shareppy
FINANCIAL SERVICES

Conceptos principales

API Collections

1. Generalidades

A través del tag item se definen los elementos de la colección. Es posible también definir elementos de la colección mediante "" o ${}.

Si la colección usa clave. El id del tag item#id define la clave. El atributo por defecto es el valor de la colección.

Es posible usar bucles (for, while) y condicionales (if) en la creación de los elementos de la colección. también se pueden usar los atributos _if, _while, _foreach.

Cuando se usan los tags for, while e if los elementos de la colección se tendrán que definir con el tag item.

2. hash_map

Crea un objeto equivalente al HashMap de java.

```
var#map1 > hash_map {
   item#key1("value1")
   item#key2(${value2})
   item("value3" id=${key3})
   ...
}
```

3. Convertir un cursor a un mapa

```
var#map1 > hash_map {
      item#campo1(@cursor{cursor.campo1})
      item#campo2(@cursor{cursor.campo2})
      item#campo3(@cursor{cursor.campo3})
      …
 }
```

4. array_list

Crea un objeto equivalente al ArrayList de java

```
var#map1 > array_list {
   item("value1")
   item(${value2})
   item("value3")
   ...
}
```

5. Crear una lista de maps de un cursor

```
var#map1 > array_list {
      item(_while=${cursor}) > hash_map {
      item#campo1(@cursor{cursor.campo1})
      item#campo2(@cursor{cursor.campo2})
      item#campo3(@cursor{cursor.campo3})
      ...
   }
}
```

6. hash_set

Crea un objeto equivalente al HashSet de java

```
var#map1 > hash_set {
   item("value1")
   item(${value2})
   item("value3")
   ...
}
```

7. Crear un set de maps de un cursor

```
var#map1 > hash_set {
   item(_while=${cursor}) > hash_map {
      item#campo1(@cursor{cursor.campo1})
      item#campo2(@cursor{cursor.campo2})
      item#campo3(@cursor{cursor.campo3})
      ...
   }
}
```

8. json_object

Crea un String equivalente a un objeto json.

```
var#map1 > json_object {
   item#key1("value1")
   item#key2(${value2})
   item("value3" id=${key3})
   ...
}
```

9. Convertir un cursor a un JSON.

```
var#map1 > json_object {
   item#campo1(@cursor{cursor.campo1})
   item#campo2(@cursor{cursor.campo2})
   item#campo3(@cursor{cursor.campo3})
   ...
}
```

10. json_array

Crea un String equivalente al JSON array

```
var#map1 > json_list {
   item("value1")
   item(${value2})
   item("value3")
   ...
}
```

11. Crear una array de objetos de un cursor

```
var#map1 > json_array {
   item(_while=${cursor}) > json_object {
      item#campo1(@cursor{cursor.campo1})
      item#campo2(@cursor{cursor.campo2})
      item#campo3(@cursor{cursor.campo3})
      ...
   }
}
```

Otros tipos de Maps

Sus usos son similares al hash_map

- linked_map: Equivalente a un LinkedHashMap
- tree_map: Equivalente a un TreeMap
- fill_map(${mapVar}): Se usa para llenar una variable de tipo Map con nuevas claves

Otros tipos de Listas

Su uso es similar al array_list

- linked_list: Equivalente a un LinkedList
- fill_list(${mapVar}): Se usa para llenar una variable de tipo List con nuevos elementos.

Otros tipos de Sets

Su uso es similar al hash_set

- tree_set: Equivalente a un TreeSet.
- linked_set: Equivalente a un LinkedHashSet.
- fill_set(${mapVar}): Se usa para llenar una variable de tipo Set con nuevos elementos.

shareppy
FINANCIAL SERVICES

API de Formularios (formView)

Un api que nos permite crear formularios a nivel del cliente. La información del formulario es almacenada como un json.

Para crear un formulario utilizando la API de formularios es necesario tener las siguientes consideraciones en cuenta:

Crear el archivo en la siguiente ruta o paquete:

Dentro del proyecto, ubicar la carpeta views y crear el paquete view.cdn.shareppy.proyect, dentro de este paquete deben estar ubicados los archivos que utilicen el API de formularios.

Es necesario importar la librería de formularios, lo cual se hace escribiendo en la primera línea del documento la siguiente instrucción:

```
import shareppy.forms.tag as frm;
```

Para crear un formulario se utiliza la etiqueta siguiente:

```
frm:formView#id(atributos){}
```

La etiqueta frm:formView seguida de # y el id, El es el texto que se pondrá como valor en el atributo id de la etiqueta form en el html generado e identificará el formulario, dicho id debe corresponder al nombre del archivo, si el archivo tiene como nombre formTest.ui la etiqueta debe quedar de la siguiente forma:

```
frm:formView#formTest(los atributos){}.
```

Los atributos que pude contener un formulario son:

- default: Se indica el paquete donde se crea en formulario EJ. (default="_.shareppy.proyect") siempre debe comenzar por "_."
- layout: Indica el tipo de layout a utilizar, al momento de escribir este documento se encontraba implementado el layout "basic" EJ. (layout=basic).
- nowrap: Es un atributo de tipo boolean e indica si debe o no generarse un salto de línea si el contenido supera el ancho del formulario, generalmente se establece "false" sin comillas (nowrap=false).

shareppy
FINANCIAL SERVICES

Ejemplo del código para generar un formulario de vista básica:

```
import shareppy.forms.tag as frm;

frm:formView#loan(default="_.shareppy.core_banking_loan" layout=basic nowrap=false){


}
```

Secciones

Sección simple

Para crear una sección sencilla se usa section(""){}

Dentro de las comillas se escribe el título que contendrá la sección

Atributos:

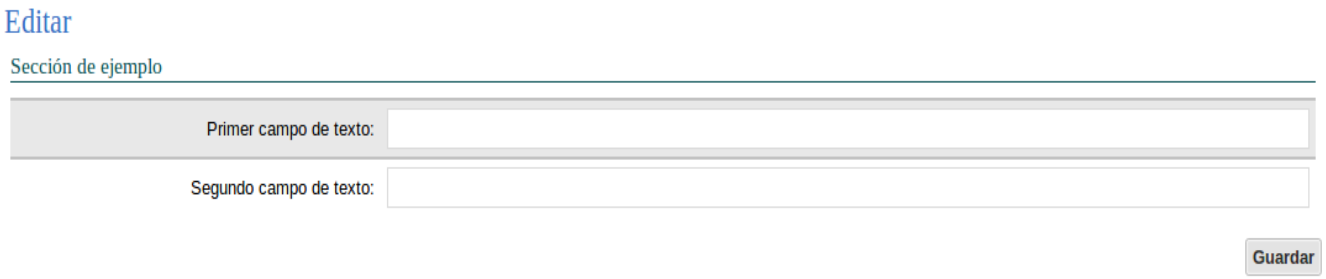
- " " Dentro de las comillas se escribe el título que mostrará la sección.
- i, el parámetro "i" recibe un valor numérico el cual indica el color de la primera fila de la sección, EJ. i=${1}, el 1 indica que se iniciara con el color definido para odd en el tema del sitio, si no se agrega este atributo, el color de la primera fila de la sección sera el color definido para even en el tema del sitio.

