Provided directives, filters, services and model values

Servoy default components in ng-client provide functionality for it's properties via directives applied to the component template.It is known that some properties are design time only properties and some properties are also runtime properties and thus the implementation of a design time property is different than a runtime property. The design time directive only applies the property once at initialization and it's job is done .The runtime directives 'watches' (via angular $watch()) any property change coming from the server and updates the component.

These directives were created with webclient behavior in mind and implementation can change since servoy 8 is alpha.

The directives provided are:

svy-autoapply Is the main way of 'pushing' dataprovider updates (for **'dataprovider' typed properties** defined in the web component's model) from browser to server (to the record or global/form var ...). It requires angular's **'ng-model'** directive (*value being the model dataprovider property*) to be used on the same input element. It's purpose is to listen for any change (html 'change' event) from the input element to which it is applied as attribute and to send the new value to the server record/dataprovider after setting it in the web component's model as well (client-side). It also monitors focus and triggers startEditing on the server when the input receives focus. Behind the scenes it uses servoyApi provided to the web component by Servoy in svyServoyapi. See also apply and startEdit.

> **IMPORTANT**
>
> In order for svy-autoapply to work properly, the .spec file **must** declare that dataprovider property from the model as **pushToServer: allow** or higher. Otherwise the server will reject the dataprovider value updates and log a change denied warning.

For example:

```
<my-component name="mySpecialTextfield" svy-model="model.myDataproviderProperty" svy-autoapply (...)/>
```

svy-decimal-key-converter is a directive that can be used in (text) input fields to convert the numeric pad decimal key to the character the locale of the user defines. So for a US (international) keyboard that displays a . on the key will input a , for the dutch locale (the decimal delimiter is a , in the dutch locale). This is used by the default servoy textfield.

svy-enter (Design, domevent:'keydown') this directive triggers the given function when the enter key is pressed in a text input field. It does supply an $event parameter.

```
svy-enter='handlers.onActionMethodID($event)'
```

svy-click (Design, domevent:'click') this directive triggers the given function when a mouse click is done. It does supply an $event parameter.

```
svy-click='handlers.onActionMethodID($event)'
```

> **IMPORTANT**
>
> Note that all svy-* handlers will be executed asynchronous because of Servoy order of events. This may affect some operations inside handler (for example $event.preventDefault() may not work)

svy-dblclick (Design, domevent:'dblclick') this directive triggers the given function when a double click happens, It does supply an $event parameter.

```
svy-dblclick='handlers.onDoubleClickMethodID($event)'
```

svy-rightclick (Design, domevent:'contextmenu') this directive triggers the given function when a right click happens. It does supply an $event parameter.

```
svy-rightclick='handlers.onRightClickMethodID($event)'
```

svy-focusgained (Design, domevent:'focus') this directive triggers the given function when the element receives focus. It does supply an $event parameter.

```
svy-focusgained='handlers.onFocusGainedMethodID($event)'
```

svy-focuslost (Design, domevent:'blur') this directive triggers the given function when the element looses focus. It does supply an $event parameter.

```
svy-focuslost='handlers.onFocusLostMethodID($event)'
```

svy-background (Runtime | accepts 'color' property type from spec as the model attribute value) sets the **'background-color'** style of the element being applied to.It also watches the property values and updates the compoent if the compoent is updated on the server (being a runtume) . Background color is dependend on the trasparency flag in servoy and it only applies the background color if the trasparency flag is not set .It's usage in ngClient components is:

```
svy-background='model.transparent ? "transparent" : model.background'
```

svy-foreground (Runtime | color) - same as svy-background but for **'color'** css property and independent of trasparent property.

```
svy-foreground='model.foreground'
```

svy-scrollbars (Design | scrollbars) - sets provided scrollbars on the html element being applied to

```
svy-scrollbars='model.scrollbars'
```

svy-font (Runtime | font) - sets provided font on the html element being applied to

```
      svy-font='model.fontType'
```

svy-margin(Design | dimension) - sets provided magin **as css padding** on the html element being applied to

```
svy-margin='model.margin'
```

svy-border (Runtime | border) - sets provided border css on the html element being applied to

```
svy-border='model.borderType'
```

svy-tooltip (Runtime | tagstring) - sets the tooltip text (TODO change to $watch() because of tagstring)

```
svy-tooltip='model.toolTipText'
```

svy-textrotation(Design | {type:'int', values:[0,90,180,270]}) applies css 'transform:rotate(<X>deg);' to the element and swaps height and width. So it rotates only the contents of the element

```
svy-textrotation ='model.textRotation'
```

svy-selectonenter(Design | boolean) - if true will select text when field gets focus

```
svy-selectonenter ='model.selectOnEnter'
```

svy-rollovercursor(Design | int) - value 12 means hand rollover cursor (css pointer)

```
svy-rollovercursor='model.rolloverCursor'
```

svy-imagemediaid(Runtime | media)

```
svy-imagemediaid='model.imageMediaID'
```

svy-mnemonic(Design | string) - sets the 'accesskey' html attribute value

```
svy-mnemonic='model.mnemonic'
```

svy-attributes (Design | object) - sets provided attributes on the html element being applied to - added in 2020.09

```
svy-attributes='model.attributes'
```

sablo-tabseq(Runtime | tabseq) - a nice way to control client side tabIndexes. See sablo-tabseq page for more info.

```
svy-tabseq='model.tabSeq'
```

svy-format(Runtime | {for:'dataProviderID' , type:'format'}) - requires to be placed on input fields with an ng-model . Registers parsers and forrmatters for ng-model controller to parse and format based on the provided servoy format;

```
svy-format="model.format"
```

svy-component-wrapper(Design | web component) - can be used to easily integrate a child component directive based on a 'component' type property value. For more information see Component (child) property type.

```
<svy-component-wrapper component-property-value="model.childComponent2"></svy-component-wrapper>
    or
<svy-component-wrapper
     tagname="model.childComponent1.componentDirectiveName"
     name="model.childComponent1.name"
     svy-model="model.childComponent1.model"
     svy-api="model.childComponent1.api"
     svy-handlers="model.childComponent1.handlers"
     svy-servoyApi="model.childComponent1.servoyApi">
</svy-component-wrapper>
```

