Angular services

Services are similar to WebComponents except they have no user interface. They are mapped in solution code to the "**plugins"** scope - in order to be able to call their api from scripting.

A service must contain at least the **specification file** and the client-side **js file**.

The **service specification file is mostly the same as for web components**. See the specification (.spec file) section. However, as there is no UI, *there is no support for handlers*. Another difference is that *default values* and *initial values* for model properties *are currently ignored* for services.

It is recommended to have the suffix "services" in your service *package name* (for example default servoy services are found into "servoyservices" directory) - to avoid naming collisions between component and service packages. Furthermore, in order to avoid naming collisions among services themselves, a new service name should adhere to the same naming convention as WebComponents.

A service can get it's client side scope from the *$services* angular service/factory that the system provides. On that you can ask for the service scope:

```
$services.getServiceScope(serviceName);
```

This returns an angular $scope instance; that one holds a model property that is the javascript object that is kept in sync between server and client (just as it happens for a webcomponent). That scope instance can also be used to add watches on its own properties - so that the service can interact with state changes from the server. This is for example very handy if the service must interact with a browser refresh; then the state is transferred over to the client and the service should use the model state of the scope to reconstruct it's behavior. Watches can be added in an angular run function that is executed when the browser page loads.

When a service scope (the model object) is changed by a server push-to-client, or when a service api function is called, the system will call the angular $digest() function on the scope object of the service. This way all the watches that are on that service scope will be evaluated by angular. If your service can be used throughout the whole page - so webcomponents are using your service to get state from it (webcomponent do have watches on your service) then you have to make sure that you call the rootScope digest so that a full digest cycle will happen:

```
if (!$rootScope.$$phase) $rootScope.$digest();
```

The if is for checking if there is already a digest cycle happening, else you call the $rootScope $digest method so that all the watches of the page are evaluated and webcomponents or other services that have watches on your state will see the change.

Services can also have **server side api** - just like webcomponents, The same kind of object structure is then also provided, so there is a $scope object which has a model property that is the object that is synced between server and client.

An example service:

testservice.spec

```
{
        "name": "mypackage-testservice",
        "displayName": "Test service that says helloworld",
        "definition": "servoyservices/testservice/testservice.js",
        "libraries": [],
        "model":
        {
                "text": "string"
        },
        "api":
        {
                 "talk": {
                },
                "helloworld":
                   {
                    "parameters":
                    [
                            {
                                    "name":"text",
                                    "type":"string"
                            }
                        ]
                }
        }
}
```

The service js file must define the api from the spec:

testservice.js

```
angular.module('mypackageTestservice',['servoy'])
.factory("mypackageTestservice",function($window,$services) {
        var scope= $services.getServiceScope('mypackageTestservice');
        return {
                talk: function() {
                        alert("talk: " + scope.model.text);
                        scope.model.text = "something else"
                },
                /**
                 * Say hello.
                 * @param {string} name your name
                 */
                helloworld: function(name) {
                        alert("Hello " + name);
                }
        }
})
.run(function($rootScope,$services)
{
        var scope  = $services.getServiceScope('mypackageTestservice');
    // watch the whole model (you can also use 'model.text' to only watch the text property)
        scope.$watch('model', function(newvalue,oldvalue) {

                // handle state changes
        }, true);
})
```

From scripting, when calling plugins.testservice.talk() it should execute the service talk method. The service model is automatically synchronized with the server. In order to observe server side modifications the service must add a watch to the service state.

A service can have a **method called "cleanup"** that will be called when solution is closed in order to clear its state.

It is possible to use TypeScript for writing the service code, see How to use TypeScript for Web Package projects