comuni, probabilmente è meglio incorporarli.

Capitolo 10

Concetti per l'uso in estensioni specializzate

Questo capitolo contiene informazioni su alcuni argomenti utili solo a determinate classi di estensioni.

10.1 Estensioni che producono un grafico

Creare un grafico da un'estensione è facile da fare. Tuttavia, ci sono alcuni piccoli inconvenienti da evitare, e anche alcune grandi funzionalità generiche di cui si dovrebbe essere a conoscenza. Questa sezione mostra i concetti di base e termina con un esempio canonico che si dovrebbe seguire ogni volta che si crea un'estensione di grafici.

10.1.1 Tracciare un grafico nella finestra del risultato

Per disegnare un grafico nella finestra di output, utilizzare `rk.graph.on()` direttamente prima di creare il grafico e `rk.graph.off()`, subito dopo. Questo è simile a ad es. chiamando `postscript()` e `dev.off()` in un normale sessione di R.

È importante sottolineare, tuttavia, che è necessario *sempre* invocare `rk.graph.off()` dopo aver invocato `rk.graph.on()`. Altrimenti il file di output sarà lasciato in uno stato incoerente. Per garantire che `rk.graph.off()` venga realmente invocato, dovrete racchiudere *tutti* i comandi di R tra le due chiamate nell'istruzione `try()`. Non ne avete mai sentito parlare? Nessun problema, è facile. Tutto quello che dovete fare è seguire lo schema mostrato nell'[esempio](#), di seguito.

10.1.2 Aggiungere una funzionalità di anteprima

NOTA

Questa sezione discute l'aggiunta di funzionalità di anteprima alle estensioni che producono grafici. Ci sono sezioni separate su [anteprime dell'output \(HTML\)](#), [anteprime dei dati \(importati\)](#) e [anteprime personalizzate](#). Tuttavia, si consiglia di leggere prima questa sezione poiché l'approccio è simile in ogni caso.

Una funzionalità molto utile per tutti le estensioni che generano un grafico/traccia è fornire un'anteprima che si aggiorna automaticamente. Per farlo, saranno necessarie due cose: aggiungere una casella di controllo **<preview>** alla propria [GUI definizione](#) e modificare il [codice generato](#) per l'anteprima.

Aggiungere una casella di controllo **<anteprima>** è semplice. Basta inserire il seguente comando da qualche parte nella GUI. Si occuperà di tutta la magia dietro le quinte della creazione di un dispositivo di anteprima, aggiornando l'anteprima ogni volta che le impostazioni cambiano, ecc. Esempio:

NOTA

A partire dalla versione 0.6.5 di RKWard **<preview>**, gli elementi di anteprima sono trattati in modo speciale nelle finestre delle estensioni (non nelle procedure guidate): saranno posizionati nella colonna dei pulsanti, indipendentemente da dove siano definiti nell'interfaccia utente. È comunque consigliabile definirli in un punto sensato del layout, per garantire la retrocompatibilità.

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

E questo è tutto per la definizione della GUI.

La regolazione del modello JS richiede solo un po' più di lavoro, qui dovreste assicurarvi che venga generata solo il grafico stesso e mostrato in un dispositivo su schermo, anziché indirizzato all'output, cioè nessuna stampa di intestazioni, `rk.graphics.on()` o chiamate simili. Per aiutarvi in questo, RKWard invocherà le funzioni `preprocess()`, `calculate()` e `printout()` con un parametro aggiuntivo impostato su `true`, durante la generazione del codice per un'anteprima. (Il parametro viene omissso durante la generazione del codice finale. In javascript questo risulterà `false` se utilizzato all'interno di un'istruzione `if`.) Vedere l'[esempio](#), di seguito per il modello tipico che utilizzerete.

In alternativa, se hai bisogno di un controllo maggiore, puoi aggiungere una nuova funzione chiamata `preview()` al tuo modello JS e generare lì il codice necessario per un'anteprima (probabilmente, almeno in parte, chiamando nuovamente `calculate()`, ecc.).

10.1.3 Opzioni generiche grafico

Avrete notato che la maggior parte delle estensioni di tracciamento dei grafici in RKWard fornisce una vasta gamma di opzioni generiche ad es. per personalizzare i titoli degli assi o i margini delle figure. Aggiungere queste opzioni alla propria estensione è semplice. Sono fornite da un'estensione [integrabile](#) chiamata **`rkward::plot_options`**. Integra questa estensione nell'interfaccia della propria estensione in questo modo:

```
<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"/>
```

Introduzione alla scrittura delle estensioni per RKWard

```
<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="Opzioni ↵
    del grafico"/>
  [...]
</dialog>
[...]
```

Ciò aggiungerà un pulsante all'interfaccia utente per visualizzare una finestra con le opzioni di stampa. La sezione logica è solo un esempio. Consente un certo controllo sull'estensione delle opzioni del grafico. Maggiori informazioni nella pagina della guida dell'estensione `plot_options` (collegata dalla pagina della guida di qualsiasi estensione che fornisce le opzioni generiche).

Successivamente è necessario assicurarsi che il codice corrispondente alle opzioni del grafico sia aggiunto al codice generato per il grafico. Per farlo, recuperare le proprietà **`code.preprocess`**, **`code.printout`** e **`code.calculate`** e inserirle nel proprio codice, come mostrato nel [esempio](#), di seguito.

10.1.4 Un esempio classico

Ecco un esempio di file .JS che si dovrebbe utilizzare come modello, ogni volta che si crea un'estensione con creazione di grafici:

```
function preprocess () {
  // «unpacchetto» è necessario per creare il grafico
  echo ("require (unpacchetto)\n");
}

function printout (is_preview) {
  // Se «is_preview» è impostata a falso, genera il codice completo, ↵
  include le intestazioni.
  // Se «is_preview» è impostata a vero, sarà generata solo la parte ↵
  essenziale.

  if (!is_preview) {
    echo ('rk.header (' + il8n ("Un grafico di esempio") + ')\n\n');
    echo ('rk.graph.on ()\n');
  }
  // sarà generata solo la seguente sezione per is_preview==true

  // ricorda: qualsiasi cosa tra rk.graph.on() e rk.graph.off() dovrebbe ↵
  essere inglobata in un'istruzione try():
  echo ('try ({\n');
  // inserisci qualsiasi codice di definizione delle opzioni che dovrebbe ↵
  essere eseguito prima dei comandi effettivi di tracciamento dei ↵
  grafici.
  // Il codice stesso è fornito dall'estensione delle opzioni di ↵
  tracciamento dei grafici. printIndentedUnlessEmpty() si occupa di ↵
  rendere piacevole la formattazione.
  printIndentedUnlessEmpty ('\\t', getString ("plotoptions.code.preprocess ↵
    "), '', '\\n');
```

```
// crea il grafico. plotoptions.code.printout fornisce la parte delle ←  
  opzioni generiche dei grafici  
// che devono essere aggiunte alla chiamata di tracciamento, stessa.  
echo ('plot (5, 5' + getString ("plotoptions.code.printout") + ')\n');  
  
// inserisci qualsiasi codice di definizione delle opzioni che dovrebbe ←  
  essere eseguito dopo il grafico effettivo.  
printIndentedUnlessEmpty ('\t', getString ("plotoptions.code.calculate ←  
  "), '\n');  
echo ('}')'\n'); // la chiusura dell'istruzione try()  
  
if (!is_preview) {  
  echo ('rk.graph.off ()\n');  
}  
}
```

10.2 Anteprime dei dati, uscita e altri risultati

10.2.1 Anteprima del risultato (HTML)

NOTA

In questa sezione viene descritta l'aggiunta della funzionalità di anteprima alle estensioni che creano stampe del risultato / HTML. Si consiglia di leggere la sezione separata su [anteprime dei grafici](#), prima di questa sezione.

La creazione di un'anteprima del risultato HTML è quasi la stessa procedura della creazione di un'anteprima del grafico. In questo caso, assicurarsi semplicemente che **preview()** generi i relativi comandi **rk.print()/rk.results()**. In genere, tuttavia, è consigliabile omettere le istruzioni di intestazione nell'anteprima. Ecco un esempio ridotto:

```
<!-- Nel file XML dell'estensione -->  
  <dialog label="Importa dati CSV" >  
    <browser id="file" type="file" label="Nome file"/>  
    <!-- [...] -->  
    <preview id="preview" mode="output"/>  
  </dialog>  
>
```

Nota la specifica di *mode="output"* in **<preview>**.

```
// Nel file JS dell'estensione  
function preview () {  
  // genera il codice utilizzato per l'anteprima  
  printout (true);  
}  
  
function printout (is_preview) {  
  // genera un'intestazione solo se is_preview==false  
  if (!is_preview) {  
    new Header ("Questa è una didascalia").print ();  
  }  
  echo ('rk.print (result)');  
}
```

10.2.2 Anteprima dei dati (importati)

NOTA

Questa sezione discute l'aggiunta di funzionalità di anteprima alle estensioni che creano (importano) dati. Si consiglia di leggere la sezione separata sulle [anteprime dei grafici](#), prima di questa sezione.

La creazione di un'anteprima dei dati importati (qualsiasi tipo di dati che **rk.edit()** è in grado di gestire), è molto simile alla creazione di [anteprima del grafico](#). Il seguente esempio ridotto dovrebbe aiutare a illustrare come creare un'anteprima dei dati:

```
<!-- Nel file XML dell'estensione -->
<dialog label="Importa dati CSV" >
  <browser id="file" type="file" label="Nome file"/>
  <!-- [...] -->
  <preview id="preview" active="true" mode="data"/>
</dialog>
>
```

Questa volta l'elemento **<preview>** specifica `mode="data"`. `active="true"` rende semplicemente attiva l'anteprima per impostazione predefinita.

```
// Nel file JS dell'estensione
function preview () {
  // genera il codice utilizzato per l'anteprima
  calculate (true);
}

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

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

Ancora una volta, la funzione **preview()** genera quasi lo stesso codice R come la funzione **calculate()**, quindi creiamo una funzione di supporto **doCalculate()** per escludere le parti comuni. La cosa più importante da notare è che dovrete assegnare i dati importati a un oggetto chiamato `preview_data` (all'interno dell'ambiente locale attuale). *Tutto il resto avverrà automaticamente* (in parole povere, RKWard chiamerà **rk.edit(preview_data)**, racchiuso in una chiamata a **.rk.with.window.hints()**).

NOTA

Sebbene le anteprime siano un'ottima funzionalità, consumano risorse. Nel caso delle anteprime dei dati, potrebbero esserci casi in cui le anteprime possono causare problemi di prestazioni significativi. Ciò potrebbe riguardare l'importazione di insiemi di dati enormi (che sono semplicemente troppo grandi per essere aperti per la modifica nella finestra dell'editor di RKWard), ma anche insiemi di dati «normali» potrebbero essere importati erroneamente, creando un numero enorme di righe o colonne. *Si consiglia vivamente di limitare `preview_data` a una dimensione che fornisca un'anteprima utile, senza il pericolo di creare notevoli problemi di prestazioni (ad es. 50 righe per 50 colonne dovrebbero essere più che sufficienti nella maggior parte dei casi).*

10.2.3 Anteprime personalizzate

L'elemento `<preview>` può essere utilizzato per creare anteprime per qualsiasi tipo di finestra «documento» che può essere collegata all'area di lavoro di RKWard. Oltre ai [grafici](#) e alle [finestre di dati](#), questo include file HTML, R script e finestre di riepilogo degli oggetti. Per questi ultimi, dovrete utilizzare `<preview mode="custom">`.

Se avete letto le sezioni che descrivono l'anteprima della trama e l'anteprima dei dati, dovrete avere un'idea generale della procedura, ma le anteprime «personalizzate» richiedono un po' più di lavoro manuale dietro le quinte. La funzione più importante di R da guardare è `rk.assign.preview()`, qui. Il seguente breve elenco mostra cosa hai generato (anteprima) il codice di R potrebbe essere simile a quello di una estensione che crea un file di testo in uscita:

```
## Da generare nella sezione di codice preview() di un'estensione
pdata <- rk.get.preview.data("UN_ID")
if (is.null (pdata)) {
  outfile <- rk.get.tempfile.name(prefix="anteprima", ←
    extension=".txt")
  pdata <- list(filename=outfile, on.delete=function (id) {
    unlink(rk.get.preview.data(id)$filename)
  })
  rk.assign.preview.data("UN_ID", pdata)
}
try ({
  cat ("Questo è un test", pdata$filename)
  rk.edit.files(file=pdata$filename)
})
```

Qui si dovrebbe ottenere il valore `UN_QUALCHE_ID` dalla proprietà `id` dell'elemento `<preview>`. Cioè utilizzando `getString("preview.id")` nel file `.js` dell'estensione.

10.3 Estensioni dipendenti dal contesto

Finora abbiamo assunto, tutte le estensioni sono sempre significative e tutte inserite nel menu principale. Tuttavia, alcune estensioni sono significative solo (o in aggiunta) in un determinato contesto. Ad esempio un'estensione per esportare i contenuti di un dispositivo grafico X11 di R è ovviamente più utile, se inserita nel menu di un dispositivo X11, non nella barra dei menu principale. Inoltre, un'estensione di questo tipo dovrebbe conoscere il numero di dispositivo sul quale dovrebbe operare, senza che questo debba essere richiesto all'utente.

Chiamiamo tali estensioni dipendenti dal contesto. Di conseguenza, nel file `.pluginmap`, non sono (o non solo) posizionati nell'elemento `<hierarchy>` principale ma piuttosto in un elemento `<context>`. Finora sono supportati solo due contesti diversi (ne arriveranno altri in seguito): `x11` e importazione di file. Ci occuperemo di questi a turno. Anche se siete interessati solo al contesto di importazione, leggete anche la sezione sul contesto `x11`, poiché è leggermente più elaborata.

10.3.1 Contesto del dispositivo X11

Per utilizzare un'estensione nel contesto di un dispositivo x11, ovvero posizionarla nella barra dei menu della finestra che si apre quando si invoca `x11()` nella console, bisogna prima dichiararlo come di consueto nel file `.pluginmap`:

```
<document [...]>
  <components>
    [...]
    <component id="my_x11_plugin" file="my_x11_plugin.xml" ↵
      label="Un'estensione del contesto X11"/>
    [...]
  </components>
```

Tuttavia, non è necessario definirla nella gerarchia (è possibile, se è anche significativa come estensione di livello superiore):

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

Invece, aggiungere una definizione del contesto «x11» e aggiungerla ai menu:

```
<context id="x11">
  [...]
  <menu id="edit">
    [...]
    <entry id="mia_estensione_x11"/>
  </menu>
</context>
</document>
```

Nella [sezione logica del xml dell'estensione](#), ora è possibile dichiarare due proprietà **<external>**: `devnum` e `context`. `context` (se dichiarato) sarà impostato su `"x11"` quando l'estensione viene invocata in questo contesto. `devnum` sarà impostato sul numero del dispositivo grafico su cui operare. E questo è tutto.

10.3.2 Importa contesto di dati

Prima di leggere questa sezione, assicurarsi di leggere la sezione sul [contesto del dispositivo X11](#), poiché ciò spiega i concetti di base.

Il contesto `"import"` viene utilizzato per dichiarare le estensioni di filtro dei file di importazione. Basta inserirli in un contesto con `id="import"` nel file `.pluginmap`. Tuttavia, c'è un'ulteriore svolta quando si dichiarano queste estensioni: per offrire una finestra unificata per la selezione dei file per tutti i tipi di file supportati, è necessario dichiarare un'ulteriore informazione sul componente:

```
<document [...]>
  <components>
    [...]
    <component id="mia_estensione_xyz_importazione" file=" ↵
      mia_estensione_xyz_importazione.xml" label="Importa ↵
      file XYZ">
      <attribute id="format" value="*.xyz *.zyx" label=" ↵
        File di dati XYZ"/>
    </component>
    [...]
  </components>
```

```

</components>
<hierarchy>
  [...]
</hierarchy>
<context id="import">
  [...]
  <menu id="import">
    [...]
    <entry id="mia_estensione_xyz_importazione"/>
  </menu>
</context>
  [...]
</document>

```

La riga dell'attributo indica semplicemente che le estensioni associate ai file XYZ sono *.xyz o *.zyx e che il filtro dovrebbe essere etichettato come 'file di dati XYZ' nella finestra di selezione dei file.

È possibile dichiarare due proprietà **<external>** nella propria estensione. *filename* sarà impostato sul nome del file selezionato e *context* sarà impostato su "import".

10.4 Interrogazione delle informazioni di R

In alcuni casi, si potrebbe voler recuperare ulteriori informazioni da R, da presentare nell'interfaccia utente dell'estensione. Ad esempio, si potrebbe voler offrire una selezione dei livelli di un fattore che l'utente ha selezionato per l'analisi. Dalla versione 0.6.2 di RKWard è possibile farlo. Prima di iniziare, è importante essere a conoscenza di alcuni avvertimenti:

Il codice R eseguito dall'interno della logica dell'interfaccia utente dell'estensione viene valutato nel ciclo di eventi di R, il che significa che può essere eseguito *mentre* sono in esecuzione altri calcoli. Questo serve a garantire che l'interfaccia utente dell'estensione sia utilizzabile anche mentre R è impegnato in altre attività. Tuttavia, è fondamentale che il codice non abbia effetti collaterali. In particolare:

- *Non* fare alcuna assegnazione .GlobalEnv o in qualsiasi altro ambiente non locale.
- *Non* stampare nulla sul file di uscita.
- *Non* tracciare nulla sullo schermo.
- In generale, *non* fare qualcosa che abbia effetti collaterali. Il proprio codice può *leggere informazioni*, non «fare» qualsiasi cosa.

Con questo in mente, ecco lo schema generale. Lo si utilizzerà all'interno di una sezione **logica dell'interfaccia utente programmabile**:

```

<script><![CDATA[
    let update = gui.addChangeCommand (" ←
        variable", function () {
            gui.setValue ("selector.enabled", ←
                0);
            variable = gui.getValue ("variable ←
                ");
            if (variable == "") return;

            new RCommand('levels (' + variable ←
                + ')', "myid").then(result =
> {

```

```

gui.setValue ("selector. ←
  enabled", 1);
gui.setListValue ("selector ←
  .available", result);
}).catch(msg =

> {

    if (msg === "outdated") ←
      return; // il comando è ←
      stato annullato, perché ←
      ne sta per arrivare uno ←
      nuovo -> benign
    // possibilmente altra ←
    gestione degli errori, ←
    msg contiene gli avvisi ←
    e i messaggi di errore ←
    generati,
    // se il comando non è ←
    andato a buon fine ad es ←
    ..
    gui.setListValue ("selector ←
      .available", Array (" ←
      ERROR:", msg));

    });

