Array property type

Purpose of this property type

The array property type can be used by web components to hold a variable number of same-property-type values.

Such array types are able to **send granular updates between server and client**. For example if you change from Rhino one element's value (or for more complex element property types - such as 'component' type - if something changes in only one of the array elements) then only the change is sent over to the browser/client instead of the whole array (and the other way around - a change on client doesn't send the full array to server). In the future array types might support granular updates for add/remove element operations as well but for now those operations will result in the full value being sent.

Array types are simply defined in .spec files using ***'[]'*** suffix.

.spec file

```
    "model": {
        (...)
        "myIntArray": "int[]",
        "myStringArray": "string[]",
        "myCustomTypeArray": "person[]",
               (...)
    },
    (...)
    "types": {
        "person": {
          "firstName": "string",
          "lastName": "string",
          "photo": "dataprovider"
        }
    }
```

Array types also allow configuration options to be specified for the element type (defaults/values/tags/... + some types support type specific configuration options in .spec file) using ***'elementConfig'***.

For example when you would have only one 'component' typed property in the component model, you could link it to a foundset like this (more info in 'component' property type):

```
    "model":
    {
        (...)
        "myFoundset" : "foundset",
        "childElement" : { "type" : "component", "forFoundset": "myFoundset" },
        (...)
    }
```

But if you want to have an array of 'component' values that are all linked to a foundset you can use array specific configuration option ***'elementConfig'*** like this

```
    "model":
    {
        (...)
        "myFoundset" : "foundset",
        "childElements" : { "type" : "component[]", "elementConfig" : { "forFoundset": "myFoundset" } },
        (...)
    }
```

Advanced .spec options

Another configuration option (that you will most likely never need) for arrays (available starting with Servoy 2019.06) is to be able to skip null design time element values completely at runtime (**"skipNullItemsAtRuntime": true**):

```
"arraySkipNullsAtRuntime": { "type" : "string[]", "skipNullItemsAtRuntime": true }
```

For example if in developer properties view you set on an array property ["a", "b", null, "c"] then, at runtime, the property will have ["a", "b", "c"] (it skips the null). You will probably never need to use this except when you want to create advanced custom components that have arrays of child components (so "type" : "component[]", "skipNullItemsAtRuntime": true) and where the security settings of a form might not allow some of the child components to be visible at runtime depending on the user that logs in. In that case those items in the array would be set to null automatically by Servoy and most of the times in this case you want to just get browser-side the child components that you can show in that array and not worry about nulls.

> ***"pushToServer"*** .spec setting of an array property **is currently automatically inherited by the elements** of that property (so any "pushToServer" setting defined on elements of the array, so in "elementConfig" will be ingnored). That means that for example if you define the array prop. to be "shallow" watched, all it's elements will be shallow watched. If you don't define pushToServer or define "reject" then the elements of that array will not be watched inside the browser and any changes to them will not be sent to the server.
>
> You **should not use** pushToServer: **"deep"** on array property types, as that will actually add a deep watch on the array and any change within an element of that array will be interpreted as an array prop. change and will send the full array value to the server. If you use "shallow" then the changes in array elements are watched anyway (so in the end it is similar to a deep watch), so you normally do not need "deep" anyway; "deep" is meant more for properties of types like 'object' where you can have random JSON content that you want to be deep watched (so it sends it's whole value to the server for any change - so without granular updates).

Browser/client side value

The browser value of such a property is a Javascript array. Depending on it's **pushToServer** setting it can watch for granular changes and send granular changes to server (so for example if only one of it's elements has changed it will only send that change to server).

Server side javascript value

There is one difference though. In order to be able to send fine-grained updates to the client/browser, those values are 'watched'. That means that whenever you assign a completely new javascript array directly to the property (or if you assign a new object/array into one element of it or nested on any level), that new value (reference) you assign will not be 'watched' directly; you have to take/read it back from the property (which will give you an equivalent 'watched' value) before using it further in code. Or you can access the value of the property directly every time, not kept as a reference.

For example:

DO it like this

```
var newPropertyValue = [ 1, 2, 3 ];
// here you assign a new array to the property
elements.myCustomComponent.myArrayProperty = newPropertyValue;
// here you update the reference that you want to use later in code with the 'watched' new value
newPropertyValue = elements.myCustomComponent.myArrayProperty;

(...then later on, maybe during another event handler execution...)

// this modification will be detected because it's using the new 'watched' value you got from elements.
myCustomComponent.myArrayProperty after it was assigned - and the change will be sent to the browser
newPropertyValue[1] = 5;
```

OR like this

```
var newPropertyValue = [ 1, 2, 3 ];
// here you assign a new array to the property
elements.myCustomComponent.myArrayProperty = newPropertyValue;

(...then later on, maybe during another event handler execution...)

// this modification will be detected because it's using the property directly not through 'newPropertyValue'
elements.myCustomComponent.myArrayProperty[1] = 5;
```

DON'T do it like this

```
// DO NOT DO IT LIKE THIS
var newPropertyValue = {};
// here you assign a new array to the property
elements.myCustomComponent.myArrayProperty = newPropertyValue;

(...then later on, maybe during another event handler execution...)

// this will modify the value in newPropertyValue but myCustomComponent.myArrayProperty will not be aware of
that to send changes to client/browser
newPropertyValue[1] = 5;
```

The server side JS value of such a property is a custom implementation that inherits from the Javascript array prototype - so you should be able to use it just like a normal JS array.

Developer handling of array properties

Array properties can be edited at design-time from Servoy Developer's properties view.

If the array property contains in it's configuration object in the .spec file **"droppable": true** and the element types are either custom object or component then drag and drop operations/visual selection can be used as well with the elements of the array in form designer.

Nesting with custom object and array types

Array types can be nested together with custom object types. This allows you to organize your model's properties better. For example (in .spec file):

```
    "model": {
        (...)
        "persons": "person[]",
                (...)
    },
    (...)
    "types": {
        "person": {
          "firstName": "string",
          "lastName": "string",
          "profilePhotos": "dataprovider[]"
        }
    }
```

So the 'persons' property at runtime (client side) could look like this:

```
[
    { "firstName": "John", lastName: "Doe",
      "profilePhotos": [ "https://....", "https://...." ] },
    { "firstName": "Bob", lastName: "Smith",
      "profilePhotos": [ "https://....", "https://...." ] },
    { "firstName": "Jane", lastName: "Doe",
      "profilePhotos": [ "https://....", "https://...." ] },
]
```