Código para crear una sección simple:

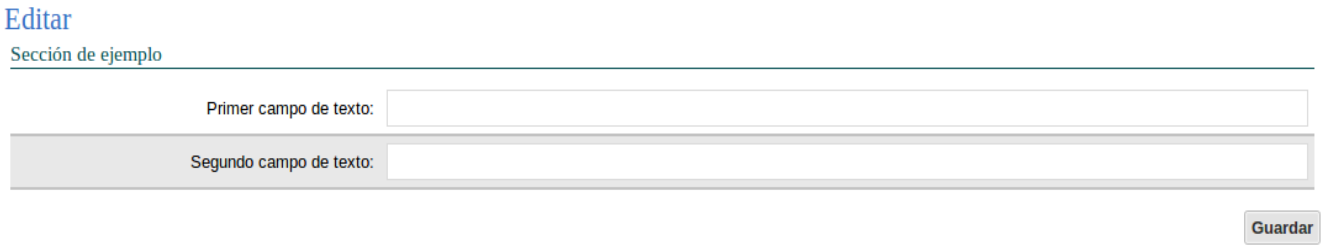
```
import shareppy.forms.tag as frm;
frm:formView#loan(default="_.shareppy.core_banking_loan" layout=basic
nowrap=false){
      section("Sección de ejemplo" i=1){
            text#firstFieldText("Primer campo de texto")
            text#secondFieldText("Segundo campo de texto")
      }
}
```

shareppy
FINANCIAL SERVICES

Ejemplo de una sección sencilla con i=1:



Ejemplo de una sección sencilla sin el parámetro i:



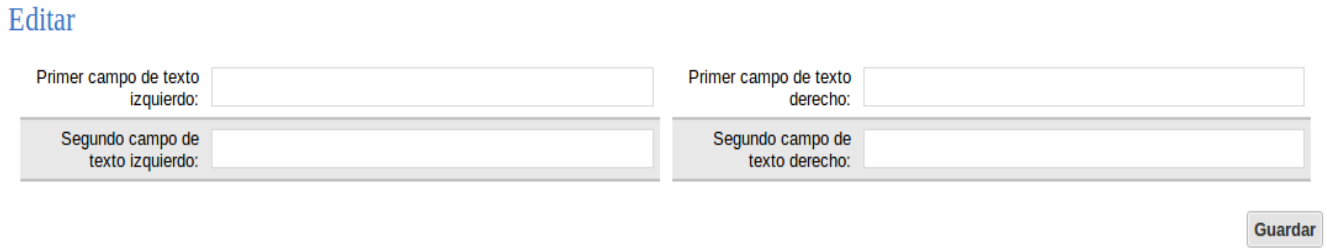
Sección en columnas

Para crear una sección de columnas se usa cols_section("" cols=${2}){}, dentro de esta se especifican las etiquetas col(""){}, las cuales indican las columnas dentro de la sección, si este atributo no existe, el valor por defecto es 2, para especificar más columnas se hace utilizando este atributo. Ejemplo de sección en columnas:

```
import shareppy.forms.tag as frm;
frm:formView#loan(default="_.shareppy.core_banking_loan" layout=basic
nowrap=false){
      section("Sección de ejemplo" i=1){
            col(""){
                  text#firstLeftFieldText("Primer campo de texto izquierdo")
                  text#secondLeftFieldText("Segundo campo de texto izquierdo")
            }col(""){
                  text#firstRightFieldText("Primer campo de texto derecho")
                  text#secondRigthFieldText("Segundo campo de texto derecho")
            }
      }
}
```

Dentro de la primera etiqueta col(""){} se encuentran o agregan los campos que se mostrarán en la columna izquierda de la sección, el orden descendente de las columnas corresponde al orden de izquierda a derecha en que se mostrarán las mismas.

Ejemplo de como se visualiza el código anterior sin el atributo i:



Secciones múltiples

Para definir secciones múltiples se usa la siguiente etiqueta:

sections("Título"){}

Dentro de las llaves {} de la etiqueta sections se crean todas las secciones que se necesiten en el formulario; existen dos tipos de secciones que se pueden implementar dentro de la etiqueta sections(""){}:

- Secciones simples.
- Secciones en columnas.

Datagrid

Para crear un datagrid se usa la siguiente etiqueta:

datagrid#id("Título"){}

Dentro de las llaves {} de la etiqueta datagrid se definen:

- header: Se definen las etiquetas de las columnas en el datagrid, como estas no reciben ningún valor, son tipo label.
- row: Se definen los campos en los cuales en usuario ingresará información, pueden ser de cualquier tipo, ésta opción recibe el atributo repeat, el cual define el número de filas que se van a mostrar por defecto en el datagrid.
- add_row: Se definen los campos que se mostrarán en la nueva fila luego de hacer clic en el vínculo "Agregar" , ésta opción recibe el texto del vínculo que se va a mostrar para agregar una nueva fila.

