Sokratis Karkalas – 2/3/2015

Widget - Platform Communication

Interface

This text refers to the method of communication between a widget that resides in its own web page (possibly coming from a different domain) and the enclosing web page that hosts it. Integration of the two takes place through an <iframe>. We call the - foreign - widget 'guest' and the enclosing page 'host'.

The basic interface at that level comprises two functions:

- sendMessage(message)
- receiveMessage(event)

This interface provides the ability for two-way communication between the host and the guest. In both cases the data is sent in the form of a 'message' object. The only difference is that in the latter case the message is received as a property of an event object.

The format of the 'message' object follows:

content: This property can contain data of any type. The purpose in this case is to send some data and let the receiver decide what to do with it.
command: This is an instruction (command) that is possibly sent along with some data in the form of arguments. The intention in this case is to utilise the receiver's-specific functionality and perform some processing there. This functionality is exposed in the form of a public interface that the receiver makes available to other communicating parties. The command can only be given as a 'string'.
args: This is an array of values that accompany the command. These values are addressed to a method of the receiver's public interface. That method is what the command property refers to.
callback: This is a boolean value indicating whether an answer is required after the completion of the operation the command refers to. If the value is 'true' then the receiver is expected to send back another message with the result.

A message object is not valid if:

- command is not a string
- args is not an array
- callback is not a boolean
- both content and command are not given
- args is given without a command
- callback is given without a command

Widgets by their nature are expected to be very diverse in terms of functionality. As a consequence of that their interfaces are expected to be heterogeneous. On the other hand platform - widget interoperability should be based on a standardised uniform way of communication. The solution to this problem is to enable interprocess communication through message objects. Instead of extending the basic interface presented above with more functions we pass method names and parameters as properties of messages. This way we keep a simple and uniform interface that facilitates basic communication for any type of widget and at the same time we accommodate the utilisation of the widgets' particular functionality without constraints.

According to the above system every widget (or communicating party, ex. platform) must provide

an object called **publicIF** that exposes its particular functionality to its host. This functionality can be utilised locally by calling this object's public member methods. The same methods can also be called remotely through an executor method that is provided by the same object.

If, for example, there is a method called add(a, b) that performs addition, this method can be called in two ways:

locally: **publicIF.add(a, b)**
remotely: **publicIF.execute({'command':'add', 'args':[a, b]}, [callback])**

The first argument of the executor is the message object described above. The second is optional and may be a reference to the function **sendMessage(message)** that will return the resulting value to the caller.

Protocol

The communication protocol for widget – platform communication doesn't have to be particularly complex. There are three communication scenarios that can possibly take place:

1. The guest wants to register itself with the host. This message is supposed to be sent immediately after the guest loads up and is fully functional within its page. The message follows:

content: 'ready'
command: null
args: null
callback: false

2. The guest or the host wants to send a message to the other party without any instruction as to what the receiver should do with it. This is a simple message with some content (like the previous one).

content: 'some content'
command: null
args: null
callback: false

3. The guest or the host wants to use a service provided by the other party. That entails the execution of a remote method. An answer may be required as well. The message would look like the following in this case:

content: null
command: 'add'
args: [2,3,4]
callback: true

The receiver is expected to perform the operation and return the result in another message.

4. The guest or the host wants to send some data and instruct the receiver explicitly what to do with the data. That, again, entails the execution of a remote method. The data is send as an argument for the remote method to be executed. The message would look like the following in this case:

content: null
command: 'logActions'
args: [{action1},{action1},..,{actionN}]
callback: false