Besides these directives there are also some angular filters:

mnemonicletterFilter - should be applied to an expression bound to an 'ng-bind-html' to underline the first letter coresponding to the filter parameter parameter example :

```
ng-bind-html='model.myStringProperty | mnemonicletterFilter:model.mnemonicProp'
```

formatFilter- formats the given expression result with the corresponding servoy format property(it is used when you don't need editing support , for example labels, buttons)

```
ng-bind='model.myStringProperty | formatFilter:model.formatProp'
```

Services

$svyI18NService:

A component or service can ask for the $svyI18NService to get translated messages for 1 or a set of keys, it returns a promise object where the result object in the then() will be the key:value pairs of the keys that where asked for:

I18NService

```
 var x = $svyI18NService.getI18NMessages("servoy.filechooser.button.upload","servoy.filechooser.upload.
addMoreFiles","servoy.filechooser.selected.files","servoy.filechooser.nothing.selected","servoy.filechooser.
button.remove","servoy.filechooser.label.name","servoy.button.cancel")
    x.then(function(result) {
            $scope.i18n_upload = result["servoy.filechooser.button.upload"];
            $scope.i18n_chooseFiles = result["servoy.filechooser.upload.addMoreFiles"];
            $scope.i18n_cancel = result["servoy.button.cancel"];
            $scope.i18n_selectedFiles = result["servoy.filechooser.selected.files"];
            $scope.i18n_nothingSelected = result["servoy.filechooser.nothing.selected"];
            $scope.i18n_remove = result["servoy.filechooser.button.remove"];
            $scope.i18n_name = result["servoy.filechooser.label.name"];
    })
```

$svyProperties:

Used for setting some ui properties to a component: border, alignment, scrollbars, tooltips..

SvyProperties

```
 Object.defineProperty($scope.model,$sabloConstants.modelChangeNotifier, {configurable:true,value:function
(property,value) {
                                switch(property) {
                                        case "borderType":
                                                $svyProperties.setBorder($element,value);
                                                break;
                                        case "background":
                                        case "transparent":
                                                $svyProperties.setCssProperty($element,"backgroundColor",$scope.
model.transparent?"transparent":$scope.model.background);
                                                break;
                                        case "foreground":
                                                $svyProperties.setCssProperty($element,"color",value);
                                                break;
                                        case "fontType":
                                                $svyProperties.setCssProperty($element,"font",value);
                                                break;
                                        case "format":
                                                if (formatState)
                                                        formatState(value);
                                                else formatState = $formatterUtils.createFormatState($element,
$scope, ngModel,true,value);
                                                break;
                                        case "horizontalAlignment":
                                                $svyProperties.setHorizontalAlignment($element,value);
                                                break;
                                        case "enabled":
                                                if (value) $element.removeAttr("disabled");
                                                else $element.attr("disabled","disabled");
                                                break;
                                        case "editable":
                                                if (value) $element.removeAttr("readonly");
                                                else $element.attr("readonly","readonly");
                                                break;
                                        case "placeholderText":
                                                if(value) $element.attr("placeholder",value)
                                                else $element.removeAttr("placeholder");
                                                break;
                                        case "margin":
                                                if (value) $element.css(value);
                                                break;
                                        case "selectOnEnter":
                                                if (value) $svyProperties.addSelectOnEnter($element);
                                                break;
                                        case "styleClass":
                                                if (className) $element.removeClass(className);
                                                className = value;
                                                if(className) $element.addClass(className);
                                                break;
                                }
                        }});
                        $svyProperties.createTooltipState($element, function() { return $scope.model.toolTipText
});
```

Default tooltip comes styled using bootstrap css, thus having a max-width of 200 pixels (for text wrapping). You can override this css property from your solution using the tooltip element id: 'mktipmsg'

$apifunctions:

Some standard re-usable API

APIFunctions

```
$scope.api.getSelectedText = $apifunctions.getSelectedText($element[0]);
$scope.api.setSelection = $apifunctions.setSelection($element[0]);
$scope.api.replaceSelectedText = $apifunctions.replaceSelectedText($element[0]);
$scope.api.selectAll = $apifunctions.selectAll($element[0]);
$scope.api.getWidth = $apifunctions.getWidth($element[0]);
$scope.api.getHeight = $apifunctions.getHeight($element[0]);
$scope.api.getLocationX = $apifunctions.getX($element[0]);
$scope.api.getLocationY = $apifunctions.getY($element[0]);
```

$svyUIProperties:

Get/Set global (application) UI Property

UIProperties

```
var initialDelay = $svyUIProperties.getUIProperty("tooltipInitialDelay");
```

$formatterUtils:

Apply format to a certain ui element

formatterUtils

```
var formatState = null;
var child = $element.children();
var ngModel = child.controller("ngModel");

if($scope.model.inputType == "text") {
   $scope.$watch('model.format', function(){
           if ($scope.model.format)
           {
                   if (formatState)
                           formatState(value);
                   else formatState = $formatterUtils.createFormatState(child, $scope, ngModel,true,$scope.
model.format);
           }
   })
}
```

Model values

Servoy provides a unique css id called **svyMarkupId** in the model object which can be used by the component to set its main css id to a page unique value.