Ejemplo de la definición de un datagrid:

```
import shareppy.forms.tag as frm;
frm:formView#loan(default="_.shareppy.core_banking_loan" layout=basic
nowrap=false){
	datagrid("Valores de la comisión"){
		header{
			label("Desde" align=right)
			label("Hasta")
			label("Valor")
		}
		row(repeat=5){
			number#from
			number#to
			number#value
		}
		add_row(repeat=5){
			number#from
			number#to
			number#value
		}
	}
	buttons
}
```

shareppy
FINANCIAL SERVICES

Ejemplo de como se muestra un datagrid en el navegador:

Listas

Una lista es un objeto dentro del formulario que se usará para llenar las opciones de un combo, por ejemplo:

En HTML se crea un select de la siguiente forma:

```
<select>
      <option value="volvo">Volvo</option>
      <option value="saab">Saab</option>
      <option value="mercedes">Mercedes</option>
       <option value="audi">Audi</option>
</select>
```

Con el API de formularios de blueml se crea de la siguiente forma:

Se define la lista de opciones que contendrá el control se usa la etiqueta list#id{}, ejemplo de la creación de una lista:

```
      list#cars {
            "Volvo"("volvo")
            "Saab"("saab")
            "Mercedes"("mercedes")
            "Audi"("audi")
      }
```

Dentro de las llaves {} de la etiqueta list, se crean las opciones, las cuales mantienen la estructura "Texto a mostrar"("value"), estas opciones no se separan por "," ni por ";" son separadas por un salto de línea. Una vez creada la lista se procede a crear el objeto combo, el cual se encarga de mostrar una lista de desplegable en el formulario. Para crear una lista desplegable en el formulario se usa la etiqueta combo de la siguiente forma:

combo#id("Etiqueta a mostrar" data=la lista_creada value=${"Valor por defecto"} eventos=funcionEnJS)

El objeto combo deberá estar dentro de una sección, de lo contrario la generación del formulario fallará.

El atributo eventos solo se especifica si se requiere ejecutar alguna acción adicional al momento de seleccionar alguna de las opciones en la lista desplegable, la palabra eventos se reemplaza por el evento a ejecutar, los eventos soportados son los mismos que soportan los objetos HTML (onclick, onchange, onblur, etc.).

Ejemplo del código para crear una lista:

```
import shareppy.forms.tag as frm;
frm:formView#loan(default="_.shareppy.core_banking_loan" layout=basic
nowrap=false){
      list#cars{
            "Volvo"("volvo")
            "Saab"("saab")
            "Mercedes"("mercedes")
            "Audi"("audi")
      }
      section("Sección de ejemplo" i=1){
            combo#carOptions("Autos" data=cars)
} }
```

shareppy
FINANCIAL SERVICES

Ejemplo del combo generado en el formulario:

Toggles

En un .js se usa el toggle de la siguiente forma:

```
frm.toggle(['+idField1', '+idField2', '-idField3']);
```

El toggle recibe los id de los campos del formulario que se quieren mostrar o que se quieren ocultar .

Para usar los toggle en el formulario se usa la etiqueta toggle de la siguiente forma:

```
toggle#idToggle("-idField1,+idField2,...,-idFieldN)
```

Esta línea crea dos toggles uno que es "idToggle" que oculta y muestra ciertos campos, y una "no_idToggle" que hace lo contrario es decir los que oculta los muestra y los que muestra los oculta.

Para usarla en un combo, check o radio simplemente se agregan como hijos del control:

combo#idCombo("Sexo" data=sex_type r=true tabindex=11){

    toggle#idToggle("1")

    toggle#no_idToggle("")

}

Cada toggle recibe el id del toggle aplicar y un campo que es el valor del combo. El valor " " significa para todos los valores del combo que no tengan toggle asociado.

Sripting

Para ejecutar acciones de javaScript se hace necesario agregar eventos a los controles sobre los cuales se basan las acciones que se desean ejecutar.

Los eventos que se pueden agregar a los controles en el formulario son:

Manejador de evento	Objetos para los que está definido
onAbort	*Image*
onBlur	*Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, window*
onChange	*FileUpload, Select, Text, Textarea*
onClick	*Button, document, Checkbox, Link, Radio, Reset, Submit*
onDblClick	*document, Link*
onFocus	*Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, window*
onKeyDown	*document, Image, Link, Textarea*
onKeyPress	*document, Image, Link, Textarea*
onKeyUp	*document, Image, Link, Textarea*
onLoad	*Image, Layer, window*
onMouseDown	*Button, document, Link*
onMouseMove	Ninguno (debe asociarse a uno)
onMouseOut	*Layer, Link*
onMouseOver	*Layer, Link*
onMouseUp	*Button, document, Link*
onReset	*Form*
onSelect	*Text, Textarea*
onSubmit	*Form*

Las funciones a ejecutar por los eventos en los controles del formulario se deben declarar dentro de un archivo logic y deberán ser declarados de la siguiente forma:

load_logic("/shareppy/proyecto/forms/logic/archivo.js")

Las funciones dentro de este archivo se deben declarar de la siguiente forma:

```
_ext_.nombre_de_la_funcion = function(f) {
      ...
}
```

shareppy
FINANCIAL SERVICES

Calculates

Los campos calculates son campos en los cuales su valor depende del valor que tome otro campo u otros campos, Ejemplo:

Para convertir de kilos a gramos:

Se creará un campo tipo numérico para que se indique el peso en kilos, el campo calculate se llenará automáticamente con el valor en gramos.

Los campos calculates son campos que al cambiar el campo del cual dependen ejecutan una función javaScript previamente definida.

Para ejecutar las funciones con los campos calculates, el archivo javaScript donde se declaran las funciones se importa al formulario con la siguiente línea:

logic("/shareppy/proyecto/forms/logic/archivo.js")

La estructura de los campos calculates es la siguiente:

calculate#id("" b="campos" calc=funcion_a_llamar)

- b: se definen los id's de los campos de los que depende el campo calculate, si son varios id's se separan por "," Ej. b="id1, id2, id1"
- calc: se escribe el nombre de la función a llamar cuando se modifique uno de los campos puestos en el atributo b.

Ejemplo del código para crear un campo calculate que dependa de otro

```
import shareppy.forms.tag as frm;
frm:formView#loan(default="_.shareppy.core_banking_loan" layout=basic nowrap=false){
     section("Sección de ejemplo" i=1){
          number#kilos("Kilos" r=true)
          calculate#gramos("" b="kilos" calc=toGr)
     }
}
```

shareppy
FINANCIAL SERVICES

Ejemplo del código javaScript para calcular el valor del campo calculate

```
_ext_.toGr = function(e){
     $('gramos').value = parseInt($('kilos').value) * 1000;
}
```

Ejemplo del formulario con el campo calculate

Botones

Para crear los botones dentro de un formulario se usa la etiqueta buttons.