  });
]]></script>

```

Qui, *variable* è una proprietà che contiene un nome di oggetto (ad es. all'interno di un **<varslot>**). Ogni volta che ciò cambia, vorrete aggiornare la visualizzazione dei livelli all'interno del **<valueselector>**, chiamato *selector*. La funzione chiave qui è l'istruzione del costruttore **new RCommand()**, che prende come primo parametro la stringa di comando da eseguire. Tenere presente che il comando viene eseguito in modo asincrono e questo rende le cose un po' più complesse. Per prima cosa assicurarsi che il **<valueselector>** rimanga disabilitato, mentre non contiene informazioni aggiornate. In secondo luogo, poiché l'utente può apportare modifiche rapidamente, è possibile che venga generato più di un comando prima di ricevere qualsiasi risultato. Pertanto, dovrete assicurarvi di agire solo sul comando più recente.

Per gestire l'asincronia, ciò che viene restituito, qui, è un oggetto **Promise**. Maggiori informazioni su questa potente funzionalità JavaScript possono essere trovate [su Internet](#). La cosa importante da sapere, qui, è che l'aggiunta di un'istruzione **.then()** consente di specificare cosa accadrà al termine del comando e che un'istruzione **.catch()** può essere utilizzata per gestire eventuali errori. Ancora una volta, tenere presente che il blocco **.then()** non viene eseguito immediatamente. Per comprenderne le implicazioni, potrebbe essere utile, durante lo sviluppo, inserire un **Sys.sleep(1)**; nel comando R, per vedere cosa succede, quando un comando non viene completato, immediatamente.

Infine, per gestire la generazione di più comandi, è possibile specificare un secondo argomento per **new RCommand()**, ("myid", in questo esempio). Tutti i comandi con lo stesso identificatore (scelto liberamente) saranno considerati appartenenti alla stessa coda. RKWard si assicurerà quindi che solo il comando più recente attiverà effettivamente il blocco **.then()**, mentre eventuali comandi obsoleti arriveranno nel blocco **.catch()**. Qui è possibile identificare i comandi obsoleti, poiché come valore viene passata la stringa "obsoleto", mentre per eventuali altri possibili errori vengono passati gli avvisi e i messaggi di errore.

Notare che questo esempio è in qualche modo semplificato. In realtà dovrete prendere ulteriori precauzioni, ad es. per evitare di inserire un numero eccessivo di livelli nel selettore. La buona notizia è che probabilmente non dovrete fare tutto questo da soli. L'esempio sopra è preso dall'estensione **rkward::level_select**, ad esempio, che potete semplicemente [incorporare](#) nella vostra estensione. Ciò vi consente anche di specificare un'espressione diversa da eseguire al posto di **levels()**.

NOTA

Nelle versioni precedenti di RKWard, i comandi R venivano eseguiti utilizzando una funzione **doR-Command()** leggermente più complessa. Questa funzione potrebbe essere ancora presente in alcune estensioni, ma non è consigliabile utilizzarla nel nuovo codice.

10.5 Riferimento all'oggetto attuale o al file attuale

Per molte estensioni è preferibile lavorare sull'oggetto 'attuale'. Ad esempio, un'estensione di 'ordinamento' potrebbe pre-selezionare il `data.frame` attualmente in fase di modifica per l'ordinamento. Il nome dell'oggetto attuale è disponibile per le estensioni come proprietà predefinita chiamata `current_object`. Potete connettervi a questa proprietà nel solito modo. Se nessun oggetto è attuale, la proprietà equivale a una stringa vuota. Allo stesso modo, l'URL del file di script attuale è accessibile come proprietà predefinita denominata `current_filename`. Quella proprietà è vuota se nessun file di script è attualmente in fase di modifica o se il file di script non è stato ancora salvato.

Attualmente `current_object` può essere solo della classe `data.frame`, ma non fate affidamento su questo, poiché in futuro sarà esteso ad altri tipi di dati. Se siete interessati solo agli oggetti `data.frame`, connettetevi invece alla proprietà `current_dataframe`. In alternativa, puoi applicare i requisiti di tipo utilizzando i vincoli appropriati sui **<varslot>** o utilizzando lo [scripting logico della GUI](#).

10.6 Ripetere (un insieme di) opzioni

A volte potreste voler ripetere una serie di opzioni per un numero arbitrario di elementi. Per esempio, supponiamo che vogliate implementare un'estensione per ordinare un `data.frame`. Potreste voler consentire l'ordinamento in base a un numero arbitrario di colonne (in caso di parità tra le prime colonne). Ciò potrebbe essere realizzato semplicemente consentendo all'utente di selezionare più variabili in un **<varslot>** con `multi="true"`. Ma se volete estenderlo, ad es. consentendo all'utente di specificare per ciascuna variabile se deve essere convertita in carattere/numerico o se l'ordinamento deve essere ascendente o discendente, è necessaria maggiore flessibilità. Altri esempi potrebbero essere tracciare più linee in un unico grafico (permettendo di selezionare oggetto, stile di linea, colore di linea, ecc. per ogni linea), o specificare una mappatura per ricodificare da un insieme di vecchi valori a nuovi valori.

Immettere il **<optionset>**. Diamo un'occhiata a un semplice esempio, prima:

```
<dialog [...]>
  [...]
  <optionset id="set" min_rows="1">
    <content>
      <row>
        <input id="firstname" label="Nome proprio"  ←
          size="small">
        <input id="lastname" label="Cognome" size="  ←
          small">
        <radio id="gender" label="Genere">
          <optioncolumn label="Maschio" value  ←
            ="m"/>
          <optioncolumn label="Femmina" value  ←
            ="f"/>
        </radio>
      </row>
    </content>
```

Introduzione alla scrittura delle estensioni per RKWard

```
<optioncolumn id="firstnames" label="Nome proprio" connect ↔  
  ="firstname.text">  
<optioncolumn id="lastnames" label="Cognome" connect=" ↔  
  lastname.text">  
<optioncolumn id="gender" connect="gender.string">  
</optionset>  
[...]  
</dialog>
```

Qui abbiamo creato un'interfaccia utente per specificare un numero di persone (ad es. autori). L'interfaccia utente richiede almeno una voce (*min_rows="1"*). All'interno dell'elemento **<optionset>**, iniziamo specificando **<content>**, cioè quegli elementi che appartengono al set di opzioni. Avrete familiarità con la maggior parte degli elementi all'interno di **<content>**.

Successivamente specifichiamo le variabili di interesse che vorremo leggere dall'opzione impostata nel nostro file JS. Poiché avremo a che fare con un numero arbitrario di elementi, non possiamo semplicemente leggere `getString ("firstname")` in JS. Piuttosto, per ciascun valore di interesse, specifichiamo un **<optioncolumn>**. Per la prima colonna di opzioni nell'esempio, **<connect="firstname.text">** significa che il contenuto dell'elemento **<input>** "firstname" viene letto per ciascun elemento. Le **<optioncolumn>** per le quali viene fornita un'etichetta saranno visualizzate sullo schermo, in una colonna accanto a quell'etichetta. In JS, ora possiamo recuperare i nomi di tutti gli autori utilizzando `getList ("set.firstname")`, `getList ("set.lastnames")` per i nomi di famiglia e `getList ("set.gender")` per un array di stringhe "m"/"f".

Tenere presente che non ci sono restrizioni su cosa potete inserire all'interno di un **<optionset>**. Potete anche utilizzare componenti **incorporati**. Proprio come con qualsiasi altro elemento, tutto ciò che dovete fare è raccogliere le variabili di output di interesse in una specifica **<optioncolumn>**. Nel caso delle estensioni incorporate, questa è spesso una sezione della proprietà «code». Ad esempio:

```
<dialog [...]>  
  [...]  
  <optionset id="set" min_rows="1">  
    <content>  
      [...]  
      <embed id="color" component="rkward::color_chooser" ↔  
        label="Colore"/>  
    </content>  
  
    [...]  
    <optioncolumn id="color_params" connect="color.code. ↔  
      printout">  
  </optionset>  
  [...]  
</dialog>
```

Naturalmente è possibile anche utilizzare la **logica dell'interfaccia utente** all'interno di un insieme di opzioni. Ci sono due opzioni per farlo: potete farlo effettuando una connessione (o creando uno script) nella sezione **<logic>** principale della vostra estensione, come al solito. Tuttavia, accederete agli elementi dell'interfaccia utente nell'area dei contenuti come (ad es.) "set.contents.firstname.XYZ". Notare il prefisso "set" (l'*id* che avete assegnato agli insiemi e ai "contenuti"). In alternativa, potete aggiungere una sezione **<logic>** separata come elemento figlio del vostro **<optionset>**. In questo caso, gli *id* saranno indirizzati in relazione all'area del contenuto, ad es. "nome.XYZ". Solo l'elemento **<script>** non è consentito nella sezione logica di un insieme di opzioni. Se volete utilizzare lo scripting, dovrete utilizzare la sezione **<logic>** principale dell'estensione.

NOTA

Quando si esegue lo scripting della logica in un insieme di opzioni, tutto ciò che si può fare è accedere all'area del contenuto *attuale*. Pertanto, in genere, è significativo solo connettere tra loro gli elementi all'interno dell'area del contenuto. La connessione di una proprietà esterna a **<optionset>** a una proprietà all'interno dell'area contenuto può essere utile per l'inizializzazione. Tuttavia, la modifica dell'area del contenuto dopo l'inizializzazione *non* si applicherà agli elementi che l'utente ha già definito. Solo all'elemento attualmente selezionato nell'insieme.

10.6.1 Insiemi di opzioni «guidati»

Finora abbiamo considerato un **<optionset>** che fornisce pulsanti per aggiungere/rimuovere elementi. Tuttavia, in alcuni casi, è molto più naturale selezionare elementi al di fuori dell'**<optionset>** e fornire solo opzioni per personalizzare alcuni aspetti di ciascun elemento in un **<optionset>**. Ad esempio, supponiamo di voler consentire all'utente di tracciare più oggetti all'interno di un diagramma. Per ciascun oggetto, l'utente dovrebbe essere in grado di specificare il colore della linea. *Potreste* risolvere questo problema inserendo un **<varselector>** e un **<varslot>** all'interno dell'area **<content>**, consentendo all'utente di selezionare un elemento alla volta. Tuttavia, ciò significherebbe molti meno clic per l'utente, se si utilizza invece un **<varslot multi="true">** esterno **<optionset>**. Quindi collegherete questa selezione di oggetti a un cosiddetto set di opzioni «guidato». Ecco come:

```
<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="Oggetti da tracciare"/>
  <optionset id="set" keycolumn="var">
    <content>
      [...]
      <embed id="color" component="rkward::color_chooser" ←
        label="Colore della linea"/>
    </content>

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

Inizieremo guardando l'esempio in basso. Noterete che due specifiche **<optioncolumn>** hanno *external="true"*. Questo dice a RKWard che questi sono controllati dall'esterno del **<optionset>**. Qui, l'unico scopo della colonna opzioni «varnames» è fornire etichette di facile lettura nella visualizzazione dell'insieme di opzioni (è collegata al modificatore «shortname» della proprietà che contiene gli oggetti selezionati). Lo scopo della colonna opzioni «vars» è di fungere da colonna «chiave», come specificato da **<optionset keycolumn="vars"...**. Ciò significa che per ciascuna voce in questo elenco, l'insieme offrirà una serie di opzioni e le opzioni sono logicamente legate a queste voci. Questa colonna è collegata alla proprietà che contiene gli oggetti selezionati nel **<varslot>**. Cioè per ogni oggetto selezionato lì, **<optionset>** consentirà di specificare il colore della linea.

NOTA

La colonna esterna può anche essere *connessa* a proprietà all'interno della regione **<content>**. Tuttavia, è importante notare che le colonne di opzioni dichiarate *external="true"* non dovrebbero mai essere modificate dall'interno di **<optionset>**, e le colonne di opzioni dichiarate *external="false"* (il valore predefinito) non dovrebbero mai essere modificate dall'esterno di **<optionset>**.

10.6.2 Alternative: quando non utilizzare gli insiemi di opzioni

Gli insiemi di opzioni sono uno strumento potente, ma a volte possono fare più creare più problemi che altro, poiché aggiungono una notevole complessità, sia dal punto di vista di uno sviluppatore di estensioni, sia dal punto di vista di un utente. Quindi, consigliamo di pensarci due volte, quando si utilizzano. Ecco qualche consiglio:

- Per alcuni casi semplici, l'elemento **<matrix>** può fornire un'utile alternativa leggera.
- Non far fare troppo alla propria estensione. Abbiamo fornito l'esempio dell'utilizzo di un optionset per un'estensione per tracciare diverse linee all'interno di un grafico. Ma in generale non è una buona idea creare un'estensione che produrrà grafici individuali per ogni elemento in un optionset. Piuttosto, fare in modo che l'estensione produca un grafico e l'utente possa invocarla più volte.
- Se non si attendono più di due o tre elementi in un insieme, considerare invece di ripetere manualmente le opzioni.

Capitolo 11

Gestione delle dipendenze e problemi di compatibilità

11.1 Compatibilità di versione di RKWard

Facciamo del nostro meglio per assicurarci che le estensioni sviluppate per una vecchia versione di RKWard rimangano funzionanti nelle versioni successive di RKWard. Tuttavia, non è sempre vero il contrario poiché vengono aggiunte nuove funzionalità. Poiché non tutti gli utenti eseguono l'ultima versione di RKWard, ciò significa che la propria estensione potrebbe non funzionare per tutti.

Quando siete a conoscenza di tali problemi di compatibilità, dovrete assicurarvi di documentarlo nel file `.pluginmap`, utilizzando l'elemento `<dependencies>`. `<dependencies>` può essere specificato come figlio diretto dell'elemento `<document>` di `.pluginmap` o come elemento figlio di singole definizioni `<component>`. Nel primo caso, le dipendenze si applicano a *tutte* le estensioni nella mappa. In quest'ultimo caso solo ai singoli `<component>`. Potete anche combinare le principali dipendenze «globali» e «specifiche». In questo caso le dipendenze «globali» si sommano a quelle del singolo componente.

Vediamo un piccolo esempio:

```
<document ...>
  <dependencies rkward_min_version="0.5.0c" />
  <components ...>
    <component id="myplugin" file="↔
      versione_ridotta_della_mia_estensione.xml" ...>
      <dependencies rkward_max_version="0.6.0z" />
    </component>
    <component id="mia_estensione" file="↔
      versione_precisa_della_mia_estensione.xml" ...>
      <dependencies rkward_min_version="0.6.1" />
    </component>
    ...
  </components ...>
</document>
```

In questo esempio, tutte le estensioni richiedono almeno la versione 0.5.0c di RKWard. Un'estensione, con `id="myplugin"`, è disponibile in due varianti alternative. La prima versione, semplificata, sarà utilizzata per le versioni di RKWard precedenti alla 0.6.1. La seconda utilizza funzionalità nuove in RKWard 0.6.1 e sarà utilizzata solo a partire dalla versione 0.6.1.

Introduzione alla scrittura delle estensioni per RKWard

Fornire varianti alternative come questa è un modo molto intuitivo per utilizzare nuove funzionalità, pur mantenendo il supporto per le versioni precedenti di RKWard. Le versioni alternative dovrebbero condividere lo stesso *id* (in caso contrario saranno generati avvisi) e possono essere definiti *solo all'interno dello stesso file* di `.pluginmap`.

Le estensioni che non sono compatibili con la versione in esecuzione di RKWard, e che non vengono forniti con una versione alternativa, saranno ignorati con un avviso.

NOTA

In realtà RKWard 0.6.1 è la prima versione che interpreta le dipendenze e che segnala errori di dipendenza. Pertanto, contrariamente a quanto può suggerire l'esempio, specificare eventuali versioni precedenti nelle dipendenze non avrà alcun effetto diretto (ma potrebbe comunque essere una buona idea ai fini della documentazione).

A volte sarà persino possibile gestire i problemi di incompatibilità di versione *all'interno* di un singolo file `.pluginmap`, utilizzando l'elemento `<dependency_check>`, descritto nella sezione seguente.

11.2 Compatibilità versione di R

Analogamente a `rkward_min_version` e `rkward_max_version`, l'elemento `<dependencies>` consente di specificare gli attributi `R_min_version` e `R_max_version`. Tuttavia, ci sono le seguenti differenze:

- Le estensioni che non soddisfano i requisiti di versione di R *non* vengono attualmente ignorate durante la lettura di un file `.pluginmap`. L'utente può comunque richiamare l'estensione e non vedrà alcun avviso immediato (nelle versioni future, probabilmente sarà visualizzato un messaggio di avviso).
- Di conseguenza, *non* è anche possibile definire versioni alternative di un'estensione a seconda della versione in esecuzione di R.
- Tuttavia, spesso è facile ottenere la retro-compatibilità come mostrato di seguito. Se si è a conoscenza dei problemi di compatibilità di R, considerare l'utilizzo di questo metodo, invece di definire una dipendenza da una particolare versione di R.

In molti casi, è facilmente possibile fornire funzionalità ridotte, se una determinata funzionalità non è disponibile nella versione in esecuzione di R. Considera il seguente breve esempio del file di un'estensione `.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="Metodo di compressione">
    <option label="Nessuno" value="">
    <option label="gzip" value="gzip">
    <option id="xz" label="xz" value="xz">
  </radio>
  [...]
</dialog>
```

In questo esempio, l'opzione di compressione «xz» sarà semplicemente disabilitata quando la versione runtime di R è precedente alla 2.10.0 (che non supportava la compressione xz). L'elemento `<dependency_check>` supporta gli stessi attributi dell'elemento `<dependencies>` nei file `.pluginmap`. Crea una proprietà booleana, che è `true` se le dipendenze specificate sono soddisfatte, `false` altrimenti.

11.3 Dipendenze dei pacchetti di R

È possibile definire dipendenze da pacchetti di R specifici, ma a partire da RKWard 0.6.1, queste dipendenze non vengono né verificate, né installate/caricate automaticamente. Sono tuttavia mostrate nei file di aiuto dell'estensione. Ecco un esempio di definizione:

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

NOTA

Assicurarsi sempre di aggiungere le chiamate `require()` appropriate, se la propria estensione richiede il caricamento di determinati pacchetti.

NOTA

Se [distribuite il vostro .pluginmap come R pacchetto](#), e tutte le estensioni dipendono da un particolare pacchetto, allora dovrete definire quella dipendenza dal livello del pacchetto di R. Definire le dipendenze dai pacchetti di R a livello delle `.pluginmap` di RKWard è molto utile, se solo alcune delle vostre estensioni necessitano della dipendenza, la dipendenza non è disponibile da CRAN o il vostro `.pluginmap` non è distribuito come pacchetto di R.

11.4 Dipendenze da altre .pluginmap di RKWard

Se le proprie estensioni dipendono da estensioni definite in un'altra `.pluginmap` (che *non* fa parte del pacchetto), è possibile definire la dipendenza in questo modo:

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

Al momento non sarà né caricato, né installato, e nemmeno avvisato della mancanza di `.pluginmap`, ma almeno le informazioni sulle dipendenze (e dove trovarle) saranno mostrate nella pagina di aiuto dell'estensione. Non dovete (e non dovrete) dichiarare le dipendenze dai `.pluginmap` forniti con la versione ufficiale di RKWard distribuzione, o sui `.pluginmap` che si trovano all'interno del vostro pacchetto. Inoltre, se è richiesto un `.pluginmap` [è distribuito come R pacchetto](#), dichiarate una dipendenza del pacchetto (come mostrato nella sezione precedente), anziché sulla mappa.

Per assicurarti che le `.pluginmap` richieste siano effettivamente caricati, usa il tag `<require>` (fai riferimento al [riferimento](#) per i dettagli).

11.5 Un esempio

Per chiarire come sia possibile mescolare le definizioni di dipendenza, ecco un esempio combinato:

```
<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>
    ...
  </components ...>
x
</document>
```

Capitolo 12

Traduzione delle estensioni

Finora abbiamo usato di sfuggita alcuni concetti riguardanti le traduzioni o «i18n» (abbreviazione di «internationalization», che ha 18 caratteri tra i e n). In questo capitolo daremo un resoconto più approfondito di ciò che i18n funzionalmente rappresenta per le estensioni di RKWard. Nella maggior parte dei casi *non* avrete bisogno di tutto questo nelle vostre estensioni. Tuttavia, potrebbe essere una buona idea leggere questo capitolo per intero, poiché comprendere questi concetti dovrebbe aiutarvi a creare estensioni che siano completamente traducibili e che consentano traduzioni di alta qualità.

12.1 Considerazioni generali

Un punto importante da comprendere riguardo alle traduzioni di software, a differenza delle traduzioni di altri materiali testuali, è che i traduttori spesso avranno difficoltà a ottenere un quadro completo di *cosa* stanno traducendo. Le traduzioni del software si basano necessariamente su frammenti di testo piuttosto brevi: ogni etichetta che dai a un <opzione> in una <radio>, ogni stringa che contrassegni per la traduzione in una chiamata alla funzione `i18n()`, formerà una «unità di traduzione» separata. In sostanza, ciascuno di questi frammenti sarà presentato al traduttore isolatamente. Beh, non un isolamento completo, poiché cerchiamo di fornire al traduttore tutto il contesto significativo che può essere estratto, automaticamente. Ma in alcuni punti i traduttori avranno bisogno di contesto aggiuntivo per dare un senso a una stringa, soprattutto se le stringhe sono brevi.

12.2 i18n nei file xml di RKWard

Per i file XML di RKWard, i18n per lo più funzionerà. Se state scrivendo il vostro `.pluginmap` (ad es. per un'estensione esterna), dovrete specificare un `po_id` accanto all'`id` del `pluginmap`. Ciò definisce il «catalogo dei messaggi» da utilizzare. In generale questo dovrebbe essere impostato identico all'`id` del vostro `.pluginmap`, ma se fornite diversi `.pluginmap` correlati in un singolo pacchetto, probabilmente vorrete specificare un `po_id` comune nelle vostre mappe. Il `po_id` di un file `.pluginmap` è ereditato da tutte le estensioni dichiarate in esso, a meno che non si dichiari un diverso `po_id`.

Per le estensioni e le pagine di aiuto, non è necessario dire a RKWard quali stringhe devono essere tradotte, perché questo risulta generalmente dal loro utilizzo. Tuttavia, come spiegato sopra, dovrete tenere d'occhio le stringhe che potrebbero essere ambigue o richiedere spiegazioni per essere tradotte correttamente. Per le stringhe che potrebbero avere significati diversi, fornire un `i18n_context` come questo:

```
<checkbox id="scale" label="Scala" i18n_context="Mostra la scala"/>
<checkbox id="scale" label="Scala" i18n_context="Ridimensiona il grafico"/>
```

Fornire *i18n_context* farà sì che le due stringhe vengano tradotte separatamente. Altrimenti condividerebbero un'unica traduzione. Inoltre, il contesto viene mostrato al traduttore. L'attributo *i18n_context* è supportato su tutti gli elementi che possono avere stringhe traducibili, da qualche parte, inclusi gli elementi che contengono testo al loro interno (elementi ad es. `<text>`).

In altri casi la stringa da tradurre ha un singolo significato non ambiguo, ma può ancora aver bisogno di spiegazioni. In questo caso puoi aggiungere un commento che sarà mostrato ai traduttori. Gli esempi potrebbero includere:

```
<!-- i18n: No, questo non è un errore di battitura per il grafico dello schermo! -->
<component id="scree_plot" label="Scree plot"/>

<!-- i18n: Se potete, scrivete questa stringa breve. Avere più di 15 caratteri
sembra davvero brutto a questo punto, e il significato dovrebbe essere per lo più evidente per
l'utente (selezione da un elenco di valori mostrato accanto a questo elemento) -->
<valueslot id="selected" label="Scegli uno"/>
```

Notare che tali commenti devono precedere l'elemento a cui si applicano e devono iniziare con «i18n:» o «TRANSLATORS:».

Infine, in rari casi, potresti voler escludere determinate stringhe dalla traduzione. Questo potrebbe avere senso, ad esempio, se offri la possibilità di scegliere tra diversi nomi di funzioni di R in un controllo `<radio>`. In questo caso, non vuoi che queste vengano tradotte (ma, a seconda del contesto, dovresti valutare l'idea di assegnare un'etichetta descrittiva):

```
<radio id="transformation" label="Funzione R da applicare">
  <option id="as.list" noi18n_label="as.list()" />
  <option id="as.vector" noi18n_label="as.vector()" />
  [...]
</radio>
```

Notare che si ometterà l'attributo *etichetta*, quindi, e si specificherà invece *noi18n_label*. Inoltre, notare che contrariamente a *i18n_context* e ai commenti, l'uso di *noi18n_label* renderà la propria estensione incompatibile con le versioni di RKWard prima della 0.6.3.

12.3 i18n nei file js e nelle sezioni di RKWard.

Contrariamente ai file `.xml`, rendere traducibili i file `.js` di un'estensione richiede più lavoro personalizzato. La differenza chiave, qui, è che non esiste un modo automatico decente per dire se una stringa deve essere visualizzata come una stringa facilmente leggibile o come un pezzo di codice. Quindi dovreste essere voi a contrassegnarlo. Abbiamo già mostrato esempi del genere, da sempre. Ecco una descrizione più completa delle funzioni i18n disponibili nel codice js e alcuni suggerimenti per casi più complessi:

i18n (msgid, [...])

La funzione più importante. Contrassegna la stringa per la traduzione. La stringa (tradotta o meno) viene restituita tra virgolette (""). Nella stringa è possibile utilizzare un numero arbitrario di segnaposto come mostrato di seguito. L'uso di tali segnaposto anziché concatenare piccole sotto-stringhe è molto più semplice per i traduttori:

```
i18n ("Confronta gli oggetti %1 e %2", getString ('x'), getString ('y')) ←
);
```

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

Come **i18n()**, ma fornisce in aggiunta un messaggio di contesto:

```
i18nc ("nome proprio, non stato mentale", "test di Mood");
```

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

Come **i18n()**, ma per i messaggi che possono essere diversi in forma singolare o plurale (e alcune lingue hanno forme più numeriche differenti). Notare che proprio come con **i18n()**, è possibile utilizzare un numero arbitrario di sostituzioni, ma è necessario il primo ("%1") e deve essere un numero intero.

```
i18np ("Confronto di una singola coppia", "Confronto di %1 coppie ←  
distinte", n_pairs);
```

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

i18np() con messaggio di contesto aggiunto.

comment (commento, [indentazione])

Visualizza un commento del codice, contrassegnato per la traduzione. Contrariamente alle altre funzioni **i18n()**, questo non è citato, ma un '#' viene aggiunto a ciascuna riga del commento.

```
comment ("Trasponi la matrice");  
      echo ('x <- t (x)\n');
```

Per aggiungere commenti ai traduttori (vedere [sopra](#) per una discussione sulle differenze tra commento e contesto), aggiungere un commento che inizia con «i18n:» o «translators:» direttamente sopra la chiamata **i18n()** per commentare. Ad esempio:

```
// i18n: l'ortografia è corretta: Scree plot.  
      echo ('rk.header (' + i18n ("Scree plot") + ')\n');
```

12.3.1 i18n e virgolette

Nella maggior parte dei casi non dovreste preoccuparvi del comportamento di **i18n()** rispetto alle virgolette. Dato che, in genere, le stringhe traducibili sono stringhe letterali, citarle è proprio la cosa giusta da fare e vi fa risparmiare un po' di digitazione. Inoltre, in funzioni come **make-HeaderCode()/HeaderCode()** che solitamente citano i loro argomenti, le stringhe **i18n()** sono protette dalle virgolette duplicate. Essenzialmente, funziona, inviando la stringa tradotta prima tramite **quote()** (per renderla tra virgolette), quindi tramite **noquote()** (per proteggerla da ulteriori virgolette). Se avete bisogno di una stringa traducibile che non sia quotata, usare **i18n(noquote("Il mio messaggio"))**. Se avete bisogno che una stringa traducibile venga citata, una seconda volta, inviatela tramite **quote()**, due volte.

Detto questo, in genere non è una buona idea rendere traducibili oggetti come nomi di funzioni o nomi di variabili. Per prima cosa, il linguaggio di programmazione è intrinsecamente in inglese e non c'è internazionalizzazione del linguaggio stesso. I commenti in codice sono una cosa diversa, ma si dovrebbe utilizzare la funzione **comment()** per quelli. In secondo luogo, rendere traducibili parti sintatticamente rilevanti del codice generato significa che le traduzioni potrebbero effettivamente creare problemi all'estensione. Ad esempio, se un traduttore ignaro traduce una stringa intesa come nome variabile in due parole distinte con uno spazio in mezzo.

12.4 Manutenzione delle traduzioni

Ora che l'estensione è stata resa traducibile, come è possibile tradurla effettivamente? In generale, è necessario preoccuparsi solo di questo quando si sviluppa un'estensione esterna. Per le estensioni nel deposito principale di RKWard, tutto è già pronto. Ecco il flusso di lavoro di base. Notare che è necessario che gli strumenti «gettext» siano installati:

- Contrassegna tutte le stringhe, fornendo contesto e commenti in base alle esigenze
- Eseguire `python3 scripts/update_plugin_messages.py --extract-only /percorso/a/mia.pluginmap`. `scripts/update_plugin_messages.py` non fa attualmente parte dei sorgenti rilasciati, ma può essere trovato in un checkout del deposito dei sorgenti.
- Distribuire il file `rkward__POID.pot` risultante ai propri traduttori. Per le estensioni esterne, si consiglia di inserirlo in una sottocartella «po» in `inst/rkward`.
- Il traduttore apre il file in uno strumento di traduzione come `lokalize`. In realtà, anche se non si ha intenzione di preparare alcuna traduzione, si dovrebbe provare questo passaggio per se stessi. Sfogliare le stringhe estratte alla ricerca di problemi / ambiguità.
- Il traduttore salva la traduzione come `rkward__POID.xx.po` (dove `xx` è il codice della lingua) e te lo restituisce.
- Copia `rkward__POID.xx.po` nelle tue fonti, accanto a `rkward__POID.pot`. Esegui `python3 scripts/update_plugin_messages.py /path/to/my.pluginmap` (Nota: senza `--extract-only`, questa volta). Questo unirà la traduzione con eventuali modifiche provvisorie alla stringa, compilerà la traduzione e la installerà in `DIR_OF_PLUGINMAP/po/x/LC_MESSAGES/rkward__POID.mo` (dove `xx` è ancora il codice della lingua).
- Si dovrebbe includere anche la traduzione non compilata (cioè `rkward__POID.xx.po`) nella propria distribuzione, nella sottocartella «po».
- Per qualsiasi aggiornamento della propria estensione, eseguire `python3 scripts/update_plugin_messages.py /percorso/a/mia.pluginmap` per aggiornare il file `.pot`, ma anche i file `.po` esistenti e i cataloghi dei messaggi compilati.

12.5 Tradurre le estensioni

Partiamo dal presupposto che si conosca il proprio lavoro di traduttore o che si sia disposti a documentarsi altrove. Tuttavia, vogliamo spendere alcune parole in particolare sulle traduzioni delle estensioni di RKWard:

- Le estensioni di RKWard non erano traducibili fino alla versione 0.6.3 e, per lo più, non erano scritte pensando all'internazionalizzazione, prima di allora. Quindi incontrerete stringhe piuttosto ambigue e altri problemi di internazionalizzazione che in altri progetti più maturi. Non aggirare questi problemi in silenzio, ma segnalaci tali problemi (o contatta il responsabile dell'estensione), in modo da poterli risolvere.
- Molte estensioni di RKWard si riferiscono a termini altamente specializzati, dalla gestione dei dati e dalle statistiche, ma anche da altri campi della scienza. In molti casi, una buona traduzione richiederà almeno una conoscenza di base di questi campi. In alcuni casi, *non esiste* una buona traduzione per un termine tecnico e l'opzione migliore potrebbe essere quella di lasciare il termine non tradotto o di includere il termine inglese tra parentesi. Non concentrarti troppo sul punteggio del 100% delle stringhe tradotte, concentrati sul fornire una buona traduzione, anche se ciò significa saltare alcune stringhe (o addirittura saltare alcuni cataloghi di messaggi nel loro complesso). Altri utenti potrebbero essere in grado di colmare eventuali lacune in termini tecnici.

Capitolo 13

Informazioni su autore, licenza e versione

Avete scritto una serie di estensioni e vi state preparando a [condividere il vostro lavoro](#). Per assicurarvi che gli utenti sappiano di cosa tratta, in quali termini possono usarlo e distribuirlo e chi dovrebbero contattare per problemi o idee, dovrete aggiungere alcune *informazioni* sulle vostre estensioni. Questo può essere fatto utilizzando l'elemento `<about>`. Può essere utilizzato sia nella `.pluginmap` o in singoli file `.xml` delle estensioni (in entrambi i casi come figlio diretto del tag `document`). Quando specificato nella `.pluginmap` si applicherà a tutte le estensioni. Se `<about>` è specificato in entrambi i posti, le informazioni `<about>` nel file `.xml` dell'estensione sovrascriveranno quelle nel file `.pluginmap`. Potete anche aggiungere un elemento `<about>` alle pagine `.rkh`, che non sono collegate a un'estensione, se ce n'è bisogno.

Ecco un file `.pluginmap` di esempio con solo alcune spiegazioni, di seguito. In caso di dubbio, ulteriori informazioni potrebbero essere disponibili dalla guida.

```
<document
  namespace="rkward"
  id="QuadraturadelCerchio_rkward"
>
  <about
    name="Quadratura del cerchio"
    shortinfo="Trova la quadratura del cerchio utilizzando la ←
      compensazione di Heisenberg."
    version="0.1-3"
    releasedate="2011-09-19"
    url="http://eternalwondermaths.example.org/23/stc.html"
    license="GPL"
    category="Geometria"
  >
    <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"
    />
```

Introduzione alla scrittura delle estensioni per Rkward

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

La maggior parte di questo dovrebbe spiegarsi da sola, quindi non discuteremo ogni singolo elemento del tag. Ma diamo un'occhiata ad alcuni dettagli che probabilmente necessitano di qualche commento per una più facile comprensione.

L'elemento *category* in **<about>** può essere definito piuttosto liberamente, ma dovrebbe essere significativo, poiché si ritiene che venga utilizzato per ordinare le estensioni. Tutti gli altri attributi in questo tag di apertura sono obbligatori e devono essere riempiti con un contenuto ragionevole.

Deve essere fornito anche almeno un **<author>** con un indirizzo email valido e il ruolo 'aut' ('author'). Nel caso in cui l'estensione causi problemi o qualcuno voglia condividere la sua gratitudine con voi, dovrebbe essere facile contattare qualcuno coinvolto. Per ulteriori informazioni su altri ruoli validi, come 'ctb' per i contributori del codice o 'cre' per la manutenzione dei pacchetti, fare riferimento alla documentazione [documentazione di R su person\(\)](#).

NOTA

Ricorda che puoi usare **<include>** e/o **<insert>** per ripetere informazioni in più file .xml (ad esempio, informazioni su un autore che ha collaborato a diverse estensioni). [Ulteriori informazioni](#).

SUGGERIMENTO

Non è necessario scrivere questo codice XML a mano. Se si utilizza la funzione `rk.plugin.skeleton()` dal pacchetto `rkwarddev` e si forniscono tutte le informazioni necessarie tramite l'opzione `about`, sarà automaticamente creato un file `.pluginmap` con una sezione `<about>` funzionante.

Capitolo 14

Condividere il proprio lavoro con altri

14.1 Estensioni esterne

A partire dalla versione 0.5.5, RKWard fornisce un modo comodo per installare estensioni aggiuntive di terze parti che non appartengono al pacchetto principale. Le chiamiamo ‘estensioni esterne’. Vengono rilasciate nella forma di pacchetti di R e possono essere gestite direttamente tramite le normali funzionalità di gestione dei pacchetti di R e/o RKWard.

Questa sezione della documentazione descrive come devono essere creati i pacchetti delle estensioni esterne, in modo che RKWard possa utilizzarli. La creazione dell’estensione è ovviamente identica alle sezioni precedenti. Cioè, si dovrebbe possibilmente scrivere prima un’estensione funzionante, quindi tornare qui per sapere come fare distribuirla.

Poiché le estensioni esterne sono una funzionalità relativamente giovane, questi dettagli potrebbero probabilmente cambiare nelle versioni future. Chiunque è benvenuto per contribuire con le proprie idee per migliorare il processo.

SUGGERIMENTO

Questi documenti spiegano i dettagli delle estensioni esterne in modo da poter imparare come funzionano. Inoltre, dare un’occhiata al [pacchetto rkwarddev](https://github.com/rkwarddev/rkwarddev), progettato per automatizzare gran parte del processo di scrittura.

14.2 Perché le estensioni esterne?

Il numero di pacchetti per estendere la funzionalità di R è già enorme e in crescita. Da un lato, desideriamo incoraggiare a scrivere estensioni anche per le attività più specializzate di cui si ha bisogno. D’altra parte, l’utente medio non dovrebbe perdersi in enormi alberi di menu pieni di termini statistici sconosciuti. Pertanto è sembrato ragionevole mantenere la gestione delle estensioni di RKWard abbastanza modulare. La squadra di RKWard mantiene il proprio deposito pubblico di pacchetti su <https://files.kde.org/rkward/R/>, pensato per ospitare le vostre estensioni esterne.

Come regola generale, le estensioni che sembrano servire a un ampio utilizzo (ad es. Test t) dovrebbero diventare parte del pacchetto principale, mentre quelle che servono un gruppo piuttosto limitato di persone con interessi specifici dovrebbero essere fornite come pacchetto opzionale. Per gli autori di estensioni, è consigliabile iniziare con un’estensione esterna.

14.3 Struttura del pacchetto di un'estensione

Affinché le estensioni esterne siano installate e funzionino correttamente, devono seguire alcune linee guida strutturali relative alla loro gerarchia dei file.

14.3.1 Gerarchia dei file

Diamo un'occhiata al prototipo di gerarchia dei file di un complesso archivio di estensione. Non è necessario includere tutte queste cartelle e/o file per far funzionare un'estensione (continuare a leggere per sapere cosa è assolutamente necessario), considerare questo un esempio di 'best practice':

```
nome_estensione/
    inst/
        rkward/
            plugins/
                nome_estensione.xml
                nome_estensione.js
                nome_estensione.rkh
                ...
            po/
                ll/
                    LC_MESSAGES/
                        rkward__nome_estensione_rkward.mo
                        rkward__nome_estensione_rkward.ll.po
                        rkward__nome_estensione_rkward.pot
            tests/
                testsuite_name/
                    RKTestStandards.nome_test.R
                    rkcommands.R
                    RKTestStandards.nome_test.Rkout
                    ...
                pacchetto_test.R
            nome_estensione.pluginmap
            ...

