WRT Service API Synchronous and Asynchronous calls

30 Nov 2008

When working with the WRT Service API, you can often choose between a **synchronous** and an **asynchronous** version of the same method. This allows having the same functionality with a slightly different impact on the rest of your code.

This article describes how to handle synchronous and asynchronous calls and explains when the different call types should be used.

Synchronous calls

Behaviour

Synchronous calls require only one argument, a *criteria* object that contains information relevant to the called Service API method. Since the call is synchronous, the JavaScript code execution will actually be **blocked** until the method returns.

Returned values

A synchronous method call returns an object containing 3 properties:

- **ReturnValue**: This is the actual value returned by the called method, so you should check the method definition to check its structure. As an example: calling the IDataSource.GetList() method from the Contacts Service API, this property will hold the actual contacts information. If a method does not need to return any information (e.g., the ILocation.CancelNotification() method from the Location Service API), this property is not included in the returned object.
- **ErrorCode**: A number that specifies an error code. Value zero means no error. The complete error codes table is available here: [1]
- **ErrorMessage**: A message describing the returned error. Error messages are different for each Service API. For example, this is the complete list of error messages related to the Contacts Service API: [2].

Pros and cons

- **PRO: simpler coding**: You get the expected value in the same method call, without the need to implement a separate callback mechanism.
- **CONS: code execution is blocked** until the called method returns: If the required operation takes some seconds to execute, this could be a usability problem.

Asynchronous calls

Behaviour

Asynchronous calls require two arguments:

- The same *criteria* object passed to the synchronous call.
- A *callback*: The method to be called when the method returns.

Being asynchronous, your code continues its natural execution without the need to wait for the called method to return. When the return information is available, the callback method will be called to handle it.

Returned values

The object returned from an asynchronous call is different from the one returned from a synchronous one, and it will contain these properties:

- **TransactionID**: A unique ID that identifies the asynchronous call. You can use it, for example, in your callback method to identify a specific call.
- **ErrorCode**: A number that specifies an error code, as for the synchronous call.
- **ErrorMessage**: A message describing the returned error, as for the synchronous call.

Callback method

The *callback method* is the method that is called when the asynchronous call has ended and the information it returns is available. It must be defined to accept three arguments:

(C) Copyright Nokia 2014. All rights reserved.

- **transId**: The ID of the transaction calling the callback method.
- **eventCode**: A number representing the callback status. Possible values are:
  - **2**: Event completed
  - **4**: Event error
  - **9**: Event in progress
- **result**: The actual object returned by the called method. This is the same object that is returned by synchronous calls, so it will contain the same three properties:
  - **ReturnValue** (optional, depending on called method)
  - **ErrorCode**
  - **ErrorMessage**

Common callback implementations usually involve:

- **transaction ID checking**: To identify the asynchronous call.
- **event code checking**: To check if the event has correctly ended (code 2), is still in progress (code 9), or had an error (code 4).
- **result error checking**: As for the synchronous call, before handling the **ReturnValue** property of the *result* object, you have to check if there were any errors in the method call itself by checking the **ErrorCode** property.

Pros and cons

- **PRO: code execution is not blocked**: You can do something else (e.g., animations, other method calls) while the called method is executed.
- **CONS:** You have to **manually synchronise** your code, defining appropriate actions within callback methods and carefully manage application states and UI.

Exceptions

Not all methods support both synchronous and asynchronous calls.

For example, the IMessaging.GetList() method from the Messaging Service API supports only synchronous calls, while the IDataSource.GetList() method from Media Management Service API can only be asynchronous.

Use case: Retrieving contact list

This sample use case shows how to retrieve the device contact list, with both synchronous and asynchronous calls. Error management is not handled in this example but you must handle it in your widgets.

First, instantiate the Service object:

```
var so = device.getServiceObject("Service.Contact", "IDataSource");
```

Now define the criteria object that will be used to retrieve contacts. This object is the same for both synchronous and asynchronous calls.

```
var criteria = new Object();
criteria.Type = "Contact";
```

Then, define a function that will parse the retrieved contacts. This can be equally defined to work with both calls:

```
function handleContacts(contacts)
{
 try
 {
  contacts.reset();

  var contact;

  while((contact = contacts.getNext()) != undefined)
```


(C) Copyright Nokia 2014. All rights reserved.

```
   {
    //handle current contact
   }
  }
  catch(e)
  {
   alert('Error while showing contacts');
  }
 }
```

Synchronous version

The synchronous version is quite straightforward. You only need to call the GetList() method and pass the returned information to the handleContacts() method:

```
var result = so.IDataSource.GetList(criteria);

handleContacts(result.ReturnValue);
```

Asynchronous version

Different from before, you have to define a callback method that will be called when the asynchronous GetList() call returns. In real applications, you also have to **check both transIds**, to check which method call is actually calling your callback method, **and eventCode** to check if the event is completed (the value must be 2).

```
function contactsCallback(transId, eventCode, result)
{
 handleContacts(result.ReturnValue);
}
```

So, the previous (synchronous) method call will change, taking contactsCallback as the second argument. Also here, you should check the error returned by this call, and store the returned **TransactionID** property that will be used in the callback method.

```
var result = so.IDataSource.GetList(criteria, contactsCallback);
```

Further reading

- WRT Service API reference
- Category:Symbian Web Runtime
- Web Runtime technology landing page

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.