Ejemplo de la creación de botones dentro de un formulario:

```
import shareppy.forms.tag as frm;
frm:formView#loan(default="_.shareppy.core_banking_loan" layout=basic nowrap=false){
     section("Sección de ejemplo" i=1){
          number#kilos("Kilos" r=true)
     }
     buttons(style="text-align:right;"){
          "Mostrar gramos"#showGr
     }
}
```

Ejemplo del código javaScript para la función del botón:

```
_ext_.showGr = function(e){
     alert(parseInt($('kilos').value) * 1000);
}
```

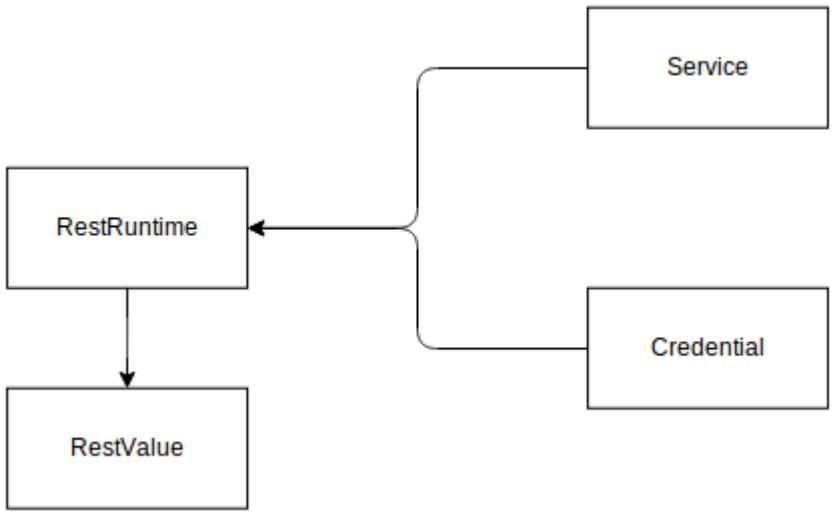
NOTA: Cabe aclarar que el id que se establece al botón deberá ser el nombre de la función, para el ejemplo anterior se uso como id showGr y la función se llamo _ext_.showGr, se usa _ext_. por sintaxis de la plataforma.

shareppy
FINANCIAL SERVICES

Ejemplo del botón creado:

Shareppy Api Rest

A través de esta API podemos consumir los servicios expuesto por la Plataforma Shareppy.



shareppy
FINANCIAL SERVICES

Creación del RestRuntime

El RestRuntime es la clase encargada de consumir los servicios Rest que la plataforma expone.

```
final RestRuntime runtime = new RestRuntime("http://url_server.com");
```

Esta instancia es ThreadSafe y puede usarse una sola por servidor en diferentes Hilos.

Autenticación

La plataforma puede soportar diferentes modelos de autenticación. El más común es con usuario y contraseña:

```
final Login login = new Login("usuario", "contraseña"); login.login(runtime);
```

Consumir un servicio

Una vez se ha hecho login se puede consumir servicios mediante crear instancias del tipo Service.

```
final Service service = new Service("endpoint");
//Agrega atributos al servicio
service.add("attr1", "value1");
//Se puede usar get o post
final RestResult result = runtime.get(login, service);
```

El runtime soporta dos tipos de comandos REST: get y post. La respuesta a la invocación es un objeto del Tipo RestValue.

Usando la respuesta de un servicio

El objeto RestValue tiene dos métodos con los que podemos manipular la respuesta:

1. getResult -> Retorna un objeto que representa el valor
2. getHeader -> Obtener el valor de alguna Cabecera de respuesta. Las cabeceras sirven como metadata a la petición.

Una vez se haya usado la respuesta se debe invocar el método release para liberar los recursos consumidos

```
final RestResult result = runtime.get(login, service);
try{
    ...
}finally{
   result.release();
}
```

Cliente de servicios web Rest

Existe un API para el consumo de servicios web REST en la plataforma.

Cliente para consumo de servicios de la plataforma Shareppy

En la plataforma Shareppy se ha decidido el uso de servicios web basados en tecnología REST. Existen dos situaciones desde donde será necesario realizar el consumo de un servicio web de la plataforma, estos son: Desde una aplicación basada en tecnología Shareppy y desde una aplicación externa.

Consumo de un servicio web desde una aplicación basada en tecnología shareppy

En el desarrollo de una aplicación usando la tecnología shareppy existen también dos situaciones desde donde se puede consumir un servicio, estos son: desde un proyecto web o desde un proyecto de formulas. En el presente artículo nos enfocaremos en el consumo desde un proyecto web, para profundizar respecto al consumo desde un proyecto de formulas véase el siguiente artículo.

Para realizar el consumo de un servicio web desde una aplicación web, es un prerrequisito adicionar la dependencia del proyecto shareppy.security@hmac en el ivy del proyecto correspondiente:

```
<dependency transitive="true" conf="webplugin->webplugin" rev="latest.integration"
name="hmac" org="shareppy.security"/>
```

Una vez hecho esto, procedemos a importar en el servicio correspondiente la librería de consumo.

```
	import shareppy.security.hmac as hmac;
```

Posteriormente se incluirán las líneas de invocación al servicio web:

```
	hmac:call_esb_service#RESULT(config="ESBConfig"
	endpoint="/shareppy/tx_validator.AS400/execute/ID_SERVICIO"){
		"PARAM1"(PARAM1VALUE)
		"PARAM2"(PARAM2VALUE)
	}
```

Posteriormente a esta invocación se podrá usar la variable RESULT para obtener los valores de respuesta o con la instrucción RESULT.getResult() se obtendrá un mapa con todas las propiedades del resultado.

Consumo de un servicio web desde una aplicación externa

Para consumir un servicio web de la plataforma shareppy desde una aplicación externa es necesario obtener las librerías necesarias.

A continuación se listan las librerías necesarias para realizar el consumo en una aplicación Java, (las versiones pueden variar, en el momento de escribir este artículo las indicadas son las más recientes).

- commons-codec.commons-codec-1.9.jar
- commons-logging.commons-logging-1.2.jar
- hmac.client-1.0.jar
- junit.junit-4.12.jar
- org.apache.httpcomponents.httpclient-4.5.jar
- org.apache.httpcomponents.httpcore-4.4.1.jar
- org.hamcrest.hamcrest-core-1.3.jar
- org.slf4j.slf4j-api-1.7.12.jar
- org.slf4j.slf4j-jdk14-1.7.12.jar
- shareppy.kernel-1.0.jarshareppy.sppy-net-1.0.jar

Debe agregar estas librerías al classpath de su proyecto para poder reverenciarlas desde su proyecto.