    ChangeLog
    README
    AUTHORS
    LICENSE
    DESCRIPTION
```

NOTA

In questo esempio, tutte le occorrenze di `plugin_name`, `testsuite_name` e `sometest_name` devono essere sostituite con i loro nomi corretti, di conseguenza. Inoltre, `ll` è un segnaposto per un'abbreviazione linguistica (ad es., 'de', 'en' o 'es').

SUGGERIMENTO

Non è necessario creare questa gerarchia di file a mano. Se si utilizza la funzione `rk.plugin.skeleton()` dal pacchetto `rkwarddev`, creerà automaticamente per voi tutti i file e le cartelle necessari, tranne la cartella `po` creata e gestita dallo [script di traduzione](#).

14.3.1.1 Componenti di base di un'estensione

È obbligatorio includere almeno tre file: un `.pluginmap`, una descrizione `.xml` e un file `.js` dell'estensione. Cioè, anche la cartella «plugins» è facoltativa. Potrebbe essere utile dare un po' di ordine ai propri file, soprattutto se si includono più di un'estensione/finestra nell'archivio, il che ovviamente non è un problema. Per i file dell'estensione è possibile avere tutte le cartelle che si ritengono opportune, devono solo assomigliare rispettivamente a `.pluginmap`. È anche possibile includere diversi file `.pluginmap`, se soddisfano le vostre esigenze, ma dovrete includerli tutti in `'plugin_name.pluginmap'`.

Ogni pacchetto di R deve avere un file `DESCRIPTION` valido, anch'esso fondamentale per RKWard per riconoscerlo come fornitore di estensioni. La maggior parte delle informazioni che contiene sono necessarie anche nelle [Meta-informazioni](#) dell'estensione e possibilmente nelle [dipendenze](#), ma in un formato diverso (la documentazione di R spiega [il file DESCRIPTION in dettaglio](#)).

Oltre ai contenuti generali di un file `DESCRIPTION`, assicurarsi di includere anche la riga `'Enhances:rkward'`. Ciò avvierà una scansione automatica da parte di RKWard del pacchetto alla ricerca di estensioni, se è installato. Un file `DESCRIPTION` di esempio è simile al seguente:

```
Package: QuadraturadelCerchio
Type: Package
Title: Squadra il cerchio
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: Trova la quadratura del cerchio utilizzando la ←
             compensazione di Heisenberg.
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"))
```

SUGGERIMENTO

Non è necessario scrivere questo file a mano. Se si utilizza la funzione `rk.plugin.skeleton()` dal pacchetto [rkwarddev](#) e si forniscono tutte le informazioni necessarie tramite l'opzione `'about'`, creerà automaticamente un file `DESCRIPTION` funzionante per te.

14.3.1.2 Informazioni aggiuntive (facoltativo)

`ChangeLog`, `README`, `AUTHORS`, `LICENSE` dovrebbero essere autoesplicativi e sono del tutto facoltativi. In realtà, non saranno interpretati da RKWard, quindi sono piuttosto destinati a contenere informazioni aggiuntive che potrebbero essere rilevanti ad es. per i distributori. La maggior parte dei loro contenuti rilevanti (riconoscimenti dell'autore, termini di licenza ecc;) saranno comunque inclusi nei file effettivi dell'estensione (vedere la [sezione sulle meta-informazioni](#)). Tenere presente che tutti questi file potrebbero anche essere posizionati da qualche parte nella cartella «inst», se volete che siano presenti non solo nell'archivio sorgente, ma anche nel pacchetto installato.

14.3.1.3 Test automatico dell'estensione (facoltativo)

Un'altra cartella facoltativa è «tests», che ha lo scopo di fornire i file necessari per [test automatizzati delle estensioni](#). Questi test sono utili per verificare rapidamente se le proprie estensioni funzionano ancora con le nuove versioni di R o RKWard. Se desiderate includere dei test, dovrete davvero limitarvi allo schema di denominazione e alla gerarchia mostrati qui. Cioè, i test dovrebbero risiedere in una cartella chiamata `tests`, che include un file `testsuite.R` e una cartella con gli standard dei test che prendono il nome dalla suite di test appropriata. Potete, tuttavia, fornire più di una suite di test; in tal caso, se non volete aggiungerli tutti in un unico file `testsuite.R`, potete dividerli in ad es. un file per ogni suite di test e creare un file `testsuite.R` con chiamate `source()` per ogni file di suite. In entrambi i casi, creare sottocartelle separate con standard di test per ciascuna suite definita.

I vantaggi di sostenere questa struttura sono che i test delle estensioni possono essere eseguiti semplicemente invocando `rktests.makplugintests()` dal pacchetto [rkwardtests](#) senza argomenti aggiuntivi. Dare un'occhiata alla documentazione in linea su [Test automatico delle estensioni](#) per ulteriori dettagli.

14.4 Creazione del pacchetto dell'estensione

Come spiegato in precedenza, le estensioni di RKWard sono in effetti pacchetti di R, e quindi il processo di pacchettizzazione è identico. A differenza dei pacchetti di R, il pacchetto di un'estensione non contiene ulteriore codice R (sebbene sia possibile ovviamente aggiungere estensioni di RKWard anche ai soliti pacchetti di R, usando gli stessi metodi spiegati qui). Ciò dovrebbe rendere ancora più semplice la creazione di un pacchetto funzionante, purché sia presente un file `DESCRIPTION` valido e rispetti la gerarchia dei file spiegata nelle [sezioni precedenti](#).

Il modo più semplice per compilare e provare la propria estensione è utilizzare il comando R nella riga di comando, ad esempio:

```
R CMD build SquaretheCircle
```

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

SUGGERIMENTO

Non è necessario compilare il pacchetto in questo modo sulla riga di comando. Se si utilizza la funzione `rk.build.package()` dal [pacchetto rkwarddev](#), costruirà e/o controllerà il pacchetto dell'estensione al posto vostro.

Capitolo 15

Sviluppo di estensioni con il pacchetto `rkwarddev`

15.1 Panoramica

La scrittura di estensioni esterne implica la scrittura di file in tre linguaggi (XML, JavaScript e R) e la creazione di una gerarchia standardizzata di cartelle. Per rendere tutto questo molto più semplice per gli sviluppatori di estensioni disponibili, stiamo fornendo il pacchetto `rkwarddev`. Fornisce una serie di semplici comandi R funzioni per creare il file XML codice per tutti gli elementi della finestra come schede, caselle di controllo, elenchi a tendina o browser di file, nonché funzioni per creare JavaScript codice e RKWard file di aiuto con cui iniziare. La funzione `rk.plugin.skeleton()` crea l'albero delle cartelle previsto e tutti i file necessari dove dovrebbero essere.

Questo pacchetto non è installato per impostazione predefinita, ma deve essere installato manualmente dal proprio deposito [RKWard](#). È possibile farlo utilizzando la GUI (**Impostazioni** → **Configura pacchetti**), o da qualsiasi sessione di R in esecuzione:

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

`rkwarddev` dipende da un altro piccolo pacchetto chiamato 'XiMpLe', che è un analizzatore e generatore XML molto semplice e presente nello stesso deposito.

L'intera [documentazione in formato PDF](#) può anche essere trovata qui. Un'introduzione più dettagliata all'utilizzo del pacchetto è disponibile nella [rkwarddev vignette](#).

15.2 Esempio pratico

Per farsi un'idea di come 'scrivere un'estensione', rispetto all'approccio diretto visto nei capitoli precedenti, creeremo ancora una volta l'estensione del test t completo - questa volta solo con le funzioni di R del pacchetto `rkwarddev`.

SUGGERIMENTO

Il pacchetto aggiungerà una nuova finestra della GUI per RKWard sotto **File** → **Esporta** → **Crea RK-Ward script dell'estensione**. Come suggerisce il nome, è possibile creare scheletri di estensioni per ulteriori modifiche. Questa finestra stessa è stata a sua volta generata da uno script `rkwarddev` che potete trovare nella cartella 'demo' del pacchetto installato e nei sorgenti del pacchetto, come ulteriore esempio. Potete anche eseguirlo invocando `demo("skeleton_dialog")`

15.2.1 Descrizione della GUI

Noterete immediatamente che il flusso di lavoro è notevolmente diverso: contrariamente alla scrittura diretta del codice XML, non inizia con la definizione di `<document>`, ma direttamente con gli elementi dell'estensione che vorreste avere nella finestra. Potete assegnare ciascun elemento dell'interfaccia (che si tratti di caselle di selezione, menu a tendina, slot variabili o qualsiasi altra cosa) a singoli oggetti di R, e poi combinare questi oggetti nella GUI effettiva. Il pacchetto ha funzioni per **ogni tag XML** che può essere utilizzato per definire la GUI dell'estensione e la maggior parte di essi ha anche lo stesso nome, solo con il prefisso `rk.XML.*`. Ad esempio, la definizione di un elemento `<varselector>` e di due elementi `<varslot>` per le variabili `"x"` e `"y"` dell'esempio di t-test può essere eseguita nel modo seguente:

```
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")
```

Il dettaglio più interessante sono probabilmente le `source=variables`: una caratteristica importante del pacchetto è che tutte le funzioni possono generare ID automatici, quindi non dovete preoccuparvi di pensare ai valori `id` o di ricordarli per fare riferimento a uno specifico elemento dell'estensione. Potete semplicemente dare gli oggetti di R come riferimento, poiché tutte le funzioni che necessitano di un ID da qualche altro elemento possono leggerlo anche da questi oggetti. `rk.XML.varselector()` è un po' speciale, poiché di solito non ha un contenuto specifico da cui creare un ID (può, ma solo se specificate un'etichetta), quindi dobbiamo impostare un nome ID. Ma `rk.XML.varslot()` non avrebbe bisogno degli argomenti `id.name` qui, quindi sarebbe sufficiente:

```
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 ricreare il codice di esempio sul punto, è necessario impostare manualmente tutti i valori di ID. Ma dal momento che il pacchetto renderà le nostre vite più facili, d'ora in poi non ce ne occuperemo più.

SUGGERIMENTO

`rkwarddev` è capace di molta automazione per aiutarvi a costruire le proprie estensioni. Tuttavia, potrebbe essere preferibile non utilizzarlo fino in fondo. Se l'obiettivo è produrre un codice che non solo funzioni, ma che possa essere facilmente letto e confrontato con lo script del generatore da un essere umano, si dovrebbe considerare di impostare sempre ID utili con `id.name`. Assegnare un nome ai propri oggetti R identici a questi ID aiuteranno anche a ottenere il codice dello script che è facile da comprendere.

Se si vuole vedere come appare il codice XML dell'elemento definito se lo si è esportato in un file, è possibile semplicemente chiamare l'oggetto con il suo nome. Quindi, se si è invocata `'var.x'` nella propria sessione di R, si dovrebbe vedere qualcosa del genere:

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

Alcuni tag sono utili solo nel contesto di altri. Pertanto, ad esempio, non troverete una funzione per il tag `<opzione>`. Invece, sia i pulsanti di opzione che gli elenchi a tendina sono definiti includendo le loro opzioni come un elenco denominato, dove i nomi rappresentano le etichette da mostrare nella finestra e il loro valore è un vettore denominato che può avere due voci, `val` per il valore di un'opzione e il booleano `chk` per specificare se questa opzione è selezionata per impostazione predefinita.

Introduzione alla scrittura delle estensioni per RKWard

```
test.hypothesis <- rk.XML.radio("ipotesi del test in uso",
  options=list(
    "Due lati"=c(val="due.lati"),
    "Il primo è più grande"=c(val="maggiore"),
    "Il secondo è più grande"=c(val="minore")
  )
)
```

Il risultato è simile al seguente:

```
<radio id="rad_usngtsth" label="ipotesi di test in uso">
  <option label="Due lati" value="due.lati" />
  <option label="Il primo è più grande" value="maggiore" />
  <option label="Il secondo è più grande" value="minore" />
</radio>
```

Tutto ciò che manca agli elementi della scheda 'Impostazioni di base' è la casella di controllo per i campioni accoppiati e la strutturazione di tutti questi elementi in righe e colonne:

```
check.paired <- rk.XML.cbox("Campione accoppiato", 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()` è un'eccezione rara in cui il nome della funzione non contiene il nome completo del tag, per risparmiare battiture per questo elemento spesso utilizzato. Questo è ciò che `basic.settings` ora contiene:

```
<row id="row_vTFSP10TF">
  <varselector id="vars" />
  <column id="clm_vrsTFSP10">
    <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="ipotesi di test in uso">
      <option label="Due lati" value="due.lati" />
      <option label="Il primo è più grande" value=" ←
        maggiore" />
      <option label="Il secondo è più grande" value=" ←
        minore" />
    </radio>
    <checkbox id="chc_Pardsmpl" label="Campione accoppiato" ←
      value="1" value_unchecked="0" />
  </column>
</row>
```

In modo simile, le righe successive creeranno oggetti di R per gli elementi della scheda 'Opzioni', introducendo le funzioni per caselle numeriche, cornici e allungamento:

```
check.eqvar <- rk.XML.cbox("assumi varianze uguali equal variances", value ←
  "1", un.value="0")
conf.level <- rk.XML.spinbox("livello di confidenza", min=0, max=1, initial ←
  =0.95)
check.conf <- rk.XML.cbox("stampa intervallo di confidenza", val="1", chk= ←
  TRUE)
conf.frame <- rk.XML.frame(conf.level, check.conf, rk.XML.stretch(), label ←
  ="Intervallo di confidenza")
```

Introduzione alla scrittura delle estensioni per RKWard

Ora tutto ciò che dobbiamo fare è riunire gli oggetti in un tabbook e posizionarli in una sezione di dialogo:

```
full.dialog <- rk.XML.dialog(  
  label="Test t a due variabili",  
  rk.XML.tabbook(tabs=list("Basic settings"=basic.settings, "Options" ←  
    "=list(check.eqvar, conf.frame))")  
)
```

Possiamo inoltre creare la sezione della procedura guidata con le sue due pagine utilizzando gli stessi oggetti, quindi i loro ID saranno estratti per i tag **<copy>**:

```
full.wizard <- rk.XML.wizard(  
  label="Test t a due variabili",  
  rk.XML.page(  
    rk.XML.text("Come primo passo, seleziona le due ←  
      variabili che desideri confrontare.  
      E specifica quale ipotizzi sia la più ←  
      grande. Seleziona due lati,  
      se la tua teoria non indica quale variabile ←  
      sia più grande."),  
    rk.XML.copy(basic.settings)),  
  rk.XML.page(  
    rk.XML.text("Di seguito, sono disponibili alcune ←  
      opzioni avanzate. È generalmente sicuro non ←  
      assumere che  
      le variabili abbiano varianze uguali. Una ←  
      correzione appropriata sarà quindi ←  
      applicata.  
      La scelta di \"assumi varianze uguali\" ←  
      potrebbe aumentare la potenza del test, ←  
      tuttavia."),  
    rk.XML.copy(check.eqvar),  
    rk.XML.text("A volte è utile ottenere una stima ←  
      dell'intervallo di confidenza della  
      differenza tra le medie. Sotto, puoi ←  
      specificare se una dovrebbe essere ←  
      mostrata, e  
      quale livello di confidenza dovrebbe essere ←  
      applicato (95% corrisponde a un livello ←  
      di significatività  
      del 5%)."),  
    rk.XML.copy(conf.frame))
```

Questo è tutto per la GUI. Il documento globale sarà combinato alla fine da `rk.plugin.skeleton()`.

15.2.2 Codice JavaScript

Fino ad ora, l'utilizzo del pacchetto `rkwarddev` potrebbe non aver essere stato di aiuto. Questo cambierà proprio ora.

Prima di tutto, proprio come non dovevamo preoccuparci degli ID per gli elementi quando si definisce la struttura della GUI, non dobbiamo preoccuparci dei nomi delle variabili JavaScript nel passaggio successivo. Se si vuole maggior controllo, è possibile scrivere codice JavaScript e incollarlo nel file generato. Ma è probabilmente molto più efficiente per farlo alla maniera di `rkwarddev`.

In particolare non dovete definire alcuna variabile tu stesso, dato che `rk.plugin.skeleton()` può analizzare il vostro codice XML e definire automaticamente tutte le variabili di cui probabilmente avrete bisogno: ad esempio, non vi preoccuperete di includere una casella di selezione se non ne utilizzate il valore o lo stato in seguito. Quindi possiamo iniziare a scrivere immediatamente l'effettivo codice R che genera JS.

SUGGERIMENTO

La funzione `rk.JS.scan()` può anche eseguire la scansione dei file XML alla ricerca delle variabili.

Il pacchetto ha alcune funzioni per i costrutti del codice JS comunemente usati nelle estensioni di RKWard, come la funzione `echo()` o le condizioni `if() {...} else {...}`. Ci sono alcune differenze tra JS e R, ad es., per `paste()` in R userete la virgola per concatenare stringhe di caratteri, mentre per `echo()` in JS userete '+' e le righe devono terminare con un punto e virgola. Utilizzando il pulsante R funzioni, potete quasi dimenticare queste differenze e continuare a scrivere codice R.

Queste funzioni possono accettare diverse classi di oggetti di input: testo semplice, oggetti di R con codice XML come sopra o, a sua volta, risultati di alcune altre funzioni JS del pacchetto. Alla fine, invocherete sempre `rk.paste.JS()`, che si comporta in modo simile a `paste()`, ma a seconda degli oggetti di input li sostituirà con i loro ID XML, nome della variabile JavaScript o anche blocchi di codice JavaScript completi.

Per l'esempio del test t, abbiamo bisogno di due oggetti JS: uno per calcolare i risultati e uno per stamparli nella funzione `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)
```

Come potete vedere, `rkwarddev` fornisce anche un implementazione di R della funzione `echo()`. Restituisce esattamente una stringa di caratteri con una versione JS valida di se stesso. Potreste anche notare che tutti gli oggetti di R qui sono quelli che abbiamo creato in precedenza. Saranno automaticamente sostituiti con i nomi delle variabili, quindi dovrebbe essere abbastanza intuitivo. Ogni volta che è necessaria solo questa sostituzione, è possibile utilizzare la funzione `id()`, che restituirà anche esattamente una stringa di caratteri da tutti gli oggetti che le sono stati forniti (si potrebbe dire che si comporta come `paste()` con una sostituzione di oggetto molto specifica).

La funzione `js()` è un wrapper che ti consente di utilizzare le condizioni `if(){...} else {...}` di R come sei abituato. Saranno tradotti direttamente nel codice JS. Conserva anche alcuni operatori come `<`, `>=`, `||`, quindi puoi confrontare logicamente i tuoi oggetti di R senza bisogno di virgolette per la maggior parte del tempo. Diamo un'occhiata all'oggetto 'JS.calc' risultante, che ora contiene una stringa di caratteri con questo contenuto:

Introduzione alla scrittura delle estensioni per RKWard

```
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

In alternativa per le condizioni `if()` nidificate in `js()`, è possibile utilizzare la funzione `ite()`, che si comporta in modo simile alla `ifelse()` di R. Tuttavia, le istruzioni condizionali costruite usando `ite()` di solito sono più difficili da leggere e dovrebbero essere sostituite da `js()` quando possibile.

15.2.3 Mappa delle estensioni

Questa sezione è molto breve: non abbiamo bisogno di scrivere alcuna `.pluginmap`, in quanto può essere generata automaticamente da `rk.plugin.skeleton()`. La gerarchia dei menu può essere specificata tramite l'opzione `pluginmap`:

```
[...]
  pluginmap=list(
    name="Test t a due variabili",
    hierarchy=list("analysis", "means", "t-Test"))
[...]
```

15.2.4 Pagina della guida

Anche questa sezione è molto breve: `rk.plugin.skeleton()` non può scrivere un'intera pagina della guida che ha. Ma può analizzare il documento XML anche per gli elementi che probabilmente meritano una voce nella pagina della guida e crea automaticamente un modello di pagina della guida per la nostra estensione. Tutto quello che dobbiamo fare in seguito è scrivere alcune righe per ciascuna sezione elencata.

SUGGERIMENTO

La funzione `rk.rkh.scan()` può anche eseguire la scansione dei file XML esistenti per creare uno scheletro di file della guida.

15.2.5 Genera i file delle estensioni

Ora arriva il passaggio finale, in cui consegneremo tutti gli oggetti generati a `rk.plugin.skeleton()`:

```
plugin.dir <- rk.plugin.skeleton("t-Test",
  xml=list(
    dialog=full.dialog,
```

Introduzione alla scrittura delle estensioni per RKWard

```
wizard=full.wizard),
js=list(
  results.header="Test t a due variabili",
  calculate=JS.calc,
  printout=JS.print),
pluginmap=list(
  name="Test t a due variabili",
  hierarchy=list("analysis", "means", "t-Test")),
load=TRUE,
edit=TRUE,
show=TRUE)
```

I file saranno creati in una cartella temporanea per impostazione predefinita. Le ultime tre opzioni non sono necessarie, ma molto utili: `load=TRUE` aggiungerà automaticamente la nuova estensione alla configurazione di RKWard (poiché è in una cartella temporanea e quindi cesserà di esistere alla chiusura di RKWard, sarà automaticamente rimossa da RKWard al suo prossimo avvio), `edit=TRUE` aprirà tutti i file creati per la modifica nelle schede dell'editor di RKWard e `show=TRUE` tenterà di avviare direttamente l'estensione, in modo da poter esaminare come appare senza un clic. Potreste prendere in considerazione l'aggiunta di `overwrite=TRUE` se state per eseguire lo script ripetutamente (ad es. dopo modifiche al codice), poiché per impostazione predefinita nessun file sarà sovrascritto.

L'oggetto risultato 'plugin.dir' contiene il percorso della cartella in cui è stata creata l'estensione. Questo può essere utile in combinazione con la funzione `rk.build.package()`, per creare un vero e proprio pacchetto di R per condividere la propria estensione con altri -- ad es. inviandola alla squadra di sviluppo di RKWard da aggiungere al nostro repository di estensioni.

15.2.6 Lo script completo

Per ricapitolare quanto sopra, ecco lo script completo per creare l'esempio di test t funzionante. In aggiunta al codice già spiegato, viene caricato anche il pacchetto, se necessario, e utilizza l'ambiente `local()`, quindi tutti gli oggetti creati non finiranno nel tuo attuale spazio di lavoro (eccetto '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("ipotesi del test in uso",
    options=list(
      "Due lati"=c(val="due.lati"),
      "Il primo è più grande"=c(val="maggiore"),
      "Il secondo è più grande"=c(val="minore")
    )
  )
  check.paired <- rk.XML.cbox("Campione accoppiato", 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("assumi varianza uguale", value="1", un. ←
    value="0")
  conf.level <- rk.XML.spinbox("livello di confidenza", min=0, max=1, ←
    initial=0.95)
```

Introduzione alla scrittura delle estensioni per RKWard

```
check.conf <- rk.XML.cbox("stampa intervallo di confidenza", val ←  
  ="1", chk=TRUE)  
conf.frame <- rk.XML.frame(conf.level, check.conf, rk.XML.stretch() ←  
  , label="Intervallo di confidenza")  
  
full.dialog <- rk.XML.dialog(  
  label="Two Variable t-Test",  
  rk.XML.tabbook(tabs=list("Impostazioni di base"=basic. ←  
    settings, "Options"=list(check.eqvar, conf.frame)))  
)  
  
full.wizard <- rk.XML.wizard(  
  label="Test t a due variabili",  
  rk.XML.page(  
    rk.XML.text("Come primo passo, seleziona le ←  
      due variabili che desideri confrontare.  
      E specifica, quale ipotizzi essere ←  
        la più grande. Seleziona due ←  
          lati,  
          se la tua teoria non lo indica, ←  
            quale variabile è più grande."),  
    rk.XML.copy(basic.settings)),  
  rk.XML.page(  
    rk.XML.text("Di seguito sono disponibili ←  
      alcune opzioni avanzate. È generalmente ←  
        sicuro non assumere che le  
        variabili abbiano varianze uguali. ←  
          Una correzione appropriata sarà ←  
            quindi applicata.  
            Tuttavia, la scelta di \"assumi ←  
              varianze uguali\" potrebbe ←  
                incrementare la potenza del test ←  
                  ."),  
    rk.XML.copy(check.eqvar),  
    rk.XML.text("A volte è utile ottenere una ←  
      stima dell'intervallo di confidenza ←  
        della  
        differenza tra le medie. Sotto, ←  
          puoi specificare se una dovrebbe ←  
            essere mostrata, e  
            quale livello di confidenza ←  
              dovrebbe essere applicata (95% ←  
                corrisponde a un livello di ←  
                  significatività  
                  del 5%)."),  
    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("Test t",
  xml=list(
    dialog=full.dialog,
    wizard=full.wizard),
  js=list(
    results.header="Test t a due variabili",
    calculate=JS.calc,
    printout=JS.print),
  pluginmap=list(
    name="Test t a due variabili",
    hierarchy=list("analysis", "means", "t-Test")),
  load=TRUE,
  edit=TRUE,
  show=TRUE,
  overwrite=TRUE)
))

```

15.3 Aggiunge una pagina della guida

Se si desidera scrivere una pagina di aiuto per la propria estensione, il modo più semplice per farlo è aggiungere le istruzioni particolari direttamente alle definizioni degli elementi XML alle quali appartengono:

```

variables <- rk.XML.varselector(
  id.name="vars",
  help="Seleziona gli oggetti di dati che desideri analizzare.",
  component="Data"
)

```

Il testo fornito al parametro *help* può quindi essere recuperato da `rk.rkh.scan()` e scritto nella pagina di aiuto di questo componente dell'estensione. Affinché ciò funzioni tecnicamente, tuttavia, `rk.rkh.scan()` deve sapere quali oggetti di R appartengono a un componente di estensione. Questo è il motivo per cui è necessario fornire anche il parametro *component* e assicurarsi che sia identico per tutti gli oggetti che in comune.

Poiché di solito combinerete molti oggetti in un'unica finestra e potreste voler riutilizzare oggetti come **<varslot>** in più componenti delle proprie estensioni, è possibile definire globalmente un componente con `rk.set.comp()`. Se impostato, si presuppone che tutti i seguenti oggetti utilizzati nello script appartengano a quel particolare componente, finché `rk.set.comp()` non viene richiamato nuovamente con un nome di componente diverso. È possibile quindi omettere il parametro *component*:

```

rk.set.comp("Data")
variables <- rk.XML.varselector(
  id.name="vars",
  help="Seleziona gli oggetti di dati che desideri analizzare."
)

```

Per aggiungere sezioni globali come `<summary>` o `<usage>` alla pagina della guida, utilizzare funzioni come `rk.rkh.summary()` o `rk.rkh.usage()` di conseguenza. I loro risultati vengono quindi utilizzati per impostare gli elementi dell'elenco come `summary` o `usage` nel parametro `rkh` di `rk.plugin.component()/rk.plugin.skeleton()`.

15.4 Traduzione delle estensioni

Il pacchetto `rkwarddev` è in grado di produrre estensioni esterne con pieno supporto `i18n`. Ad esempio, tutte le funzioni rilevanti che generano oggetti XML offrono un parametro facoltativo per specificare `i18n_context` o `no_i18n_label`:

```
varComment <- rk.XML.varselector(id.name="vars", i18n=list(comment=" ←  
  Selettore variabile principale"))  
varContext <- rk.XML.varselector(id.name="vars", i18n=list(context=" ←  
  Selettore variabile principale"))  
cboxNoi18n <- rk.XML.cbox(label="Potenza", id.name="power", i18n=FALSE)
```

Gli esempi sopra generano risultato in questo modo:

```
# varComment  
<!-- i18n: Selettore variabile principale -->  
  <varselector id="vars" />  
  
# varContext  
<varselector id="vars" i18n_context="Selettore variabile principale" />  
  
# cboxNoi18n  
<checkbox id="power" no_i18n_label="Potenza" value="true" />
```

C'è anche il supporto per il codice JS traducibile. In effetti, il pacchetto prova ad aggiungere chiamate `i18n()` in modo predefinito nei punti in cui di solito è utile. La funzione `rk.JS.header()` è un buon esempio:

```
jsHeader <- rk.JS.header("Risultati del test")
```

Questo produce il seguente codice JS:

```
new Header(i18n("Test dei risultati")).print();
```

Ma è possibile anche contrassegnare manualmente le stringhe nel proprio codice JS come traducibili, utilizzando la funzione `i18n()` proprio come si farebbe scrivendo direttamente il file JS.

Appendice A

Riferimento

A.1 Tipi di proprietà/Modificatori

In alcuni punti di questa introduzione abbiamo parlato delle ‘proprietà’ degli elementi della GUI o altro. In realtà ci sono diversi tipi di proprietà. Di solito non è necessario preoccuparsi di questo, poiché basta usare il buon senso per collegare qualsiasi proprietà a qualsiasi altra proprietà. Tuttavia, internamente, esistono diverse tipologie di proprietà. Ciò che conta è quando si recuperano alcuni valori speciali nel modello JS. Nelle istruzioni `getString("id")/getBoolean("id")/getList("id")` puoi anche specificare alcuni cosiddetti ‘modificatori’ come questo: `getString("id.modifier")`. Questo modificatore influenzerà il modo in cui sarà stampato il valore. Continuare a leggere per l’elenco delle proprietà e i modificatori che ciascuna di esse rende disponibili:

Proprietà delle stringhe

Il tipo più semplice di proprietà, utilizzato semplicemente per contenere un pezzo di testo. Modificatori:

Nessun modificatore ("")

La stringa come definita / impostata

quotata

La stringa tra virgolette (adatta per il passaggio a R come carattere).

Proprietà dei booleani

Proprietà che possono essere attivate o disattivate, vere o false. Ad esempio, le proprietà create dai tag `<convert>`, così come la proprietà che accompagna una `<checkbox>` (vedi sotto). I seguenti valori saranno restituiti in base al modificatore specificato:

Nessun modificatore ("")

In modo predefinito, la proprietà restituirà 1 se è vero e 0 in caso contrario. Il modo consigliato per recuperare i valori booleani è usare `getBoolean()`. Notare che per `getString()`, sarà restituita la stringa «0» quando la proprietà è falsa. Questa stringa restituirebbe vero, non falso in JS.

"labeled"

Restituisce la stringa «true» se vero, «false», se falso o qualora siano state specificate stringhe personalizzate (in genere in una `<checkbox>`).

"true"

Restituisce la stringa come se la proprietà fosse vera, anche se è falsa

"false"

Restituisce la stringa come se la proprietà fosse falsa, anche se è vera

"not"

Questa in realtà restituisce un'altra proprietà booleana, che è il contrario dell'attuale (cioè falso se vero, vero se falso)

"numeric"

Obsoleto, fornito per compatibilità con le versioni precedenti. Come nessun modificatore `" "`. Restituisce «1» se la proprietà è vera o «0» se è falsa.

Proprietà degli interi

Una proprietà pensata per contenere un valore intero (ma ovviamente restituisce comunque una stringa di caratteri numerici al modello JS). Non accetta alcun modificatore. Usato nelle `<spinbox>` (vedi sotto)

Proprietà dei numeri reali

Una proprietà pensata per contenere un valore numerico reale (ma ovviamente restituisce comunque una stringa di caratteri numerici al modello JS). Utilizzata da `<spinbox>` (vedi sotto)

Nessun modificatore ("")

Per `getValue()/getString()`, questo restituisce lo stesso di «formatted». Nelle versioni future, sarà invece possibile ottenere una rappresentazione numerica.

"formatted"

Restituisce il numero formattato (come stringa).

Proprietà RObject

Una proprietà ha progettato una selezione di uno o più oggetti di R. Utilizzato soprattutto in `varselector` e `varslot`. I seguenti valori saranno restituiti in base al modificatore specificato:

Nessun modificatore ("")

Per impostazione predefinita, la proprietà restituirà il nome completo dell'oggetto selezionato. Se viene selezionato più di un oggetto, i nomi degli oggetti saranno separati da interruzioni di riga («\n»).

"shortname"

Come sopra, ma restituisce solo nomi brevi per gli oggetti. Ad esempio, a un oggetto all'interno di un elenco verrebbe assegnato solo il nome che ha all'interno dell'elenco, senza il nome dell'elenco.

"label"

Come sopra, ma restituisce RKWard etichette degli oggetti (se nessuna etichetta disponibile, è uguale al nome breve)

Proprietà dell'elenco di stringhe

Questa proprietà contiene un elenco di stringhe.

Nessun modificatore ("")

Per `getValue()/getString()`, vengono restituite tutte le stringhe separate da «\n». Tutti i caratteri «\n» in ogni elemento sono convertiti in «\n» letterale. Tuttavia, l'utilizzo consigliato è quello di recuperare il valore con `getList()`, che restituirà un array di stringhe.

"joined"

Restituisce l'elenco come una singola stringa, con elementi uniti da «\n». Contrariamente a nessun modificatore (""), le singole stringhe sono `_senza_escape`.

Proprietà del codice

Una proprietà detenuta dalle estensioni che ha generato codice. Questa è importante per l'integrazione delle estensioni, al fine di integrare il codice generato dall'estensione integrata nel codice generato dall'estensione di integrazione (di livello superiore). I seguenti valori saranno restituiti in base al modificatore specificato:

Nessun modificatore (")

Restituisce il codice completo, cioè le sezioni «preprocess», «calcola», »printout» e (ma non «preview») concatenate a una stringa.

"preprocess"

Restituisce solo la sezione di pre-elaborazione del codice

"calculate"

Restituisce solo la sezione di calcolo del codice

"printout"

Restituisce solo la sezione di stampa del codice

"preview"

Restituisce la sezione di anteprima del codice

A.2 Elementi di uso generale da utilizzare in qualsiasi file XML (.xml, .rkh, .pluginmap)

<snippets>

Consentito come elemento figlio diretto del nodo <document> e solo lì. Dovrebbe essere posizionato nella parte superiore del file. Consultare la [sezione sull'utilizzo degli snippet](#). Può essere presente un solo elemento <snippets>. Facoltativo, senza attributi.

<snippet>

Definisce un singolo frammento. Consentito solo come figlio diretto dell'elemento <snippets>. Attributi:

<id>

Una stringa identificativa per il frammento. Necessario.

<insert>

Inserisci il contenuto di uno <snippet>. Consentito ovunque. Attributi:

<snippet>

La stringa identificativa del frammento da inserire. Necessario.

<include>

Includere i contenuti di un altro file XML (tutto all'interno dell'elemento <document> di quel file). Consentito ovunque. Attributi:

<file>

Il nome file, relativo alla cartella, si trova nel file attuale. Obbligatorio.

A.3 Elementi da utilizzare nella descrizione XML dell'estensione

Le proprietà detenute dagli elementi sono elencate in una [sezione separata](#).

A.3.1 Elementi generali

<document>

Deve essere presente in ogni file `description.xml` come nodo radice. Nessuna funzione speciale. Nessun attributo

<about>

Informazioni su questa estensione (autore, licenza, ecc.). Questo elemento è consentito sia nel file `.xml` di una singola estensione, sia nel file `.pluginmap`. Fare riferimento al manuale `.pluginmap` per i dettagli di riferimento, [il capitolo sulle informazioni 'informazioni'](#) per un'introduzione.

<code>

Definisce dove cercare il modello JS per l'estensione. Utilizzare solo una volta per file, come figlio diretto del tag `document`. Attributi:

file

Nome file del modello JS, relativo alla cartella in cui si trova il codice xml dell'estensione

<help>

Definisce dove cercare il file della guida per l'estensione. Utilizzare solo una volta per file, come figlio diretto del tag `document`. Attributi:

file

Nome del file della guida, relativo alla cartella in cui si trova il codice xml dell'estensione

<copy>

Può essere utilizzato come elemento figlio (diretto o indiretto) degli elementi di layout principali, cioè `<dialog>` e `<wizard>`. Viene utilizzato per copiare un intero blocco di elementi XML 1:1. Attributi:

id

L'ID da cercare. Il tag `<copy>` cercherà un precedente elemento XML a cui è stato assegnato lo stesso ID e lo copierà includendo tutti gli elementi discendenti.

copy_element_tag_name

In alcuni casi, vorrete una copia quasi letterale, ma cambierete il nome del tag dell'elemento da copiare. L'esempio più importante è quando si desidera copiare un'intera `<tab>` da un'interfaccia alla `<pagina>` di un'interfaccia guidata. In questo caso, impostereste `copy_element_tag_name="page"` per eseguire questa conversione automaticamente.

A.3.2 Definizioni dell'interfaccia

<dialog>

Definisce un'interfaccia di tipo finestra. Inserire la definizione della GUI all'interno di questo tag. Utilizzare solo una volta per file, come figlio diretto del tag `document`. È richiesto almeno uno dei tag «`dialog`» o «`wizard`» per un'estensione. Attributi:

label

Didascalia per la finestra

recommended

La finestra dovrebbe essere utilizzata come interfaccia «consigliata» (cioè l'interfaccia che sarà mostrata in modo predefinito, a meno che l'utente non abbia configurato RKWard per utilizzare, in modo predefinito, un'interfaccia specifica)? Questo attributo non ha attualmente alcun effetto, poiché è implicitamente «vero», a meno che la procedura guidata sia consigliata.

<wizard>

Definisce un'interfaccia simile a una procedura guidata. Inserisci la definizione GUI all'interno di questo tag. Utilizzala una sola volta per file, come tag figlio diretto del tag document. Per un'estensione è richiesto almeno uno tra i tag «dialog» o «wizard». Accetta solo i tag <page> o <embed> come tag figlio diretto. Attributi:

label

Didascalia per la procedura guidata

recommended

La procedura guidata deve essere utilizzata come interfaccia «consigliata» (cioè l'interfaccia che sarà mostrata in modo predefinito, a meno che l'utente non abbia configurato RKWard per utilizzare come impostazione predefinita un'interfaccia specifica)? Facoltativo, il valore predefinito è "false".

A.3.3 Elementi della struttura

Tutti gli elementi in questa sezione accettano un attributo id="stringaidentificatore". Questo attributo è facoltativo per tutti gli elementi. Può essere utilizzato, ad esempio, per nascondere/disabilitare l'intero elemento della disposizione e tutti gli elementi in esso contenuti (vedere [capitolo logica della GUI](#)). La stringa dell'ID non può contenere «.» (punto) o «>» (punto e virgola) e generalmente dovrebbe essere limitata a caratteri alfanumerici e al carattere di sottolineatura ("_"). Vengono elencati solo gli attributi aggiuntivi.

<page>

Definisce una nuova pagina all'interno di una procedura guidata. Consentito solo come figlio diretto di un elemento <wizard>.

<row>

Tutti i figli diretti di un tag «row» saranno posizionati da sinistra a destra.

<column>

Tutti i figli diretti di un tag «column» saranno posizionati dall'alto verso il basso.

<stretch>

Per impostazione predefinita, gli elementi nella GUI occupano tutto lo spazio disponibile. Ad esempio, se avete due colonne affiancate, quella di sinistra è piena di elementi, ma quella di destra contiene un <radio> solitario, il controllo <radio> si espanderà verticalmente, anche se in realtà non ha bisogno dello spazio disponibile, e non sarà visivamente gradevole. In questo caso vorrete davvero aggiungere uno «spazio vuoto» sotto l'elemento <radio>. A questo scopo, utilizzate l'elemento <stretch>. Utilizzerà semplicemente un po' di spazio. Non abusare di questo elemento, di solito è una buona idea per gli elementi della GUI per ottenere tutto lo spazio disponibile, solo a volte la disposizione risulterà distanziata. L'elemento <stretch> non accetta argomenti, nemmeno un "id". Inoltre non potete posizionare figli all'interno dell'elemento <stretch> (in altre parole, lo utilizzerete sempre e solo come "<stretch/>")

<frame>

Disegna una cornice/riquadro attorno ai suoi elementi figli diretti. Può essere utilizzato per raggruppare visivamente opzioni correlate. La disposizione all'interno di una cornice è dall'alto verso il basso, a meno che non si inserisca una <row> al suo interno. Attributi:

label

Didascalia per la cornice (facoltativo)

checkable

Le cornici possono essere controllate. In questo caso, tutti gli elementi contenuti saranno disabilitati quando la cornice è deselezionata e abilitata, quando è selezionata. (facoltativo, il valore predefinito è "false")

checked

Solo per cornici verificabili: la cornice dovrebbe essere controllata in modo predefinito? Il valore predefinito è «true». Non interpretato per cornici non controllabili.

<tabbook>

Organizza gli elementi in un tabbook. Accetta solo tag <tab> come figli diretti.

<tab>

Definisce una pagina in un tabbook. Inserisci la definizione GUI per la scheda all'interno di questo tag. Può essere utilizzato solo come figlio diretto di un tag <tabbook>. Un tag <tabbook> deve avere almeno due schede definite. Attributi:

label

Didascalia per la scheda (richiesto)

<text>

Mostra il testo racchiuso in questo tag nella GUI. Alcuni semplici contrassegni HTML di stile sono supportati (in particolare , <i>, <p> e
). Tuttavia, mantenete la formattazione al minimo. L'inserimento di una riga completamente vuota aggiunge un'interruzione di riga. Attributi:

type

Tipo di testo. Uno tra «normale», «avviso» o «errore». Questo influenza l'aspetto del testo (facoltativo, il valore predefinito è normale)

A.3.4 Elementi attivi

Tutti gli elementi in questa sezione accettano un attributo `id="identifierstring"`. Questo attributo è obbligatorio per tutti gli elementi. Sono elencati solo gli attributi aggiuntivi. La stringa `id` non può contenere «.» (punti).