Posteriormente podrá usar las clases requeridas para la creación del cliente, como se explica a continuación:

Para la creación de un cliente de consumo a webservices Shareppy debe solicitar al administrador del sistema los siguientes datos:

- Endpoint donde se expone el servicio.
- Usuario.
- Clave.
- Identificador del servicio que desea ejecutar.
- Definición de los parámetros que recibe el servicio.

Para este caso, vamos a tomar de ejemplo los siguientes datos:

- Endpoint: http://192.168.34.140:8081/sucursalnbta
- Usuario: FC188
- Clave:wJWmugrrdoTR/RIP0KBcjw38GinQ9jdT7GzaKKZ4yCGNa/fl7hPN2oDlrpUcKWxqYK2Ks88y8wr21tB7vahNJg==
- Identificador de servicio: 28dc472b-ce1f-4a84-9916-762910de37fa
- Parámetros: Numero Documento=91266853

Con estos parámetros se procederá a realizar el consumo como se explica a continuación:

```
final RestCallConfig config = new RestCallConfig();
config.setHost("http://192.168.34.140:8081/sucursalnbta");
config.setUser("FC188");
config.setKey("wJWmugrrdoTR/RIP0KBcjw38GinQ9jdT7GzaKKZ4yCGNa/fl7hPN2oDlrpUcKWxq
YK2Ks88y8wr21tB7vahNJg==");  final RestCall call = RestCall.get(config,
"/shareppy/tx_validator.AS400/execute/28dc472b-ce1f-4a84-9916-762910de37fa");
call.add("Numero Documento", "91266853"); call.run();
System.out.println(call.getResult());
```

Al ejecutar este ejemplo obtendremos como salida:

{Nro Cuenta III=000000000000000000, Producto=06, Canal=08, Transferencia Electronica=0, Valor Movimiento V=000000000000000, Saldo Cuenta I=000000000000000, Tipo Cuenta II=000, Codigo Facturador=000, Valor Movimiento IV=000000000000000, Saldo Cuenta V=000000000000000, Tipo Cuenta Destino=000, Tipo Cuenta IV=000, Codigo Transaccion=002, Nro Factura=000000000000000000, Valor Movimiento II=000000000000000, Cuenta Destino=000000000000000000, Error Mensaje=Valor Efectivo Supera Limite, Cuenta Origen=000000000000000000, Nro Cuenta I=000000000000000000, Track II=00000000000000000000000000000000000000, Hora Transaccion=15120907, Nro Cuenta V=000000000000000000, Error Codigo=0809, Categoria Comercio=0000, Valor Movimiento I=000000000000000, Tipo Cuenta III=000, Voz=0, Fecha Movimiento V=00000000, Convenio=0, Comprobante=000000, Nro Cuenta II=000000000000000000, Codigo ISO=0200, Tipo Cuenta I=000, Cantidad Productos=00000, Referencia=000000000001, Numero Documento=000000000091266853, Fecha Movimiento I=00000000, Saldo Cuenta III=000000000000000, Referencia Movimiento IV=000000000000000000, Valor Transaccion=000000000000000, Saldo Cuenta IV=000000000000000, Nombre Cliente=, Codigo Proceso=000000, Valor Movimiento III=000000000000000, Nro Cuenta IV=000000000000000000, Saldo Cuenta II=000000000000000, Fecha Vencimiento=0000, Fecha Movimiento II=00000000, Fecha Movimiento III=00000000, Referencia Movimiento V=000000000000000000, Tipo Cuenta Origen=000, Numero Tarjeta=000000000000000000, Cajero=CAEXT05501, Terminal=9999999999, Red=1111, Referencia Movimiento I=000000000000000000, Referencia Movimiento III=000000000000000000, Pin=0000, Valor Cheque=000000000000000, Referencia Movimiento II=000000000000000000, Fecha Movimiento IV=00000000, Sin Uso=0000000000000000000, Fecha Transaccion=20151118, Valor Efectivo=000000000000000, Valor Nota Debito=000000000000000, Tipo Cuenta V=000}

Como se puede ver en este ejemplo al usar el método getResult() se obtiene un mapa con todas las propiedades de retorno del servicio, sin embargo si solo desea obtener algunos valores específicos puede usar el método get("CLAVE").

```
call.get("Error Mensaje")
```

Y se obtendrá la respuesta:

Valor Efectivo Supera Limite.

Servicios (service)

Los servicios son objetos de la capa web, que permiten realizar la conexión entre la capa cliente y la capa de lógica de negocio, lo podríamos ver como un Servlet de la especificación JEE, que permite acceder a los datos de una petición web y preparar una respuesta de vuelta al cliente.

Scripting del lado del cliente (jsView)

A través de estas vistas se puede agregar funcionalidad en el lado del cliente. Permitiendo crear interfaces de usuario.

Esta API permite escribir código javascript mediante código blueml. Hay tres formas de usar esta API:

1. js_code: Se usa dentro de una vista HTML.
2. jsView: Se usa como un documento independiente de scripting siempre se crea en un entorno de variables privado.
3. jsFragment: Se usa como un documento independiente para ser incrustado en un jsView.

Notas básicas

1. Esta API soporta código Blue y Javascript.
2. Los bucles y condicionales en Javascript empieza con una js:: js:if, js:while, js:for
3. Declaración de variables en javascript: js:var

4. Para incrustar código js en la API se usa %{...}
5. La API soporta string expansión mediante el uso de %{código js} y ${código blue}

Vistas JS

Para crear una vista js se utiliza jsView o jsFragment.

jsView

jsView es una vista Javascript en un espacio de nombres para crearla se define

```
jsView("espacio de nombres"){
   ....
}
```

jsView usa el patrón de espacio de nombres. La estructura que genera será:

```
(function(){
    var pck = _.path("espacio de nombres", true);
    ...
})()
```

Para poder acceder variables y funciones en otro script sea (blueml o javascript) se debe usar el tag export:

```
export#variable(%{variable})
```

El valor debe ir en %{} porque sólo se puede enlazar código js.

Lenguaje

Definiendo variables Javascript

Para definir variables que son propias de javascript es decir no pueden acceder se en el servidor sino en el lado del cliente se usa el tag jvar.