<varselector>

Fornisce un elenco di oggetti disponibili da cui l'utente può selezionarne uno o più. Richiede uno o più <varslot> come controparte per essere utili. Attributi:

label

Etichetta per il varselector (opzionale, il valore predefinito è «Seleziona variabili»)

<varslot>

Utilizzato insieme a un «varselector» per consentire all'utente di selezionare una o più variabili. Attributi:

label

Etichetta per il varslot (consigliato, il valore predefinito è «Variabile:»)

source

Il varselector da cui prelevare la selezione (obbligatorio, a meno che non ci si connetta manualmente o utilizzando `source_property`)

source_property

Una proprietà arbitraria da cui copiare i valori, quando si fa clic sul pulsante di selezione. Se specificato, questo sovrascrive l'attributo «source».

required

Se, per l'invio del codice, è necessario che questo varslot contenga un valore valido. Vedere [required-property](#) (facoltativo, il valore predefinito è falso)

multi

Se il varslot contiene solo uno (valore predefinito, «falso») o più oggetti

allow_duplicates

Se il varslot può accettare solo oggetti univoci (valore predefinito, «falso») o se lo stesso oggetto può essere aggiunto più volte.

min_vars

Significativo solo se `multi="true"`: numero minimo di variabili da selezionare perché la selezione sia considerata valida (facoltativo, il valore predefinito è «1»)

min_vars_if_any

Significativo solo se `multi="true"`: alcuni varslot possono essere considerati validi, se, ad esempio, il varslot è vuoto o contiene almeno due valori. Questo specifica quante variabili devono essere selezionate se presenti (2 nell'esempio). (facoltativo, il valore predefinito è «1»)

max_vars

Significativo solo se `multi="true"`: numero massimo di variabili da selezionare (facoltativo, il valore predefinito è «0», il che significa che non esiste un massimo)

classes

Se si specifica qui uno o più nomi di classe di R (separati da spazi (« »)), il varslot accetterà solo oggetti appartenenti a quelle classi (facoltativo, *usare con grande attenzione*, all'utente non dovrebbe essere impedito di fare scelte valide e R ha molte classi diverse)

types

Se si specifica uno o più tipi di variabili (separati da spazi (" ")), qui, varslot accetterà solo oggetti di quei tipi. I tipi validi sono "sconosciuto", "numero", "stringa", "fattore", "non valido". (Facoltativo, *usalo con grande attenzione*, all'utente non dovrebbe essere impedito di fare scelte valide e RKWard non sempre conosce il tipo di una variabile)

num_dimensions

Il numero di dimensioni che deve avere un oggetto. «0» (impostazione predefinita) significa che è accettabile qualsiasi numero di dimensioni. (facoltativo, il valore predefinito è «0»)

min_length

La lunghezza minima che un oggetto deve avere per essere accettabile. (facoltativo, il valore predefinito è «0»)

max_length

La lunghezza massima che un oggetto deve avere per essere accettabile. (facoltativo, il valore predefinito è il numero intero più grande rappresentabile sul sistema)

<valueselector>

Fornisce un elenco di stringhe disponibili (non oggetti di R) da selezionare in uno o più <valueslot> associati. Le opzioni delle stringhe possono essere definite utilizzando i tag <option> come figli diretti (vedere di seguito) oppure impostate utilizzando le [proprietà](#) dinamiche. Attributi:

label

Etichetta per il valueselector valori (facoltativo, per impostazione predefinita nessuna etichetta)

<valueslot>

Utilizzato insieme a <valueselector> per consentire all'utente di selezionare uno o più elementi stringa. Questo elemento è pressoché identico a <varslot> e condivide gli stessi attributi, ad eccezione di quelli che fanno riferimento alle proprietà degli elementi accettabili (ad esempio, classi, tipi, num_dimensions, min_length, max_length).

<radio>

Definisce un gruppo di pulsanti radio-esclusivi (è possibile selezionarne solo uno alla volta). Richiede almeno due tag <option> come figli diretti. Non sono consentiti altri tag come figli. Attributi:

label

Etichetta per il controllo radio (consigliato, il valore predefinito è «Seleziona uno:»)

<dropdown>

Definisce un gruppo di opzioni, di cui è possibile selezionarne solo una alla volta, tramite un elenco a tendina. Funziona come un <radio>, ma ha un aspetto diverso. Richiede almeno due tag <option> come figli diretti. Non sono consentiti altri tag come figli. Attributi:

label

Etichetta per l'elenco a tendina (consigliato, il valore predefinito è «Seleziona uno:»)

<select>

Fornisce un elenco di stringhe disponibili da cui l'utente può selezionare un numero arbitrario. Le opzioni delle stringhe possono essere definite utilizzando i tag <option> come figli diretti (vedi sotto) oppure impostate utilizzando le [proprietà](#) dinamiche. Attributi:

label

Etichetta per la <select> (facoltativo, per impostazione predefinita nessuna etichetta)

single

Se impostato a vero, sarà selezionabile solo un singolo valore, anziché valori multipli contemporaneamente (booleano, il valore predefinito è false)

<option>

Può essere utilizzato solo come elemento secondario diretto di <radio>, <dropdown>, <valueselector> oppure <select> elemento. Rappresenta un'opzione selezionabile in un controllo radio o in un elenco a tendina. Poiché <opzione> gli elementi fanno sempre parte di uno degli elementi di selezione, normalmente non hanno un «id» proprio, ma vedi sotto. Attributi:

label

Etichetta per l'opzione (richiesto)

value

Il valore della stringa restituito dall'elemento genitore se questa opzione è marcata/selezionata (richiesto)

checked

Indica se l'opzione deve essere marcata/selezionata per impostazione predefinita: "true" o "false". In una <radio> o <dropdown>, solo un'opzione può essere impostata su *checked="true"* e se nessuna opzione è impostata su marcata, la prima opzione nell'elemento genitore sarà marcata/selezionata automaticamente. In una <select>, qualsiasi numero di opzioni può essere impostato su selezionato. (facoltativo, predefinito su "falso")

id

Specificando il parametro «id» per gli elementi <opzione> è facoltativo (e infatti è consigliabile non impostare un «id», a meno che non ne sia realmente necessario). Tuttavia, specificando un «id» potrete abilitare/disabilitare le <opzioni>, dinamicamente, collegandoti alla proprietà booleana *id_of_radio.id_of_optionX.enabled*. Attualmente funziona per le opzioni all'interno di elementi <radio> o <dropdown>, solo; <valueselector> e <select> le opzioni attualmente non supportano gli ID.

<checkbox>

Definisce una casella di controllo, cioè una singola opzione che può essere attivata o disattivata. Attributi:

label

Etichetta per la casella di controllo (obbligatorio)

value

Il valore restituito dalla casella di controllo se selezionato (obbligatorio)

Introduzione alla scrittura delle estensioni per RKWard

value_unchecked

Il valore che sarà restituito se la casella di controllo non è selezionata (facoltativo, il valore predefinito è "", cioè una stringa vuota)

checked

Se l'opzione dovrebbe essere selezionata in modo predefinito «true» o «false» (opzionale, impostazione predefinita per «false»)

<frame>

L'elemento cornice viene generalmente utilizzato come puro elemento di disposizione ed è elencato nella sezione [elementi di disposizione](#). Tuttavia, può anche essere reso controllabile, agendo così come una semplice casella di controllo allo stesso tempo.

<input>

Definisce un campo di inserimento di testo libero. Attributi:

label

Etichetta per il campo di input (obbligatorio)

initial

Testo iniziale del campo di testo (facoltativo, il valore predefinito è "", cioè una stringa vuota)

size

Uno tra «piccolo», «medio» o «grande». «grande» definisce una campo di inserimento a righe multiple, «piccolo» e «medio» sono campi a riga singola (facoltativo, il valore predefinito è «medio»)

required

Se, per l'invio del codice, è necessario che questo input non sia vuoto. Vedere [required-property](#) (facoltativo, il valore predefinito è falso)

<matrix>

Una tabella per l'immissione di dati di matrici (o vettori) nella GUI.

NOTA

Questo elemento di input *non* è ottimizzato per l'immissione/modifica di grandi quantità di dati. Sebbene non vi siano limiti rigorosi alle dimensioni di una <matrice>, in generale non dovrebbe superare una decina di righe / colonne. Se ci si attende dati più grandi, consentire agli utenti di selezionarli come oggetti di R (che potrebbe essere una buona idea come opzione alternativa, in quasi *ogni* istanza in cui si utilizza un elemento matrice).

Attributi:

label

Etichetta per la tabella (obbligatorio)

mode

Uno tra «intero», «reale» o «stringa». Il tipo di dati che sarà accettato nella tabella (richiesto)

min

Valore minimo accettabile (per matrici di tipo «intero» o «reale»)(facoltativo, il valore predefinito è il più piccolo valore rappresentabile)

max

Valore massimo accettabile (per matrici di tipo «intero» o «reale»)(facoltativo, il valore predefinito è il più grande valore rappresentabile)

allow_missings

Se nella matrice sono consentiti valori mancanti (vuoti). Ciò è implicito per le matrici o la modalità «stringa» (facoltativo, il valore predefinito è falso).

allow_user_resize_columns

Se impostato a vero, l'utente può aggiungere colonne digitando sulle celle più a destra (inattive) (facoltativo, il valore predefinito è true).

Introduzione alla scrittura delle estensioni per RKWard

allow_user_resize_rows

Se impostato a vero, l'utente può aggiungere righe digitando sulle celle più in basso (inattive) (facoltativo, il valore predefinito è true).

rows

Numero di righe nella matrice. Non ha alcun effetto per `allow_user_resize_rows="true"`.

NOTA

Questo può anche essere controllato impostando la proprietà «rows».

(facoltativo, il valore predefinito è 2).

columns

Numero di colonne nella matrice. Non ha alcun effetto per `allow_user_resize_columns="true"`.

NOTA

Questo può anche essere controllato impostando la proprietà «columns».

(facoltativo, il valore predefinito è 2).

min_rows

Numero minimo di righe nella matrice. La matrice rifiuterà la riduzione al di sotto di questa dimensione. (facoltativo, il valore predefinito è 0; vedere anche: *allow_missions*).

min_columns

Numero minimo di colonne nella matrice. La matrice rifiuterà la riduzione al di sotto di questa dimensione. (facoltativo, il valore predefinito è 0; vedere anche: *allow_missions*).

fixed_height

Forza l'elemento della GUI a rimanere alla sua altezza iniziale. Non utilizzare in combinazione con matrici, in cui il numero di righe può cambiare in qualsiasi modo. Utile, specialmente, durante la creazione di un elemento di input vettoriale (`columns="1"`). Con questa opzione impostata a vero, non sarà visualizzata alcuna barra di scorrimento orizzontale, anche se la matrice supera la larghezza disponibile (poiché ciò influirebbe sull'altezza). (facoltativo, il valore predefinito è falso).

fixed_width

Leggermente inaccurato: si supponga che il conteggio delle colonne non cambierà. L'ultima (o in genere l'unica) colonna sarà allungata per occupare la larghezza disponibile. Non utilizzare in combinazione con matrici, in cui il numero di colonne può cambiare in qualsiasi modo. Utile, specialmente, durante la creazione di un elemento di ingresso vettoriale (`rows="1"`). (facoltativo, il valore predefinito è falso).

horiz_headers

Stringhe da utilizzare per l'intestazione orizzontale, separate da «>». L'intestazione sarà nascosta, se impostata su "" (facoltativo, il valore predefinito è il numero di colonna).

vert_headers

Stringhe da utilizzare per l'intestazione verticale, separate da «>». L'intestazione sarà nascosta, se impostata su "" (facoltativo, il numero predefinito è il numero di riga).

<optionset>

Un'interfaccia utente per la ripetizione di un insieme di opzioni per un numero arbitrario di elementi ([introduzione agli insiemi di opzioni](#)). Attributi:

min_rows

Se specificato, l'insieme sarà contrassegnato come non valido, a meno che non abbia almeno questo numero di righe (facoltativo, intero).

min_rows_if_any

Come *min_rows*, ma sarà testato solo se è presente almeno una riga (facoltativo, intero).

max_rows

Se specificato, l'insieme sarà contrassegnato come non valido, a meno che non contenga al massimo questo numero di righe (facoltativo, intero).

keycolumn

ID della colonna per fungere da colonna dei tasti. Un insieme di opzioni con una colonna chiave (valida) fungerà da insieme di opzioni «guidato». Un insieme di opzioni senza colonna chiave consentirà l'inserimento / la rimozione manuale degli articoli. La colonna chiave deve essere contrassegnata come esterna. (facoltativo, il valore predefinito è nessuna colonna dei tasti).

Elementi figli:

<optioncolumn>

Dichiara un'optioncolumn dell'insieme. Per ogni valore che si desidera recuperare dall'insieme di opzioni, è necessario dichiarare una <optioncolumn> separata. Attributi:

id

L'ID dell'optioncolumn (obbligatorio, stringa).

external

Impostato su vero, se l'opzione colonna è controllata dall'esterno dell'insieme di opzioni (facoltativo, booleano, il valore predefinito è false).

label

Se indicato, l'optioncolumn sarà visualizzata in una colonna in base a quell'etichetta (facoltativo, stringa, per impostazione predefinita non è visualizzata).

connect

La proprietà a cui collegare questa colonna di opzioni, fornita come ID all'interno dell'area <content>. Per le <optioncolumn> esterne, il valore corrispondente sarà impostato sul valore impostato esternamente. Per le <optioncolumn> regolari (non esterne), la riga corrispondente della proprietà <optioncolumn> sarà impostata quando la proprietà cambia all'interno dell'area del contenuto. (facoltativo, stringa, il valore predefinito è non connesso).

default

Solo per colonne esterne: il valore da assumere per questa colonna, se non è noto alcun valore per una voce. Raramente utile (Facoltativo, per impostazione predefinita è una stringa vuota)

<content>

Dichiara il contenuto/interfaccia utente dell'insieme. Nessun attributo. Tutti i normali elementi attivi, passivi e di disposizione sono consentiti come elementi childname. Inoltre, nelle versioni precedenti di RKWard (fino alla 0.6.3), lo speciale elemento figlio <optiondisplay> era consentito. Questo è obsoleto in RKWard 0.6.4 e dovrebbe essere semplicemente rimosso dalle estensioni esistenti.

<logic>

Specifica facoltativa della logica dell'interfaccia utente da applicare *all'interno* dell'area dei contenuti dell'insieme di opzioni. Vedere [il riferimento su <logic>](#)

<browser>

Un elemento progettato per selezionare un singolo nome file (o nome di cartella). Notare che questo campo accetterà qualsiasi stringa, anche se è pensato per essere utilizzato solo per i file:

label

Etichetta per il browser (opzionale, il valore predefinito è «Inserisci nome file»)

initial

Testo iniziale del browser (facoltativo, il valore predefinito è "", cioè una stringa vuota)

Introduzione alla scrittura delle estensioni per RKWard

type

Uno tra «file», «dir» o «savefile». Per selezionare un file esistente, una cartella esistente o un file non esistente, rispettivamente (facoltativo, il valore predefinito è «file»)

allow_urls

Se è possibile selezionare URL (non locali) (facoltativo, il valore predefinito è «false»)

filter

Filtro del tipo di file, ad es. ("*.txt*.csv" per i file .txt e .csv). Viene aggiunta automaticamente una voce separata per «Tutti i file» (facoltativo, valore predefinito "", cioè tutti i file)

required

Se, per l'invio del codice, è necessario che il campo non sia vuoto. Notare che ciò non significa necessariamente che il nome file selezionato sia valido. Vedere [required-property](#) (facoltativo, il valore predefinito è true)

<saveobject>

Un elemento progettato per selezionare il nome con cui salvare un oggetto di R (cioè generalmente non già esistente, a differenza di un varslot):

label

Etichetta per l'ingresso (facoltativo, il valore predefinito è «Salva in:»)

initial

Testo iniziale dell'ingresso (facoltativo, il valore predefinito è «my.data»)

required

Se, per l'invio del codice, è necessario che il campo contenga un nome oggetto consentito. Vedere [required-property](#) (facoltativo, il valore predefinito è true)

checkable

In molti casi d'uso, salvare in un oggetto di R è facoltativo. In questi casi, una casella di controllo può essere integrata nell'elemento saveobject usando questo attributo. Se impostato a vero, l'oggetto di salvataggio sarà attivato/disattivato dalla casella di controllo. Vedere [active-property](#) di saveobject (facoltativo, il valore predefinito è falso)

checked

Solo per gli elementi saveobject controllabili: se il controllo è marcato/abilitato per impostazione predefinita (facoltativo, il valore predefinito è false)

<spinbox>

Una casella di selezione in cui l'utente può selezionare un valore numerico, usando l'inserimento diretto da tastiera o le piccole frecce in su/giù. Attributi:

label

Etichetta per la casella di selezione (consigliato, impostazione predefinita per «Inserisci valore:»)

min

Il valore più basso che l'utente può inserire nella casella di selezione (facoltativo, il valore predefinito è il più basso tecnicamente rappresentabile nella casella di selezione)

max

Il valore più grande che l'utente può inserire nella casella di selezione (facoltativo, il valore predefinito è il più alto tecnicamente rappresentabile nella casella di selezione)

initial

Il valore iniziale mostrato nella casella di selezione (facoltativo, il valore predefinito è «0»)

type

Uno di «reale» o «intero». Se la casella di selezione accetterà numeri reali o solo numeri interi (facoltativo, il valore predefinito è «reale»)

default_precision

Significativo solo se la casella di selezione è di tipo="real". Specifica il numero predefinito di posizioni decimali visualizzate nella casella di selezione (saranno visualizzati solo questi zeri finali). Quando l'utente preme le frecce in su/giù, questo punto decimale sarà modificato. L'utente può comunque essere in grado di inserire valori con una precisione maggiore (vedere sotto) (facoltativo, il valore predefinito è «2»)

max_precision

Il numero massimo di cifre che possono essere rappresentate in modo significativo (opzionale, il valore predefinito è «8»)

<formula>

Questo elemento avanzato consente all'utente di selezionare una formula/insieme di interazioni dalle variabili selezionate. Ad esempio per un GLM, questo elemento può essere utilizzato per consentire all'utente di specificare i termini di interazione nel modello. Attributi:

fixed_factors

L'ID del varslot contenente i fattori fissi selezionati (richiesto)

dependent

L'ID del varslot che contiene la variabile dipendente selezionata (richiesto)

<embed>

Integrare un'estensione diversa in questa (consultare il capitolo [sull'integrazione](#)). Attributi:

component

Il nome registrato del componente da integrare (vedere il [capitolo sulla registrazione dei componenti](#)) (obbligatorio)

as_button

Se impostata a «true», solo un pulsante sarà inserito nella GUI di integrazione, la GUI integrata sarà mostrata (in una finestra separata) solo quando si preme il pulsante (opzionale, il valore predefinito è «false»)

label

Significativo solo se *as_button*="true": l'etichetta del pulsante (si consiglia, l'impostazione predefinita è «Opzioni»)

<preview>

Casella di selezione per attivare o disattivare la funzionalità di anteprima. Notare che, a partire dalla versione 0.6.5 di RKWard, gli elementi di anteprima **<preview>** sono un caso speciale nelle finestre delle estensioni (non nelle procedure guidate): saranno posizionati nella colonna dei pulsanti, indipendentemente da dove sono definiti esattamente nell'interfaccia utente. È comunque una buona idea definirli in una posizione ragionevole della disposizione, per compatibilità con le versioni precedenti. Attributi:

label

Etichetta della casella (opzionale, il valore predefinito è «Anteprima»)

mode

Tipo di anteprima. I tipi supportati sono «grafico» (vedere [capitolo sulle anteprime dei grafici](#)), «risultato» (vedere [capitolo sulle \(HTML\) anteprime del risultato](#)), «dati» (vedere [anteprime dei dati](#)) e «personalizzata» (vedere [anteprime personalizzate](#)). (facoltativo, il valore predefinito è «grafico»)

placement

Posizionamento dell'anteprima: «agganciata» (all'area di lavoro principale), «sganciata» (finestra autonoma), «ancorata» (agganciata alla finestra dell'estensione) e «predefinita» (attualmente è uguale a «ancorata», ma potrebbe diventare configurabile dall'utente a un certo punto). In generale, si consiglia di lasciarla come impostazione predefinita per una migliore coerenza dell'interfaccia utente (facoltativo, il valore predefinito è «predefinita»)

active

Se l'anteprima è attiva in modo predefinito. In generale, solo le anteprime ancorate devono essere rese attive per impostazione predefinita e, anche per queste, c'è un motivo per cui l'impostazione predefinita è l'anteprima inattiva (opzionale, l'impostazione predefinita è «falso»)

A.3.5 Sezione logica

<logic>

L'elemento contenitore per la sezione logica. Tutti gli elementi seguenti sono consentiti solo all'interno dell'elemento <logic>. L'elemento <logic> è consentito solo come elemento figlio diretto dell'elemento <document> (al massimo una volta per documento) o di elementi <optionset> (al massimo una volta per insieme di opzioni). La sezione logica del documento si applica sia alle interfacce <finestra> e <procedura guidata> allo stesso modo.