El valor de la variable deberá definirse usando %{...} de lo contrario será interpretado como un string.

```
js:var#year_days(%{365}) { ... }
```

Los hijos dentro de la variable se usan como valores por defecto. si el valor del atributo es null el usara el valor de su primer hijo, si este es null usara el del siguiente hijo y así:

```
var year = 365 || ... || ...
```

También es posible definir varias variables en bloque mediante el tag declare

```
declare {
   year(%{2013})
   month(%{2})
   ...
}
```

Definiendo funciones Javascript

Para definir funciones se deberá usar el tag func:

```
func#sum("a, b"){
   ....
}
```

En el ejemplo se declara una función llamada sum y recibe dos argumentos a y b. Para retornar valores usamos el tag return:

```
return(...)
```

Ejecutando funciones Javascript

Para ejecutar alguna función se usa e tag call. El atributo default es la función a invocar Los hijos del tag son los argumentos de la función:

```
call(%{func} this=%{...}) {
  ....
}
```

Para definir el contexto de la función se usa el atributo this. Si no se especifica this será el objeto global (*window en los navegadores*).

Condicionales If

El if se usa mediante el tag js:if. dentro del if es posible usar elif y else. Dentro del elif no se puede usar ni elif y ni else:

```
if(%{..}){
   ....
   elif(%{..}){
     ....
   }
   elif(%{...}){
     ....
   }
   else {
     .....
   }
}
```

Para los bucles for y while se pueden usar los tags break y continue.

Condicionales Each_array

Con el tag each_array crea un bucle para recorrer cada elemento del array iniciando en el primer elemento o en el sub elemento.

```
each_array#childId(%{array} i=1){
   ....
}
```

El valor por defecto del atributo i es 0.

Soporte JSON

Para crear objetos JSON se usan dos tags: object y array.

```
js:var#id > object (clave="valor", ...) {
 clave(valor)
 clave(valor)
 clave(valor) {
    valor
    valor
    valor
 }
}
```

Los valores del objeto se pueden pasar como hijos del objeto o como atributos. Los atributos se procesarán primero que los hijos. Si una clave tiene varios hijos el valor de la clave será el primer valor que no sea null o cero o vacío.

```
js:var#id > array { valor, valor, valor }
```

valor puede ser un valor blue, js, func, object o array.

El atributo default del object y el array especifica un objeto ya existente sobre el cual se desea actualizar propiedades.

El atributo i de array especifica a partir de que elemento empezar agregar los nuevos elementos y el atributo d especifica cuantos elementos eliminar antes de empezar agregar. Estos dos atributos solo se pueden usar con default

Timeout

Timeout permite ejecutar un código después de cierto tiempo:

```
timeout(%{...}) {
   ...
}
```

El atributo por defecto del tag define el tiempo que deberá esperar para poder ejecutar el timeout. Es posible volver a invocar el timeout a través del tagrepeat_timeout(%{...}):

```
timeout(%{...}) {
   ...
   repeat_timeout(%{...})
}
```

El tiempo del delay deberá especificarse en milisegundos.

Operaciones Ajax

Para hacer operaciones ajax se usan los tags get y post.

```
get("url", qs=%{qs} ...){
   data#id
   ...
}

post("url", qs=%{qs} ...){
   data#id
   ...
}
```

El atributo por defecto es la url del servicio, el querystring se construye con los atributos del tag. Cuando la petición responda se invoca el contenido del tag. Es posible pasar un objeto como querystring mediante el atributo qs. Los atributos adicionales se agregarán a ese objeto adicional.

A través del tag data se obtiene referencia a la variable de la función ajax.

Si se usa el atributo async="false" la operación será sincrona.

Interacción con el DOM

Crear Dom

Para crear dom se usa el tag dom_builder dentro del tag se usa los tags de html.

```
js:var#dom1 > dom_builder (%{...} position="") {
   div > a#item(href="#" style="color:red")
}
```

Este código crea un div con un hipervínculo dentro. El retorno es un objeto json con esta forma:

```
{
   'dom':  <Un documentFragment con el dom generado>,
   'item': 'El hipervínculo creado'
}
```

El dom_builder soporta el atributo por defecto es un nodo sobre el cual se desea agregar el hijo. position es la posición con respecto al nodo valor por defecto last. Valores soportados: first, last, before, after.

Es posible anidar elementos dom mediante el uso del tag ref:

```
js:var#dom1 > dom_builder {
   div > a#item(href="#" style="color:red")
}
```

```
js:var#dom2 > dom_builder {
   ref(%{dom1.dom})
   div > a#item(href="#" style="color:red")
}
```

shareppy
FINANCIAL SERVICES

Los atributos que empiecen con css_* representan propiedades de estilo.

Los atributos que empiecen con ds_* representan propiedades del datastore.

Propiedades del DOM

- %id{id} -> Retorna un nodo dom buscado por id.
- %f{frm, 'elemento'} -> Retorna un elemento de un formulario.
- %ṕ{dom, 'clase'} -> Busca un padre que tenga una clase.
- %t{dom, 'transversal'} -> Aplica transversal sobre el dom y retorna el hijo.
- %ds{dom, 'clave', [valor]} -> retorna un valor guardado en el dom.
- %a{dom, 'clave', …} -> retorna el valor de un atributo del dom o configura los atributos de un elemento.
- %class{dom, 'clase', …} -> agrega clases al dom si la clase empieza con - la elimina.
- %wclass{dom, 'clase', …} -> retorna true si el dom tiene alguna de las clases.
- %pos{dom,[true|false]} -> obtiene la posición del control con el scroll de la vista. Si se envía el valor de true la posición será global.
- %dim{dom} -> Obtiene la dimensión del control.

estas propiedades también se pueden usar dentro del %{...} mediante usar @:

- @id(id) -> Retorna un nodo dom buscado por id
- @f(frm, 'elemento') -> Retorna un elemento de un formulario
- @p(dom, 'clase') -> Busca un padre que tenga una clase.
- @t(dom, 'transversal') -> Aplica transversal sobre el dom y retorna el hijo
- @ds(dom, 'clave', [valor]) -> retorna un valor guardado en el dom
- @a(dom, 'clave', …) -> retorna el valor de un atributo del dom o configura los atributos de un elemento.
- @class(dom, 'clase', …) -> agrega clases al dom si la clase empieza con - la elimina
- @wclass(dom, 'clase', …) -> retorna true si el dom tiene alguna de las clases
- @pos(dom, [true|false]) -> obtiene la posición del control con el scroll de la vista. Si se envía el valor de true la posición será global.
- @dim(dom) -> Obtiene la dimensión del control.

shareppy
FINANCIAL SERVICES

Modificar los atributos de un Nodo Dom

A través del tag dom_attrs es posible agregar atributos a un nodo:

```
dom_attrs(%{node} attr=value ...) {
   attr(value) > ...
}
```

Los atributos que empiecen con css_* representan propiedades de estilo.

Eventos en Dom

A través del tag filter_event es posible crear un patrón de condicionales:

```
filter_event#target(%{...}){
   pattern {
      ....
   }
   pattern {
      ....
   }
}
```

El atributo default es la referencia al objeto event. El atributo id es el nombre que tendrá la variable que hace referencia al target.

Los hijos del tag definen los patrones que se desean manejar.

Patrones del filter_event

- has_class(...) -> Se activa si el target tiene la clase que se define en el atributo default.
- has_id(...) -> Se activa si el target tiene el id definido.
- is_tag(...) -> Se activa si el target tiene el tag definido. El nombre del tag debe estar en minúsculas.
- if(%{...}) -> Si cumple una condición.
- has_attr(...) -> Si tiene un atributo attr.

shareppy
FINANCIAL SERVICES

Drag And Drop

Esta versión de drag and drop no funciona en móviles. Es recomendado que exista un mecanismo alterno al drag.

Para hacer drag and drop los elementos DOM que se van a poder mover deberán tener el atributo draggable=true. El tag setup_drag se usa para configurar drag and drop.

```
setup_drag#item(%{container}) {
   ...
}
```

El id del tag es la referencia al elemento que se esta arrastrando. El atributo por defecto es el nodo que contiene los elementos que se pueden arrastrar.

on_start

Este callback se activa cuando un elemento dom se va a mover.

```
setup_drag#item(%{container}) {
   on_start("...", css_opacity="40%"){
      ....
   }
   ...
}
```

Los atributos que recibe el tag on_start manipulan los atributos del elemento que se ha de mover.

on_enter

Este callback se invoca cuando el elemento entra en el área de un elemento que permite drop

```
setup_drag#item(%{container}) {
   on_enter#id("class" css_opacity="40%"){
      ....
   }
   ...
}
```

Los atributos que recibe el tag on_enter manipulan los atributos del elemento que soporta drop. El id del tag representa el elemento sobre el cual se entró

on_leave

Este callback se invoca cuando el elemento sale del área de un elemento que permite drop

```
setup_drag#item(%{container}) {
   on_leave#id("class" css_opacity="40%"){
      ....
   }
   ...
}
```

Los atributos que recibe el tag on_enter manipulan los atributos del elemento que soporta drop. El id del tag representa el elemento sobre el cual se entró

on_over

Este callback se invoca cuando el elemento es soltado sobre un elemento que no soporta drop.

```
setup_drag#item(%{container}) {
   on_over{
      ....
   }
   ...
}
```

on_end

Este callback se invoca cuando el elemento es soltado sobre un elemento que no soporta drop.

```
setup_drag#item(%{container}) {
   on_end{
      ....
   }
   ...
}
```

shareppy
FINANCIAL SERVICES

Overlays

- El tag show_save para mostrar la opción de guardar.
- El tag show_load para mostrar la opción de cargar.
- El tag hide_overlay se usa para ocultar el overlay que se ve actualmente. Este tag soporta hijos que serán invocados una vez el overlay se halla ocultado.

Show Overlay

El tag show_overlay para mostrar algún overlay. Este tag recibe los siguientes atributos:

- w -> Opcional. Ancho
- h -> Opcional. Alto
- id -> El id del elemento dom a mostrar debe cargarse en la vista HTML
- default -> define los nombres de los argumentos que relacionan la ventana modal y el tamaño.

```
show_overlay#ventana("overlay,size" w=200, h=300) {
    log(%{win})
    ...
}
```

El tag show_overlay soporta hijos que se invocan cuando el overlay ha cargado.

save_overlay

Use el tag save overlay para guardar un overlay y mostrar otro, de manera que cuando se cierre el último abierto vuelva a mostrar el anterior.

shareppy
FINANCIAL SERVICES

Load Overlay

- El tag load_overlay para mostrar un overlay cuyo vista puede ser ajax o una dom creada en BlueJS. Los atributos:
- default -> Opcional. La url ajax.
- W -> Ancho
- h -> Alto
- qs -> EL querystring para la petición ajax
- id -> El id del contenedor del overlay
- data -> Este tag define información para el overlay
- on_click -> Este tag se invocará cuando se de click sobre algún elemento del dom cargado. El id del tag ese el nombre de la variable que representa el evento
- on_show -> Este tag se invocará cuando el overlay se ha mostrado. Puede que aun no se haya cargado la url ajax.
- on_load -> Este es un tag para definir código cuando la petición ajax fue cargada en el overlay
- ui -> define la interfaz de usuario del overlay.

```
load_overlay#container("url" w=200, h=300, qs="", ...) {
      data#dt_id > ... //Aqui sólo se espera un valor Object, array o una
función
      ui {
            //La variable container es donde se deberá asociar el ui
            // Aquí es posible obtener el data
            ....
      }
      on_load {
            //Aqui es posible acceder la variable container
            // Aqui es posible obtener el data
            ....
      }
```

```
      on_click#eventId {
            //Aqui lo recomendado es usa %{eventId.target}
            // Es posible acceder la variable container
            // Aquí es posible obtener el data
            ....
      }
      on_show("overlay, size") {
            //Aqui es posible acceder el container
            // Aqui es posible obtener el data
            ...
      }
}
```

Los atributos adicionales del tag load_overlay serán interpretados como querystring.

Alert

Para mostrar una ventana de alerta en Javascript se usa el tag alert:

```
alert("msg" title="" no="") {
      ...
}
```

Si se quiere que la ventana tenga titulo usar el atributo title. El atributo no define el label del botón.

Confirm

Para mostrar una ventana de configuración usar el tag confirm:

```
confirm("msg" title="" yes="" no="") {
...
}
```

Si se quiere que la ventana tenga titulo usar el atributo title. Los atributos yes y no definen el label de los botones.

shareppy
FINANCIAL SERVICES

Quickviews

- Para ocultar un quickview usamos el tag hide_quickview.

Para mostrar un quickview se usa el tag quickview. El atributo por defecto será una URL o un Id a un elemento dom. Para que sea una url el atributo ajax=true.

El atributo bind enlaza el quickview a un elemento dom. Si se pide otra vez el quickview con ese bind el usara el que se mostró anteriormente.

Las coordenadas del quickview se pueden fijar por el atributo position(un objeto con las propiedades x e y. También se puede usar %pos{dom}) ó por los atributos x e y.