<external>

Crea una nuova proprietà (stringa) che dovrebbe essere connessa a una proprietà esterna se il l'estensione viene integrata. Vedere la sezione [sulle estensioni «incomplete»](#). Attributi:

id

L'ID della nuova proprietà (richiesto)

default

Il valore di stringa predefinito della nuova proprietà, cioè il valore utilizzato, se la proprietà non è connessa a una proprietà esterna (facoltativo, per impostazione predefinita è una stringa vuota)

<i18n>

Crea una nuova proprietà (stringa) che dovrebbe fornire un'etichetta i18n. Attributi:

id

L'ID della nuova proprietà (richiesto)

label

L'etichetta. Questa sarà tradotta. (necessario)

<set>

Imposta una proprietà su un valore fisso (ovviamente, se collega anche la proprietà a qualche altra proprietà, il valore non rimane fisso). Ad esempio, se si incorpora un'estensione, ma si desidera nascondere alcuni dei suoi elementi, potreste impostare la proprietà di visibilità di tali elementi su falso. Utile soprattutto per estensioni incorporate/incorporanti. Nota: se sono presenti più elementi <set> per un singolo *id*, l'ultimo a essere definito ha la precedenza. A volte sarà utile fare affidamento su questo quando si utilizzano le parti <include>. Attributi:

id

ID della proprietà da impostare (obbligatorio)

to

Il valore di stringa su cui impostare la proprietà (obbligatorio). Nota: per proprietà booleane come visibilità, abilitazione, in genere si imposta l'attributo *to* a *to="true"* o a *to="false"*.

<convert>

Crea una nuova proprietà booleana che dipende dallo stato di una o più proprietà diverse. Attributi:

id

L'ID della nuova proprietà (richiesto)

sources

Gli ID delle proprietà da cui dipenderà questa proprietà. Una o più proprietà possono essere specificate, separate da «;» (necessario)

mode

La modalità di conversione/operazione. Uno di «uguale», «non uguale», «intervallo», «e», «o». Se in modalità uguale, la proprietà sarà vera solo se il valore di tutte le sue fonti è uguale all'attributo standard (vedere sotto). Se in modalità non uguale, la proprietà sarà vera solo se il valore di tutte le sue fonti è diverso dallo standard dell'attributo (vedere sotto). Se in modalità intervallo, le fonti devono essere numeriche (intere o reali). La proprietà sarà vera solo se tutte le origini sono nell'intervallo specificato dagli attributi min e max (vedere sotto). Se in modalità e, le fonti devono essere proprietà booleane. La proprietà sarà vera solo se tutte le fonti sono vere contemporaneamente. Se in modalità o, le fonti devono essere proprietà booleane. La proprietà sarà vera solo se almeno una delle fonti è vera. (necessario)

standard

Significativo solo in modalità uguale o nota: il valore stringa da confrontare (richiesto se in una di queste modalità)

min

Significativo solo nell'intervallo di modalità: il valore minimo da confrontare (opzionale, il valore predefinito è il numero in virgola mobile più basso rappresentabile sulla macchina)

max

Significativo solo nell'intervallo di modalità: il valore massimo con cui confrontare (opzionale, il valore predefinito è il più grande numero in virgola mobile rappresentabile sulla macchina)

require_true

Se impostato a "true", la proprietà diventerà obbligatoria e sarà considerata valida solo se il suo stato è true/on. Quindi, se la proprietà è falsa, bloccherà il pulsante **Invia** (facoltativo, il valore predefinito è "false").

CAUTELA

Se lo si utilizza, assicurarsi che l'utente possa facilmente rilevare ciò che è sbagliato, ad esempio mostrando un testo esplicativo.

<switch>

Creare una nuova proprietà che sarà inoltrata a proprietà di destinazione diverse (o stringhe fisse) in base al valore di una proprietà di condizione. Ciò consente di creare logiche simili ai costrutti `if()` o `switch()`. Attributi:

id

L'ID della nuova proprietà (richiesto)

condition

L'ID della proprietà «condition» (richiesto)

Elementi figli:

<true>

Se la proprietà di condizione è booleana, è possibile specificare i due elementi secondari <true> e <false> (e solo questi). (Obbligatorio, se viene fornito anche <false>)

<false>

Se la proprietà di condizione è booleana, è possibile specificare i due elementi secondari <true> e <false> (e solo questi). (Obbligatorio, se viene fornito anche <true>)

<case>

Se la proprietà di condizione non è booleana, è possibile fornire un numero arbitrario di elementi <case>, uno per ogni valore della proprietà di condizione che si desidera far corrispondere (almeno uno di questi elementi è richiesto, se la proprietà di condizione non è booleana)

<default>

Se la proprietà di condizione non è booleana, l'elemento facoltativo <default> consente di specificare il comportamento, se nessun elemento <case> corrisponde al valore della proprietà di condizione (opzionale, consentito solo una volta, in combinazione con uno o più elementi <case>).

Gli elementi figlio <true>, <false>, <case> e <default> accettano i seguenti attributi:

standard

Solo per gli elementi <case>: il valore con cui confrontare la proprietà della condizione (richiesto, stringa).

fixed_value

Una stringa fissa che deve essere fornita come valore della proprietà <switch>, se la condizione attuale corrisponde (obbligatorio, se dynamic_value non è fornito).

dynamic_value

L'>id della proprietà di destinazione che dovrebbe essere fornito come valore della proprietà dell'<interruttore>, se la condizione attuale corrisponde (obbligatorio, se fixed_value non viene fornito).

<connect>

Collega due proprietà. La proprietà del client sarà modificata ogni volta che cambia la proprietà governor (ma non viceversa). Attributi:

client

L'ID della proprietà del client, cioè la proprietà che sarà modificata (necessario)

governor

L'ID della proprietà governor, cioè la proprietà che regolerà la proprietà del cliente. Questo può includere un modificatore (richiesto)

reconcile

Se "true", la proprietà client farà regolare la proprietà del governor sulla connessione in modo tale che la proprietà del governor accetti solo valori accettabili anche dal client (ad es. supponiamo che il governor sia una proprietà numerica con valore minimo «0» e il client sia una proprietà numerica con valore minimo «100». Il minimo di entrambe le proprietà sarà regolato su 100, se reconcile="true"). Funziona generalmente solo per proprietà dello stesso tipo di base (facoltativo, predefinito su "false")

<dependency_check>

Crea una proprietà booleana che è vera se le dipendenze specificate sono soddisfatte, falsa altrimenti. La sintassi XML dell'elemento è la stessa dell'elemento **<dependencies>**, descritto nel manuale [.pluginmap](#). A partire da RKWard 0.6.1, solo le specifiche di versione di RKWard e R vengono prese in considerazione, non le dipendenze dai pacchetti o dalle mappe di estensioni.

<script>

Definire il codice script per controllare la logica dell'interfaccia utente. Vedere [la sezione sulla logica GUI con script](#) per i dettagli. Il codice dello script da eseguire può essere fornito utilizzando l'attributo "file" o come testo (commentato) dell'elemento. L'elemento **<script>** non è consentito nella sezione **<logic>** di un insieme di opzioni. Attributi:

file

Nome del file di script. (necessario)

A.4 Proprietà degli elementi dell'estensione

Tutti gli **elementi di struttura** e tutti gli **elementi attivi** contengono le seguenti proprietà, accessibili tramite «id_elemento.nome_proprietà»:

visible

Decidere se l'elemento della GUI è visibile o meno (booleano)

enabled

Decidere se l'elemento della GUI è abilitato o meno (booleano)

required

Decidere se l'elemento della GUI è richiesto (per contenere un'impostazione valida) o meno. Notare che qualsiasi elemento che è disabilitato o nascosto è anche implicitamente non richiesto (booleano).

Inoltre, alcuni elementi hanno proprietà aggiuntive a cui è possibile connettersi. La maggior parte degli elementi attivi ha anche una proprietà «predefinita» il cui valore sarà restituito sulle chiamate a `getBoolean/getString/getList("...")`, se non è stata nominata alcuna proprietà specifica, come descritto di seguito.

<text>

La proprietà predefinita è `text`

text

Il testo visualizzato (testo)

<varselector>

Nessuna proprietà predefinita

selected

Gli oggetti attualmente selezionati. Probabilmente non si dovranno utilizzare. Utilizzati internamente (RObject)

root

L'oggetto radice/genitore degli oggetti offerti per la selezione (RObject)

<varslot>

La proprietà predefinita è «available»

available

Tutti gli oggetti contenuti nel varslot (RObject)

selected

Degli oggetti contenuti nel varslot, quelli attualmente selezionati. Probabilmente non si dovranno utilizzare. Utilizzati internamente (RObject)

source

Una copia degli oggetti selezionati nel corrispondente selettore. Probabilmente non si dovranno utilizzare. Utilizzati internamente (RObject)

<valueselector>

La proprietà predefinita è «selected»

selected

Le stringhe attualmente selezionate. Modificatore «labeled» per recuperare le etichette corrispondenti. In un `<valueselector>` probabilmente non si desidera utilizzarlo direttamente (solo in un `<select>`). (lettura/scrittura `StringList`)

available

L'elenco di valori di stringa da cui selezionare. (lettura/scrittura StringList)

labels

Etichette da visualizzare per i valori di stringa. (lettura/scrittura StringList)

<valueslot>

Come <varslot>, ma le proprietà sono elenchi di stringhe, anziché RObjects.

<radio>

La proprietà predefinita è «string»

string

Il valore dell'opzione attualmente selezionata (stringa)

number

Il numero dell'opzione attualmente selezionata (le opzioni sono numerate dall'alto verso il basso, a partire da 0) (intero)

<dropdown>

Uguale a <radio>

<select>

Uguale a <valueselector>

<option>

Nessuna proprietà predefinita. «enabled» è l'*unica* proprietà e non è attualmente disponibile per le opzioni all'interno di <select> o <valueselector>. <option> non ha le proprietà «visible» o «required».

enabled

Abilitare o disabilitare questa singola opzione. Nella maggior parte dei casi, si abilita/disabilita l'intero <radio> o <menu a tendina>. Ma questo può essere utilizzato per impostare dinamicamente l'abilitazione di una singola opzione all'interno di un <radio> o <menu a tendina> (bool)

<checkbox>

La proprietà predefinita è «state.labeled», il che significa che sono restituiti i valori specificati dagli attributi *value* e *value_unchecked*, non l'etichetta visualizzata della casella di selezione.

state

Stato della casella di controllo (attivata o disattivata). Nota che utili modificatori di questa proprietà (come di tutte le proprietà booleane) sono «not» e «labeled» (vedere [tipi di proprietà](#)). Tuttavia, spesso è molto utile connettersi alla proprietà senza modificatore, cioè «checkbox_id.state», che restituirà lo stato della casella di controllo in un formato adatto all'uso in un'istruzione if (0 o 1). (booleano)

<frame>

La proprietà predefinita è «checked», se - e solo se - la cornice è controllabile. Per le cornici non verificabili, non esiste una proprietà predefinita.

checked

Disponibile solo per cornici selezionabili: stato della casella di controllo (attiva o disattiva). Notare che utili modificatori di questa proprietà (come di tutte le proprietà booleane) sono «not» e «numeric» (vedere [tipi di proprietà](#)). (booleano)

<input>

La proprietà predefinita è «text»

text

Testo attuale nel campo di inserimento (stringa)

<matrix>

La proprietà predefinita è «cbind».

rows

Numero di righe nella matrice (numero intero). Se la matrice consente all'utente di aggiungere / rimuovere righe, questa proprietà deve essere considerata di sola lettura. Altrimenti, cambiandolo, cambierà la dimensione della matrice.

columns

Numero di colonne nella matrice (numero intero). Se la matrice consente all'utente di aggiungere / rimuovere colonne, questa proprietà deve essere considerata di sola lettura. Altrimenti, cambiandolo, cambierà la dimensione della matrice.

tsv

Dati nella matrice in formato tsv (stringa; lettura-scrittura). Si noti che rispetto alla solita struttura tsv, *colonne*, non righe, sono separate da caratteri da caratteri di interruzione di riga e le celle all'interno di una colonna sono separate da caratteri di tabulazione.

0,1,2...

I dati da una singola colonna (0 per la colonna più a sinistra). `getValue()` / `getString()` restituisce questo come una singola stringa, separata da «\n». Tuttavia, il modo consigliato per ottenerlo è utilizzare `getList()`, che restituisce questa colonna come un vettore di stringhe.

riga.0,riga.1,riga.2...

I dati da una singola riga (0 per la riga più in alto). `getValue()` / `getString()` restituisce questo come una singola stringa, separata da «\n». Tuttavia, il modo consigliato per ottenerlo è utilizzare `getList()`, che restituisce questa riga come un vettore di stringhe.

cbind

Dati in un formato adatto per incollare in R, racchiusi in un'istruzione cbind (stringa; sola lettura).

<optionset>

Nessuna proprietà predefinita.

row_count

Numero di elementi nell'insieme di opzioni (intero). Sola lettura.

current_row

Elemento attualmente attivo nell'insieme di opzioni (intero). -1 per nessun elemento attivo. Lettura-scrittura.

optioncolumn_ids

Per ogni <optioncolumn> specificata, sarà creata una proprietà dell'elenco di stringhe con l'id specificato.

<browser>

La proprietà predefinita è «selection»

selection

Testo attuale (nome file selezionato) nel browser (stringa)

sovrascrivere

Se l'opzione «sovrascrivi» è selezionata (booleano, di sola lettura, ovvero è possibile leggere lo stato della casella di controllo, ma non modificarlo, a livello di programmazione)

<saveobject>

La proprietà predefinita è «selection»

selection

Nome completo dell'oggetto selezionato (stringa; sola lettura - per impostarlo a livello di codice, utilizzare «parent» e «objectname»)

Introduzione alla scrittura delle estensioni per RKWard

parent

L'oggetto principale dell'oggetto selezionato. Questo è sempre un oggetto di R oggetto di un tipo che può contenere altri oggetti (ad es. un elenco o una cornice di dati). Se impostato su una stringa vuota o un oggetto non valido, viene assunto «.GlobalEnv» (RObject)

objectname

Il nome base dell'oggetto selezionato, cioè la stringa inserita dall'utente (modificata in un nome R valido, se necessario) (stringa)

active

Solo per getti di salvataggio verificabili: se il controllo è selezionato/abilitato. Sempre vero per saveobject non marcabili (booleano)

<spinbox>

La proprietà predefinita è «int» o «real.formatted» a seconda della modalità della casella di selezione

int

Valore intero mantenuto dalla casella di selezione o intero più vicino, se in modalità reale (intero)

real

Valore reale mantenuto dalla casella di selezione (e intero, se in numero intero) (reale)

<formula>

La proprietà predefinita è «model»

model

La stringa del modello attuale (stringa)

table

La cornice di dati che contiene le variabili richieste. Se vengono utilizzate variabili di una sola cornice, viene restituito il nome di quella cornice. Altrimenti viene creata una nuova cornice di dati come richiesto (stringa)

labels

Se sono coinvolte variabili da più cornici di dati, i loro nomi potrebbero essere alterati (ad esempio, se entrambe le cornici contengono una variabile denominata «x»). Ciò restituisce un elenco con i nomi alterati come indici e l'etichetta descrittiva come valore (stringa)

fixed_factors

I fattori costanti. Probabilmente non si dovranno utilizzare. Utilizzati internamente (RObject)

dependent

Le variabili dipendenti. Probabilmente non si dovranno utilizzare. Utilizzate internamente (RObject)

<embed>

Nessuna proprietà predefinita

code

Il codice generato dall'estensione integrata (codice)

<preview>

La proprietà predefinita è «state»

state

Se la casella di anteprima è selezionata (non necessariamente se l'anteprima è già stata mostrata) (booleano)

<convert>

Questo elemento (utilizzato nella sezione <logic>) è speciale, in quanto tecnicamente *è* una proprietà, invece di contenere solo una o più proprietà. È di tipo booleano. Nota che i modificatori utili di questa proprietà (come di tutte le proprietà booleane) sono «not» e «numeric» (vedere [tipi di proprietà](#))

<switch>

Questo elemento (utilizzato nella sezione <logica>) è speciale, in quanto tecnicamente *è* una proprietà (stringa), invece di contenere solo una o più proprietà. Permette di passare tra diverse proprietà di destinazione a seconda del valore di una proprietà di condizione o di rimappare i valori della proprietà di condizione. Tutti i modificatori forniti vengono passati alle proprietà di destinazione, quindi, ad es. se tutte le proprietà di destinazione sono proprietà RObject, potete utilizzare anche il modificatore "shortname" sullo switch. Tuttavia, se le proprietà di destinazione sono di tipo diverso, l'utilizzo dei modificatori potrebbe causare errori. Per *fixed_value*, qualsiasi modificatore viene eliminato silenziosamente. Tenete presente che le proprietà di destinazione, quando si accede tramite uno switch, sono sempre di sola lettura.

A.5 Estensioni integrabili fornite dalla versione ufficiale di RKWard

Un certo numero di estensioni integrabili è fornito con RKWard, e può essere utilizzato nelle proprie estensioni. La documentazione dettagliata è attualmente disponibile solo nella fonte di queste estensioni o nei file della guida. Tuttavia, ecco un elenco per darvi una rapida panoramica di ciò che è disponibile:

ID	Mappa delle estensioni	Descrizione	Esempio d'uso
rkward::plot_options	embedded.pluginmap	Fornisce una vasta gamma di opzioni per i grafici. La maggior parte delle estensioni di tracciamento dei grafici la utilizza.	Grafici->Grafici a barre, la maggior parte delle altre estensioni di tracciamento dei grafici
rkward::color_chooser	embedded.pluginmap	Estensione molto semplice per specificare un colore. L'attuale implementazione fornisce un elenco di nomi di colori. Le future implementazioni potrebbero fornire una scelta del colore più elaborata.	Grafici->Istogramma
rkward::plot_stepfun_options	embedded.pluginmap	Opzioni del grafico della funzione a gradino	Grafici->Grafico ECDF

rkward::histogram_options	embedded.pluginmap	Opzioni istogramma (grafico)	Grafici->Istogramma
rkward::barplot_embed	embedded.pluginmap	Opzioni del grafico a barre	Grafici->Grafico a barre
rkward::one_var_tabulation	embedded.pluginmap	Fornisce una tabulazione su una singola variabile.	Grafici->Grafico a barre
rkward::limit_vector_length	embedded.pluginmap	Limita la lunghezza di un vettore (agli n elementi più grandi o più piccoli).	Grafici->Grafico a barre
rkward::level_select	embedded.pluginmap	Fornisce un <varselector> riempito con i livelli (o valori univoci) di un vettore.	Dati->Ricodifica i dati categorici
rkward::multi_input	embedded.pluginmap	Combina caselle di selezione, controlli di inserimento e radio per fornire ingressi per caratteri, numeri e dati logici.	Dati->Ricodifica i dati categorici

Tabella A.1: Estensioni standard integrabili

A.6 Elementi da utilizzare nei file di .pluginmap

<document>

Deve essere presente in ogni file di .pluginmap come nodo radice (esattamente una volta).

Attributi:

base_prefix

Nomi file specificati nella .pluginmap si presume che il file sia relativo alla cartella dei file di .pluginmap + il prefisso specificato qui. Utile, in particolare, se tutti i componenti si trovano sotto una singola sottocartella.

namespace

Uno spazio dei nomi per gli ID dei componenti. Quando si cercano componenti per l'integrazione, i componenti saranno recuperabili tramite una stringa «namespace:id_componente». Impostare su «rkward» per ora.

id

Una stringa identificativa facoltativa per questa .pluginmap. Specificando questa opzione, i terzi autori possono fare riferimento e caricare la vostra .pluginmap dalle loro (vedere il [capitolo sulla gestione delle dipendenze](#)

priority

Uno tra "nascosto", "basso", "medio" o "alto". I .pluginmap con priorità "media" o "alta" vengono attivati automaticamente quando RKWard li trova. Usare *priority="hidden"* per i .pluginmap che non devono essere attivati, la cartella (destinati solo all'inclusione). Tuttavia, nell'attuale implementazione questo in realtà non nasconde il .pluginmap. (Facoltativo, il valore predefinito è "medio").