El atributo into especifica el elemento dom sobre el cual se agregará el contenido cargado por ajax.

```
quickview(%{..} lazy=%{dom} position=... into=%{...})
```

También es posible crear el dom del quickview dentro del tag quickview. Siempre se deberá retornar el dom que lo representa.

```
quickview(position=... into=%{...}){
      ....
      return(%{dom})
}
```

Templates

Para usar un template dentro de un javascript usamos el tag use_template, el id del tag representa la variable donde se almacenaran los ids, el atributo por defecto es la url de los templates a cargar y el contenido del tag se invoca cuando los templates ya han cargado.

```
use_template#tpls(@cdn{/shareppy/social/comments}) {
      dom_attrs(%{attachUi} html=%{tpls.att_file} onclick=%{divUploadEvent})
      ...
}
```

Los templates son texto por lo que la mayoría de las veces se usa dom_attrs y la propiedad html.

shareppy
FINANCIAL SERVICES

Stack UI

El api de stack ui esta compuesta de dos elementos el registro de componentes visuales y el pintado del stackui dentro de un control.

Registro de elementos dentro stackui (stackui_widget)

Para configurar el stackui se usa el tag stackui_widget:

```
stackui_register#type("<formulario>"){
      def_attrs(campo=valor campo=valor){
            ...
      }
      paint#container {
      }
      onload{
      }
}
```

Cuando se usa el tag paint, el id representa la variable del contenedor y hay una variable por defecto attrs que representa los atributos del elemento que se desea pintar.

Mostrar un stackui (stackui)

Usando el tag stackui mostramos un stackui dentro de un componente DOM:

```
stackui(%{dom} data=%{dataToPaint}) {
      def_attrs(type=<Tipo por defecto>) {
      }
}
```

shareppy
FINANCIAL SERVICES

Formulas BLUE

Blue es un lenguaje de programación, extendido de java el cual maneja concepto avanzado de programación como es el uso de paralelismo y ejecuciones automáticas de procesos de cargue de datos entre otras características.

API Actividades

Blueml_ActivityAPI

La interfaz Topics define la forma en que se muestran las actividades en el navegador.

Topics define los filtros, botones, imágenes y campos de texto que se mostrarán e la pantalla donde se listan las actividades.

Para utilizar la interfaz topics, se define un objeto que contendrá las siguientes funciones:

- topic
- comment
- editor
- page
- list
- button_click
- att_icon
- att_action

Topic

En la función topic se define la interfaz de las actividades, el como se muestra cada tipo de actividad en pantalla.

La interfaz visual depende de cada tipo de tópico, los botones y el número de botones se muestran dependiendo de los tipos de tópico.

shareppy
FINANCIAL SERVICES

Comment

En la función comment se definen los links que se mostraran para agregar comentarios, se permiten 0 u mas links, el número de links depende de las necesidades de quien define la interfaz del tópico.

Page

En la función page se define la configuración de la página, se definen los filtros que se van a usar para mostrar los datos, los separadores, el titulo de la página, campos de texto y pie de página.

La definición de estos atributos de la página se pueden realizar dependiendo del tipo de tópico que se vaya a mostrar.

List

En la función list se define la URL que traerá los datos a mostrar de todos los temas, estos se mostraran en la parte central de la pantalla, en medio de los controles definidos en page, se mostrarán como se configuraron en topic y editor.

```
list : function(json) {
  return _app_root + '/shareppy/Topics/list_' + json.type;
},
```

Button_click

Se implementa la lógica para asignar las funciones que se ejecutarán al hacer click sobre algún botón dentro de los tópicos que se muestran en el formulario creado.

Att_icon

Se agrega la url a la imagen que se mostrará con los archivos que se adjunten.

Att_action

Se agrega la función o acción que se ejecutará al adjuntar un archivo.

shareppy
FINANCIAL SERVICES

Ohana Apps

Ohana es un framework para crear apps para dispositivos móviles Android, iOS, Windows Phone, BlackBerry.

Conceptos

- Vista: Es una pantalla dentro de la aplicación.
- SuperLayout: Es el método con el que se define la posición de los controles en las vistas.
- Styles: Define la apariencia de una pantalla.

Vistas

Las vistas son pantallas dentro de la aplicación. Se definen en archivos .ui:

```
view(main=true) {
      ...
}
```

El atributo main indica si esta vista es la vista con la que se ingresa a la aplicación. Los componentes dentro de la vista se distribuyen usando el concepto de SuperLayout.

SuperLayout

Para posicionar un elemento dentro de una vista se usa el SuperLayout. Este layout utiliza cuatro propiedades dentro de los controles: x, y, w, h

```
view(main=true) {
      button#login("Entrar"
      y="#ry{password} + 20"
      x="(100% - @w)/2"
      w="@prefw + 30"
      h="@prefh + 5" )
}
```

shareppy
FINANCIAL SERVICES

Cada elemento de posición (x, y) y tamaño (w, h) permite operaciones matemáticas básicas (suma, resta, multiplicación y división) y puede usar la posición de otro componente dentro de la vista. O el tamaño total de la vista. Adicionalmente es posible usar % de la pantalla dentro del calculo. Las propiedades especiales son:

- @prefw: El ancho preferido del control
- @prefh: El alto preferido del control
- @w: El ancho del control (puede ser preferido o manual)
- @h: El alto del control (puede ser igual al preferido en caso de que no se haya colocado un valor manual)
- #ry{controlA}: La coordenada y + alto del control A
- #rx{controlA}: La coordenada x + ancho del control A
- #y{controlA}: La coordenada y del control A
- #x{controlA}: La coordenada x del control A
- #h{controlA}: La altura del control A
- #w{controlA}: El ancho del control A
- #prefh{controlA}: La altura preferida del control A
- #prefw{controlA}: El ancho preferido del control A

El proceso de layout primero dimensiona los elementos y luego los posiciona. Eso significa que en las propiedades x e y es posible usar todas las propiedades especiales, mientras que en las propiedades w y h sólo se pueden usar de dimensión.

Es posible usar los atributos rápidos dim y pos para dimensionar y posicionar un control. Dim configura las coordenadas w y h y permite los siguientes valores:

- equal#controlA: Hace el control del mismo tamaño que el control A.

Pos permite configurar más valores:

- next#controlA: Coloca un control al lado del control A
- equal#controlA: Coloca un control en las mismas coordenadas del control A
- below#controlA: Coloca un control debajo del control A