<dependencies>

Questo elemento, che specifica le dipendenze, è consentito come figlio diretto dell'elemento <document> (una volta) e come elemento figlio di elementi <component> (una volta per ogni elemento <component>). Specificare le dipendenze che devono essere soddisfatte per poter utilizzare le estensioni. Vedere il [capitolo sulle dipendenze](#) per una panoramica. Attributi:

rkward_min_version, rkward_max_version

Versione minima e massima consentita di RKWard. Le specifiche della versione possono includere suffissi non numerici, come «0.5.7z-devel1». Se una dipendenza specificata non viene soddisfatta, le estensioni che si applicano a *saranno ignorate*. [Ulteriori informazioni](#). Facoltativo; se non specificato, nessuna versione minima / massima di RKWard sarà richiesta.

R_min_version, R_max_version

Versione minima e massima consentita di R. Le specifiche di versione possono *non* includere suffissi non numerici, come «0.5.7z-devel1». La dipendenza della versione di R sarà mostrata nelle pagine della guida delle estensioni, ma non ha alcun effetto diretto, a partire da RKWard 0.6.1. [Ulteriori informazioni](#). Opzionale; se non specificato, nessuna versione minima / massima di R sarà richiesta.

piattaforme

Piattaforme in cui questa estensione è disponibile. I valori supportati sono *unix*, *windows*, *macos*, *qualsiasi* e combinazioni separate da due punti (ad esempio *unix:macos*). *unix* include qualsiasi variante Linux e BSD, ma *non* MacOS. Se la vostra estensione non dipende dalla piattaforma, omettete semplicemente questo attributo.

Elementi figli:

<package>

Aggiunge una dipendenza da uno specifico pacchetto di R. Attributi:

name

Nome del pacchetto (richiesto).

min_version, max_version

Versione minima / massima consentita (facoltativo).

repository

Deposito in cui è possibile trovare il pacchetto. Facoltativo, ma altamente consigliato, se il pacchetto non è disponibile su CRAN.

<pluginmap>

Aggiunge una dipendenza da una specifica .pluginmap di RKWard. Attributi:

name

Stringa ID della .pluginmap richiesta (necessario).

min_version, max_version

Versione minima / massima consentita (facoltativo).

url

URL dove può essere trovata la .pluginmap. Necessario.

<about>

Può essere presente esattamente una volta come figlio diretto dell'elemento <document>. Contiene meta informazioni sulla .pluginmap (o estensione). Vedere il [capitolo su «informazioni»](#) per una panoramica. Attributi:

name

Nome visibile dell'utente. Facoltativo. Non deve essere uguale a «id».

version

Numero della versione. Facoltativo. Il formato non è limitato, ma per essere al sicuro, segui gli schemi di versione comuni come «x.y.z».

Introduzione alla scrittura delle estensioni per RKWard

releasedate

Specifica della data di rilascio. Facoltativo nel formato «AAAA-MM-GG».

shortinfo

Una descrizione *breve* dell'estensione / .pluginmap. Facoltativo.

url

URL dove è possibile trovare ulteriori informazioni. Facoltativo, ma consigliato.

copyright

Specifiche del copyright, ad es. "2012-2013 di Mario Rossi". Facoltativo, ma consigliato.

licence

Specifiche della licenza, ad es. «GPL» o «BSD». Assicurarsi di accompagnare i propri file con una copia completa della relativa licenza. Facoltativo, ma consigliato.

category

Categoria di estensione, ad es. «Teoria della risposta dell'item». A partire da RKWard 0.6.1, nessuna categoria è predefinita. Facoltativo.

Elementi figli:

<author>

Aggiunge informazioni su un autore. Attributi:

nome, nome proprio, cognome

Specificare il nome completo per *nome* o specificare sia *nome proprio* che *cognome*, separatamente.

ruolo

Descrizione del ruolo dell'autore (facoltativo).

posta elettronica

Indirizzo di posta elettronica, dove è possibile contattare l'autore. Necessario. Può essere impostato alla lista di distribuzione rkward-devel, se si è iscritti, e la propria estensione è pensata per essere inclusa nella versione ufficiale di RKWard.

url

URL con ulteriori informazioni sull'autore, ad es. [pagina principale](#) (facoltativo).

<components>

Deve essere presente esattamente una volta come figlio diretto dell'elemento <document>. Contiene i singoli elementi <component> descritti di seguito. Nessun attributo.

<component>

Uno o più elementi <component> dovrebbero essere dati come figli diretti dell'elemento <component> (e solo lì). Registra un componente/estensione con rkward. Attributi:

type

Per future estensioni: il tipo di componente/estensione. Sempre impostato su «standard» per ora (l'unico tipo attualmente supportato).

id

L'ID con cui è possibile recuperare questo componente (per posizionarlo nel menu (vedere di seguito) o per integrarlo). Vedere lo spazio dei nomi <document> sopra.

file

Richiesto almeno per i componenti di tipo="standard": il nome del file XML che descrive la GUI.

label

L'etichetta per questo componente, quando inserito nella gerarchia dei menu.

facoltativo

Significativo solo per componenti con [dipendenze](#) definite: solitamente è considerato un errore segnalabile se un componente non è compatibile con questa versione di RKWard. Tuttavia, se il componente non è realmente necessario nell'ambiente attuale, l'impostazione di questo attributo su *"true"* sopprime qualsiasi avviso (*"false"* per impostazione predefinita).

<attribute>

Definisce un attributo di un componente. Finora solo significativo per le [estensioni di importazione](#). Consentito solo come figlio diretto di <component>. Attributi:

id

ID dell'attributo

value

Valore dell'attributo

labels

Etichetta associata all'attributo

<hierarchy>

Deve essere presente esattamente una volta come figlio diretto dell'elemento <document>. Descrive dove i componenti dichiarati sopra devono essere collocati nella gerarchia dei menu. Accetta solo elementi <menu> come figli diretti. Nessun attributo.

<menu>

Uno o più <menu> dovrebbero essere forniti come figli diretti dell'elemento <hierarchy>. Dichiarare un nuovo (sotto)menu. Se esiste già un menu con l'ID specificato (vedi sotto), i due menu vengono uniti. Il <menu> è consentito come elemento figlio diretto dell'elemento <hierarchy> (menu di livello superiore) o come elemento secondario diretto in qualsiasi altro elemento <menu> elemento (sottomenu). Al contrario, il menu <menu> accetta altri elementi <menu> o elementi <entry> come figli. Attributi:

id

Una stringa identificativa del menu. Utile, quando vengono lette le definizioni dei menu da diversi file di `.pluginmap`, per assicurarsi che le estensioni possano essere inserite negli stessi menu. Alcuni ID di menu come «file» si riferiscono a menu predefiniti (in questo caso il menu «File»). Assicurarsi di controllare con i file di `.pluginmap` esistenti per utilizzare ID coerenti.

label

Un'etichetta per il menu.

group

Consente di controllare l'ordinamento delle voci di menu. Vedere [ordinamento delle voci di menu](#). Facoltativo.

<entry>

Una voce di menu, cioè un'opzione di menu per invocare un'estensione. Può essere utilizzata solo come figlia diretta di un elemento di menu, non accetta elementi figlio. Attributi:

component

L'ID del componente che deve essere richiamato quando viene attivata questa voce di menu.

group

Consente di controllare l'ordinamento delle voci di menu. Vedere [ordinamento delle voci di menu](#). Facoltativo.

<group>

Dichiara un gruppo di voci nel menu. Vedere [ordinamento delle voci di menu](#). Attributi:

id

Il nome di questo gruppo.

separated

Facoltativo. Se impostata su «vero», l'elemento in questo gruppo sarà separato visivamente dagli elementi circostanti.

group

Il nome del gruppo a cui aggiungere questo gruppo (facoltativo).

<context>

Dichiara le voci in un [contesto](#). Consentito solo come figlio diretto del tag <document>. Accetta solo tag <menu> come figli diretti. Attributi:

id

L'ID del contesto. Finora sono stati implementati solo due contesti: «x11» e «import».

<require>

Includere un altro file `.pluginmap`. Questo file `.pluginmap` viene caricato una sola volta, anche se è <require> da diversi altri file. Il caso d'uso più importante è includere un file `pluginmap`, che dichiara alcuni componenti, che sono incorporati dai componenti dichiarati in questo `.pluginmap`. Gli elementi <require> sono consentiti solo come figli diretti del nodo <document>. Attributi:

file

Il nome del file della `.pluginmap` da includere. Questo è visto in relazione alla cartella dell'attuale file di `.pluginmap` + il prefisso_base (vedere sopra, elemento <document>). Se non si conosce il percorso relativo alla `.pluginmap` da includere, usare invece l'attributo `map` per fare riferimento ad esso tramite ID.

map

Per includere un file `.pluginmap` da un pacchetto diverso (o un file `.pluginmap` di RKWard dal proprio file `.pluginmap` esterno), è possibile farvi riferimento tramite il suo `namespace::id`, come specificato nell'elemento <document> del file `pluginmap` richiesto. L'inclusione non riuscirà se non è noto alcun file `.pluginmap` con quell'ID (ad esempio, non installato sul sistema dell'utente). Si dovrebbe usare questo metodo solo per includere file `.pluginmap` esterni al proprio pacchetto. Per le mappe interne al pacchetto, specificare un percorso relativo (attributo `file`) è più veloce e affidabile.

A.7 Elementi da utilizzare nei file `.rkh` (guida)

<document>

Deve essere presente in ogni file `.xml` come nodo radice (esattamente una volta). Nessun attributo.

<title>

Titolo della pagina di aiuto. Questo *non* viene interpretato per le pagine di aiuto di un'estensione (prende il titolo dall'estensione stessa), ma solo per le pagine autonome. Nessun attributo. Il testo contenuto nel tag <title> diventerà la didascalia della pagina di aiuto. Può essere definito solo una volta, come figlio diretto del nodo <document>.

<summary>

Un breve riepilogo della pagina della guida (o per quale scopo è utilizzata questa estensione). Questo sarà mostrato sempre nella parte superiore della pagina della guida. Nessun attributo. Il testo contenuto nell'etichetta <summary> sarà visualizzato. Consigliato, ma non richiesto. Può essere definito una sola volta, come figlio diretto del nodo <document>.

<usage>

Un riepilogo leggermente più elaborato dell'utilizzo. Questo sarà sempre mostrato direttamente dopo <summary>. Nessun attributo. Sarà visualizzato il testo contenuto nel tag <usage>. Consigliato per le pagine della guida dell'estensione, ma non richiesto. Può essere definito una sola volta, come figlio diretto del nodo <document>.

<section>

Una sezione per scopi generali. Può essere utilizzata un numero qualsiasi di volte come figlio diretto del nodo <document>. Queste sezioni sono visualizzate nell'ordine della loro definizione, ma tutte *dopo* la sezione <usage> e *prima* della sezione <settings>. Sarà visualizzato il testo contenuto nel tag <section>.

id

Un identificatore necessario per saltare a questa sezione dalla barra di navigazione (o un collegamento). Deve essere univoco all'interno del file. Richiesto, nessun valore predefinito.

title

Il titolo (didascalia) di questa sezione. Richiesto, nessun valore predefinito.

short_title

Un titolo breve adatto per essere visualizzato nella barra di navigazione. Facoltativo, per impostazione predefinita il titolo completo.

<settings>

Definisce la sezione contenente riferimenti sulle diverse impostazioni della GUI. Significativo solo e utilizzato solo per le pagine della guida relative alle estensioni. Utilizzare come figlio diretto di <document>. Può contenere solo elementi <setting> e <caption> come figli diretti. Nessun attributo.

<setting>

Spiega una singola impostazione nella GUI. Consentito solo come figlio diretto dell'elemento <settings>. Viene visualizzato il testo contenuto nell'elemento.

id

L'ID dell'impostazione nell'estensione .xml. Richiesto, nessun valore predefinito.

title

Un titolo facoltativo per l'impostazione. Se omissso (si consiglia l'omissione nella maggior parte dei casi), il titolo sarà preso dall'estensione .xml.

<caption>

Una didascalia per raggruppare visivamente diverse impostazioni. Può essere utilizzata solo come figlio diretto dell'elemento <settings>.

id

L'ID dell'elemento corrispondente (in genere <frame>, <page> o <tab>) nell'estensione .xml.

title

Un titolo facoltativo per la didascalia. Se omissso (si consiglia l'omissione nella maggior parte dei casi), il titolo sarà preso dall'estensione .xml.

<related>

Definisce una sezione contenente collegamenti a ulteriori informazioni correlate. Sarà sempre visualizzata dopo la sezione <settings>. Nessun attributo. Sarà visualizzato il testo contenuto nel tag <related>. In genere questo conterrà un elenco in stile HTML. Consigliato per le pagine della guida dell'estensione, ma non richiesto. Può essere definito una sola volta, come figlio diretto del nodo <document>.

<technical>

Definisce una sezione contenente informazioni tecniche non pertinenti per gli utenti finali (come la struttura interna dell'estensione). Sarà sempre visualizzata per ultima in una pagina della guida. Nessun attributo. Sarà visualizzato il testo contenuto nel tag `<related>`. Non richiesto e non consigliato per la maggior parte delle pagine della guida delle estensioni. Può essere definito una sola volta, come figlio diretto del nodo `<document>`.

<link>

Un collegamento. Può essere utilizzato in una qualsiasi delle sezioni sopra descritte.

href

La destinazione e l'URL. Notare che diversi URL specifici di RKWard sono disponibili. Vedere la [sezione per scrivere le pagine della guida](#) per i dettagli.

<label>

Inserisce il valore di un'etichetta dell'interfaccia utente. Può essere utilizzato in una qualsiasi delle sezioni sopra descritte.

id

L'ID dell'elemento nell'estensione, di cui copiare l'etichetta *label*.

<diversi tag HTML>

Altri tag HTML di base sono ammessi all'interno delle sezioni. Mantieni comunque la formattazione manuale al minimo.

A.8 Funzioni disponibili per la programmazione della logica della GUI

Classe «Component»

Classe che rappresenta un singolo componente o proprietà del componente. L'istanza più importante di questa classe è la variabile «gui», che è predefinita come proprietà radice del componente attuale. I seguenti metodi sono disponibili per le istanze della classe «Component»:

absoluteId(id_base)

Restituisce l'ID assoluto di *id_base* o, se *id_base* viene omissso, l'identificatore del componente.

getValue(id)

Sconsigliato. Utilizzare invece `getString()`, `getBoolean()` o `getList()`. Restituisce il valore della proprietà figlia specificata. Restituisce il valore di questa proprietà, se l'ID viene omissso.

getString(id)

Restituisce il valore della proprietà figlia specificata come stringa. Restituisce il valore di questa proprietà, se l'ID viene omissso.

getBoolean(id)

Restituisce il valore della proprietà figlia specificata come booleano (se possibile). Restituisce il valore di questa proprietà, se l'ID viene omissso.

getList(id)

Restituisce il valore della proprietà figlia specificata come vettore di stringhe (se possibile). Restituisce il valore di questa proprietà, se l'ID viene omissso.

setValue(id, valore)

Impostare il valore della proprietà figlio specificata su *valore*.

getChild(id)

Restituisce un'istanza della proprietà figlio con l'*id* indicato.

addChangeCommand(id, comando)

Eseguire *comando* ogni volta che la proprietà figlio specificata da *id* cambia. *id* può essere specificato come una singola stringa o come un array di id (se la funzione deve essere chiamata per le modifiche in più proprietà). *command* è un valore richiamabile (solitamente una funzione), tuttavia, per compatibilità con le estensioni scritte per le versioni precedenti di RKWard, può anche essere specificato come una stringa da valutare.

La funzione restituisce il parametro *comando*, per comodità (in modo da poterlo, ad esempio, assegnare a una variabile e/o richiamarlo durante l'inizializzazione).

Classe «RObject»

Classe che rappresenta un singolo oggetto di R. Un'istanza di questa classe può essere ottenuta usando **makeRObject(nome_oggetto)**. I seguenti metodi sono disponibili per le istanze della classe «RObject»:

ATTENZIONE

Se nel backend sono ancora presenti comandi, le informazioni fornite da questi metodi possono essere obsolete al momento dell'esecuzione del codice dell'estensione. *Non* farvi affidamento per operazioni critiche (rischio di perdita di dati).

getName()

Restituisce il nome assoluto dell'oggetto.

exists()

Restituisce se l'oggetto esiste. In genere, è necessario verificarlo prima di utilizzare uno dei metodi elencati di seguito.

dimensions()

Restituisce un vettore di dimensioni (simile a **dim()** in R).

classes()

Restituisce un vettore di classi (simile a **class()** in R).

isClass(class)

Restituisce vero, se l'oggetto appartiene alla classe *class*.

isDataFrame()

Restituisce vero, se l'oggetto è una cornice di dati.

isMatrix()

Restituisce vero, se l'oggetto è una matrice.

isList()

Restituisce vero, se l'oggetto è un elenco.

isFunction()

Restituisce vero, se l'oggetto è una funzione.

isEnvironment()

Restituisce vero, se l'oggetto è un ambiente.

isDataNumeric()

Restituisce vero se l'oggetto è un vettore di dati numerici.

isDataFactor()

Restituisce vero, se l'oggetto è un vettore di dati fattoriali.

isDataCharacter()

Restituisce vero, se l'oggetto è un vettore di dati caratteriali.

isDataLogical()

Restituisce vero, se l'oggetto è un vettore di dati logici.

parent()

Restituisce un'istanza di «RObject» che rappresenta il genitore di questo oggetto.

child(childname)

Restituisce un'istanza di «RObject» che rappresenta il figlio *nome_figlio* di questo oggetto.

Classe «RObjectArray»

Un vettore di istanze di RObject. Un'istanza di questa classe può essere ottenuta utilizzando **makeRObjectArray(nomi_oggetti)**. È particolarmente utile quando si tratta di «varslot» che consentono di selezionare più oggetti.

Funzione include()

include(filename) può essere utilizzato per includere un file JS separato.

Funzione doRCommand()

Obsoleto. Non utilizzare nelle nuove estensioni: **doRCommand(comando, callback)**. Utilizzare invece **new RCommand()**.

new RCommand()-function

new RCommand(comando, id_facoltativo) può essere utilizzato per richiedere informazioni a R. Leggere la sezione su come [interrogare R dall'interno di un'estensione](#) per dettagli e avvertenze.

Appendice B

Risoluzione dei problemi durante lo sviluppo delle estensioni

Si è letta tutta la documentazione, fatto tutto correttamente e non si riesce ancora a farlo funzionare? Non c'è da preoccuparsi, lo risolveremo. La prima cosa da fare è: attivare la finestra **Messaggi di debug di RKWard** finestra (disponibile dal menu **Finestre**, o fare clic con il tasto destro del mouse su una delle barre degli strumenti), quindi avviare nuovamente l'estensione. Come regola generale, non si dovrebbe vedere alcun risultato nella finestra dei messaggi quando viene invocata la propria estensione o in qualsiasi altro momento. Se ce n'è uno, è probabilmente correlato alla propria estensione. Vedere se è di aiuto.

Se sulla console tutto sembra a posto, provare ad aumentare il livello di debug (dalla riga di comando, utilizzando **rkward --debug-level 3**, o impostando il livello di debug a 3 in **Impostazioni** → **Configura RKWard** → **Debug**). Non tutti i messaggi mostrati ai livelli di debug più alti indicano necessariamente un problema, ma è probabile che il problema si presenti in qualche punto tra i messaggi.

Se ancora non si riesce a scoprire cosa c'è che non va, non disperare. Sappiamo che si tratta di cose complicate e, dopotutto, probabilmente si è incontrato un bug in RKWard, e RKWard deve essere corretto. Basta scrivere alla lista di distribuzione di sviluppo e parlarci del problema. Saremo felici di aiutarvi.

Infine, anche se avete scoperto come farlo da soli, ma avete riscontrato che la documentazione non è così utile o addirittura sbagliata per alcuni aspetti, vi preghiamo di comunicarcelo anche sulla lista di distribuzione, in modo che possiamo correggere/migliorare la documentazione.

Appendice C

Licenza

Traduzione italiana di Vincenzo Reale smart2128vr@gmail.com

Questa documentazione è concessa in licenza sotto i termini della [GNU Free Documentation License](#).